C Coding Standard and Programming Guidelines

Status : August 31, 2009

Version : 1.3.6

Author : Armin Kurt

Table of Contents

1 Introduction

This document contains the guidelines for writing C code for *Minoris Financial Software* and is intended for internal use only. Every developer and/or consultant is expected to be familiar with these guidelines and has to adhere to these.

It is recognized, however, that in some instances it may be necessary to deviate from the rules given in this document. A formal procedure will be used to authorize these deviations rather than an individual programmer having discretion to deviate at will.

Every deviation has to be clearly documented in the comments of an effected module or function so that during code reviews or later maintenance the reasons and the nature of the exceptions will be understood.

1.1 References

The guidelines described below are a compilation of the most reasonable rules (in the author's opinion) for writing readable and structured C code based on the following documents that have been available at the time of writing:

1) Title: *C Style Guide and Programming Guidelines*
   Author: *Peter van der Vlugt*
   Source: *http://www.chris-lott.org/resources/cstyle/Peter_CStyleGuide.pdf*

2) Title: *C Coding Standard*
   Author: *Jean J. Labrosse, (Micrium Inc.)*
   Source: *http://www.validatedsoftware.com/resources/an2000.pdf*

3) Title: *Software and Automation Systems Branch C++ Style Guide*
   Company: *Goddard Space Flight Center*
   Source: *http://aaaprod.gsfc.nasa.gov/WebSite/Files/Cplus/index.html*

4) Title: *Recommended C Style and Coding Standards*
   Author: *several authors*
   Source: *http://www.psgd.org/paul/docs/cstyle/cstyle.htm*

Chapter 3 of this manual (Program Organization) has been almost completely adopted from "*C Style Guide and Programming Guidelines"*. Section 2.3.2 (Spacing) was borrowed entirely from "*C Coding Standard*".

Any copyright and/or intellectual property claims of the authors of the documents mentioned above is recognized expressly.

2 Programming Guidelines

2.1 General Rules

- All code has to be strictly conforming to the ANSI C (ISO 9899) standard.
- No compiler warnings should be left and compiler optimizations should be turned off.
- The code should not contain functions that are not called and there should be no unused variables in the code and also no unrefered `#define`'d constants.
- Lines shall not exceed 101 characters.
- Return types for functions should be always specified. The check of return values of any function call is mandatory, even from functions that "cannot" fail.
- Declare each variable in a separate declaration starting in column 1.
- Any variable whose initial value is important should be explicitly initialized.
- Use only one statement per line.
- It is NOT allowed to use `goto` statements.
- Use full parenthesizing for every expression instead on relying on C's precedence table.
- Use correct type casting in calculations; do not rely on the compiler's interpretation and automatic casting.
- When testing incrementing or decrementing counters, `>=` or `<=` instead of `==` should be used.
- The validity of values passed to C standard library functions shall be checked. If possible, use "wrapped" functions which perform valid parameter checks before calling the original library function.
- Checking all code with PC-Lint is mandatory. Use PC-Lint during and after changes or additions to the code.

2.2 Naming Conventions

Names with leading and trailing underscores are reserved for system purposes and should NOT be used.

In general, all global names except for variables MUST have the common prefix '`MFS_`' (external scope) or '`mfs__`', (internal scope), standing for *Minoris Financial Software*.

2.2.1 Module Names

Header files shall have the extension '`.h`'; source files shall have the extension '`.c`'. Except for the prefix, only lowercase letters should be used.

Files with an external scope, i.e. providing functionality to the outside world, must be prefixed with '`MFS_`'; files internal to the project are prefixed with '`mfs__`':

Header files (external scope):	`MFS_file_name.h`
Source files (external scope):	`MFS_file_name.c`
Header files (internal scope):	`mfs__file_name.h`
Source files (internal scope):	`mfs__file_name.c`

2.2.2 Function Names

Function names in general should be written in lower case where an underscore should be used as a word separator.

Global functions with an external scope must be prefixed with '`MFS_`' or rather with '`mfs__`' if their scope is internal. Functions with local scope are not prefixed:

Global (external scope):	`MFS_func_name`
Global (internal scope):	`mfs__func_name`
Local scope:	`func_name`

Prototypes of global functions shall be declared in header files by using the keyword `extern` and the function parameters shall be described in the function header.

All local functions and their prototypes shall have the keyword `static` in order to keep their scope 'inside'.

2.2.3 Typedefs and Structures

The basic types of `char`, `int`, `short`, `long`, `float` and `double` are replaced by newly defined equivalents according to MISRA-C:2004 rule 6.3:

```
typedef unsigned char   int8u;          /* unsigned  8 bit integer                */
typedef signed   char   int8s;          /* signed    8 bit integer                */
typedef          char   string;         /* plain     8 bit character              */
typedef unsigned short  mfsbool;        /* logical data type (MFS_TRUE/MFS_FALSE) */
typedef unsigned short  int16u;         /* unsigned 16 bit integer                */
typedef signed   short  int16s;         /* signed   16 bit integer                */
typedef unsigned long   int32u;         /* unsigned 32 bit integer                */
typedef signed   long   int32s;         /* signed   32 bit integer                */
typedef          float  float32;        /* 32 bit, single precision floating point */
typedef          double float64;        /* 64 bit, double precision floating point */
```

Note: It is NOT allowed to define any types from pointers, e.g. `typedef char* string`

Fields in a type definition structure have to be prefixed with a data type specifier (see chapter 2.2.4); the name of the type also has to be prefixed (if global) and always ends with ‘Type’:

```
typedef struct
{
  int32s s32Year;
  int32s s32Month;
  int32s s32Day;
}
mfs__OleDataType;
```

Values in an enumerated type should be defined in capitals. The values of the constants shall be assigned explicitly and, in case of a global scope, the name of the type has to be prefixed:

```
typedef enum
{
  ICMA        = 0,
  SIFMA       = 1,
  US_TREASURY = 2,
  MOOSMUELLER = 3
}
MFS_YieldMethod;
```

2.2.4 Variables and Function Parameters

The readability of variables and function parameters is supported by using capitalizing. Depending on the data type, the whole name has to be prefixed as follows (except for counter variables like `i`, `j`, `k`, `l`):

Type	Specifier
int8u	u8
int8s	s8
string	str
mfsbool	b
int16u	u16
int16s	s16
int32u	u32
int32s	s32
float32	f32
float64	f64
structure	t

Variables with global scope are prefixed with a scope specifier:

Global (external scope):	g (i.e. gf32VarName)
Global (internal scope / module level):	m (i.e. ms16VarName)
Local scope and function parameters:	[none]

Pointer variables are specified as follows:

Pointer specifier:	p (i.e. *pf64VarName)

Arrays are specified as follows:

Array specifier:	`a (i.e. as32VarName)`

The naming convention of variables is specified as follows:

Sequence of specifiers:	`<scope><pointer><array><type>VarName`

All variables with internal scope at module level must be declared `static`. Exported variables must be specified by using the keyword `extern`.

2.2.5 Macros

Macros are always given in capitals and the words are separated by an underscore:

General macros:	`MACRO_DEFINITION`
MFS-specific with global scope:	`MFS_MACRO_DEFINITION`

Try to avoid function-like macros according to MISRA-C:2004 rule 19.7. Functions should be generally preferred as they provide a safer calling mechanism.

2.3 White Space

Adding white space in the form of blank lines, spaces, and indentation significantly improves the readability of the code.

2.3.1 Blank Lines

Make careful use of blank lines to separate logical code structures. There should always be two empty lines before a new function declaration starts in order to separate the functions properly.

2.3.2 Spacing

Never use spaces around the following primary operators:

`->`	structure pointer operator	`p->member`
`.`	structure member operator	`s.member`
`[]`	array subscripting	`a[i]`

Always use a space between a function name and the opening parenthesis when declaring or prototyping a function:

```
int16s func_name (void)
```

Do not use a space between a function name and the opening parenthesis when calling a function:

```
func_name();
```

Commas and semicolons are never preceded but always followed by a space:

```
func_name(x, y, z);
for (i = 0; i < 10; i++)
```

Binary operators and the ternary operator are always written with a space between them and their operands:

```
x + y = z;
i += 2;
n > 0 ? n : -n;
a < b
c >= 2
```

Never use spaces between unary operators and their operands:

```
!value
~bits
++i
j--
(int32s)x
*ptr
&x
sizeof(x)
```

Expressions within parentheses or argument lists are written with no space after the opening parenthesis and no space before the closing parenthesis:

```
func_name(x, y, z);
x = (a + b) * c;
```

Keywords are always followed by one space character:

```
if (a > b)
while (x > 0)
for (i = 0; i < 10; i++)
switch (x)
return (y)
```

2.3.3 Indentation

Indentation should be used to show the logical structure of the code. TAB characters (ASCII character 0x09) MUST NOT be used as different editors may use different tab sizes causing different indentation levels.

Indentation MUST be done using the SPACE character (ASCII character 0x20), where the indent level of the code is always 2 spaces with the exception of variable definitions, which always start in column 1.

All functions should be indented like this:

```
<return type> <function_name> (<args>)
{
<variable definitions>

  <body_1>
  {
    <body_2>
    {
      <body_3>
    }
  }
}
```

2.3.4 Continuation Lines

Operations that will not fit on one line should be divided into separate lines, putting the operator AT THE END of the broken line rather than at the start of the continuation line.

The indentation should be properly related to the logical context with regard to parentheses and/or operations:

```
if ((<condition_1> || <condition_2>) &&
    (<condition_3> || <condition_4>) &&
    (<condition_5>))
{
  <var_1> = (<var_2> + <var_3> + <var_4>) /
            (<var_5> - <var_6>);
}
```

2.3.5 Bracing Style

Braces shall follow the Braces-Stand-Alone method, also called „Extended Bracing Style“. The curly brace pair should be placed on separate lines aligned with the surrounding statement:

```
int16s func_name (int32s s32a, int32s s32b, int32s *ps32Result)
{
int16s s16retval;

  s16retval = MFS_ERROR;

  if (s32a > 0)
  {
    *ps32Result = s32a;
    s16retval = MFS_OK;
  }
  else
  {
    *ps32Result = s32b;
    s16retval = MFS_OK;
  }

  return s16retval;
}
```

(Please note that the „pointer“ qualifier ‘*’ is placed next to the parameter/variable name.)

2.4 Comments

- All comments have to be in English.
- Always use C style comments (`/* */`); C++ style comments (`//`) are not allowed.
- It is not allowed to have code following a comment on the same line.
- Nested comments are not allowed.
- Commenting out blocks of code is not allowed.
- Use in-line comments to document variable definitions and block comments to describe computation processes.
- Every function shall have a comment that describes its purpose

2.4.1 Block Comments

Block comments should be written at the same level of indentation as the code block they describe:

```
/* This is a block comment. The comment should be written in full
 * sentences, with correct punctuation. Use this form of comment
 * if more than one sentence is required. Each comment line should
 * begin with an asterisk and the comment terminator should be at
 * the end of the last comment line. */
```

2.4.2 In-Line Comments

Comments to the right side of a code line should comment only that line. These comments should be separated far enough from the code statements.

If more than one short comment appears in a block of code or data definition, they should all be started from the same column:

```
if (...)
{
  ...;           /* a short in-line comment */
}
else
{
  ...;           /* a little longer in-line comment */
}
```

2.5 Control Structures

Although C does not require braces around single statements, braces MUST also be used for single statement blocks to help improve the readability and maintainability of the code.

If the span of a block is large (about 50 lines or more) or if there are several nested blocks, comment closing braces to indicate what part of the process they limit.

Use an end-line comment also to identify which `#if` or `#ifdef` statement a particular `#endif` statement closes. When using `#else` conditions, mark both the `#else` and the `#endif` statements with the negated condition (i.e. preface it with not).

2.5.1 If-Statement

Always use this coding style for `if`-statements:

```
if (<condition>)
{
  ...;
}
else if (<condition>)
{
  ...;
}
else
{
  ...;
}
```

2.5.2 Switch-Statement

Every `switch` selection statement contains a `default` clause, and every clause is terminated with `break`:

```
switch (<state>)
{
  case CASE_NR_1:
    ...;
  break;

  case CASE_NR_2:
    ...;
  break;

  case CASE_NR_3:
    ...;
  break;

  default:
    ...;
  break;
}
```

2.5.3 For Loop

The `for`-statement looks like this:

```
for (<initializations>; <test conditions>; <increment value>)
{
  ...;
}
```

2.5.4 While Loop

The `while`-statement has to be coded as follows:

```
while (<tested condition is satisfied>)
{
  ...;
}
```

2.5.5 Do…While Loop

Write the `do…while`-statement as stated below:

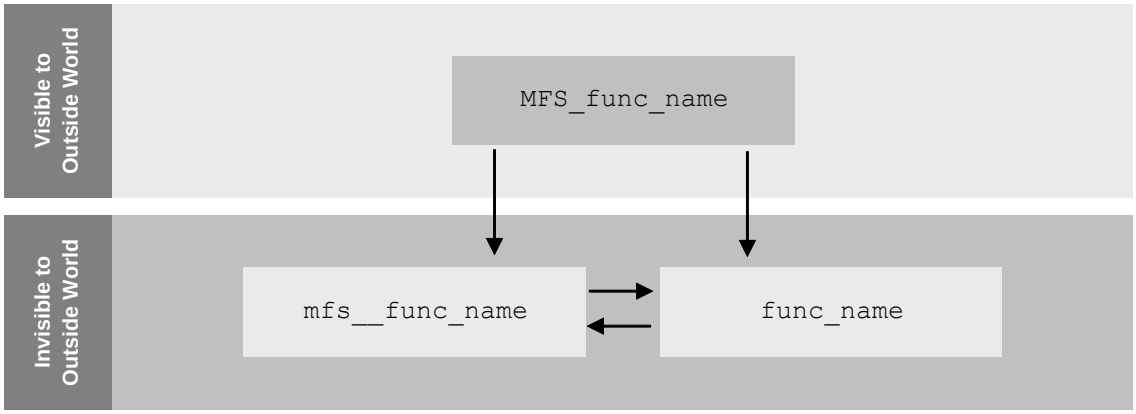
```
do
{
  ...;
}
while (<condition is satisfied>);
```

2.6 Functions

Every function should be preceded by a block comment prologue, giving a short synopsis of the function's purpose and how to use it (for templates please refer to Appendix A).

2.6.1 Scope and Calling Scheme

Functions are classified with regard to their scope. Under consideration of the naming convention given in chapter 2.2.2, the following calling scheme has to be used:



Functions with external scope (`MFS_func_name`) are only wrapper functions performing argument checks to ensure that always valid parameters are passed to internal functions.

2.6.2 Function Results

Every function returns an error status code as the function's result (0 if succeeding). The function result is provided in an out-mode parameter.

This is, however, not necessary for functions returning `void`.

2.6.3 Function Parameters

Structures should not be passed to parameters; a pointer to the structure should be used instead.

Use `const` for pointer parameters if they have to stay unchanged.

2.6.4 Error Messages

If a function returns an error indication and/or a call to that function implies a call to any sub-function which returns error information for its part (regardless of whether the sub-function belongs to the project or not), then the function being called originally should provide an error message in an out-mode parameter at the very end of the parameter list.

Functions with internal or local scope (`mfs__func_name`, `func_name`) provide two types of error messages depending on whether the error is defined or undefined.

Defined errors are displayed as text strings describing why a function call couldn't be performed successfully.

An undefined error occurs if a function call fails. An error message is being generated starting with "`#Internal error:`" followed by the name of the function which induced the error and, given in parentheses, the name of the module and the line number where the error has been detected:

Defined error:	"<error message>" (i.e. "Computation is not possible")
Undefined error:	"#Internal error: <func_name> (<file_name>:<line_number>)"

Functions with external scope (`MFS_func_name`) return error messages either in the way as stated above (as a result of an internal function call) or in the following way in case the error message was caused by a parameter check:

```
"<parameter> is not valid"
```

3 Program Organization

In this chapter the source (`.c`) and header (`.h`) file layout will be introduced and explained. Both types of files are laid out in a similar fashion as shown below and every module that is being created has to comply with that layout.

For templates of source and header files please refer to Appendix B.

The templates also contain specific compiler directives which are necessary to allow multiple inclusions of header files by the application modules and to allow inclusion of a header by its own C-source file.

3.1 Header File Layout

Every header file consists of the following sections:

- File Header
- Version History
- Modules used
- Definitions and Macros
- Typedefs and Structures
- Exported Variables
- Exported Functions
- EOF

3.1.1 File Header

General information describing the file will be found in the file header, where the following items must be present:

- Project: A short description of the project's or product's name where the module or file has been created for.
- Filename: The filename shall be treated as case sensitive. It is not allowed to add drives or directories.
- Version: The current version number of the file that keeps track with the number in the version control database.
- Date: The date of change when the last version number was created.
- Copyright: Each year a version of the file has been created in should be put in the copyright notice followed by the company's name.

3.1.2 Version History

The version history overview consists of a block of comments describing each change in chronological order starting with the 'Original Version' as the first block. For each new version of the file a block has to be added to the version history list, where the last block describes the current file version. The following items must be present:

- Version: The version number for the changes described.
- Date: The date of change to this version.
- Revised by: Responsible person for this version.
- Description: For the first version 'Original Version' has to be put here. In the subsequent revisions the description must exactly reflect the changes done to the file. The changes need to be documented as clearly as possible.

3.1.3 Modules Used

Under this section all modules that are used have to be listed using the `#include` statement. C-standard header files have to be included first and, separated by an empty line, local include files will be listed after this.

It is recommended to list all files being included in an alphabetical order.

3.1.4 Definitions and Macros

Under this section all definitions and macros which have to be known outside the modules have to be listed. Logical groups of definitions and macros have to be ordered together by always starting in column 1.

The different groups can be separated by using proper comment blocks and empty lines.

3.1.5 Typedefs and Structures

Type definitions with a global scope are placed under this section by following the rules defined in chapter 2.2.3.

3.1.6 Exported Variables

In this section all exported variables have to be declared by using the keyword `extern`.

3.1.7 Exported Functions

In this section the exported functions of the modules are placed with their function prototypes. A comment block has to be added that gives a good description of the meaning and use of the functions and their arguments.

3.1.8 EOF

This is the end of file marker section. Nothing should appear below this block.

3.2 Source File Layout

Most of the sections in the source file are the same as already described for header files. Extra sections are: *Prototypes of Local Functions*, *Global Variables* and *Local Functions*. The sequence of sections is as follows:

- File Header
- Version History
- Modules used
- Definitions and Macros
- Typedefs and Structures
- Prototypes of Local Functions
- Exported Variables
- Global Variables
- Exported Functions
- Local Functions
- EOF

3.2.1 File Header

Refer to chapter 3.1.1 for a detailed description of this section.

3.2.2 Version History

Refer to chapter 3.1.2 for a detailed description of this section.

3.2.3 Modules Used

Refer to chapter 3.1.3 for a detailed description of this section.

3.2.4 Definitions and Macros

Refer to chapter 3.1.4 for a detailed description of this section.

3.2.5 Typedefs and Structures

Refer to chapter 3.1.5 for a detailed description of this section.

3.2.6 Prototypes of Local Functions

In this section the prototypes of the local functions of the source file are placed with the keyword `static`.

3.2.7 Exported Variables

In this section all exported variables have to be declared WITHOUT the keywords `static` or `extern`. By omitting `static`, the variable is exported or public by default.

3.2.8 Global Variables

In this section all global variables are declared with the keyword `static`. The scope of these variables is global to the source file.

The variable names should be placed one below the other, preferably left aligned in the same column and on the same indent position. The declaration should start at the left margin.

3.2.9 Exported Functions

In this section the implementations of the exported functions of the module are placed.

3.2.10 Local Functions

The function declaration and the implementation of local functions are placed under this section in the source file. All functions appearing in this section must have a corresponding function prototype in the section ‘Prototypes of Local Functions’.

3.2.11 EOF

This is the end of file marker section. Nothing should appear below this block.

4 Appendix A

4.1 Comment Prologues of Functions

Functions are preceded by a block comment prologue that gives a short description what the function does and (if necessary) how to use it.

Notice that on the very first line each comment shows a number representing the sequence of the function's appearance within the code, followed by a hint of the function's scope, i.e. exported (exp) or local (loc).

4.1.1 Prologue of Function Prototypes

Template for function prototypes; note that each parameter will be described as well:

```
/*--|10 exp|-------------------------------------------------------------------------------------------
 *  Function:       mfs__year_fraction
 *
 *  Purpose:        Calculates the year fraction as a decimal number representing the number of
 *                  days between two dates.
 *
 *  Comments:       1) Function returns failure if a NULL pointer is passed to 'pf64Result'.
 *                  2) Calling function has to pass space for 'pstrErr' and needs to take
 *                     care of valid parameter inputs.
 *
 *  Returns:        MFS_OK     (0)        Success
 *                  MFS_ERROR (-1)        Failure
 *-------------------------------------------------------------------------------------------------*/
extern int16s mfs__year_fraction (
              int32s s32StartDate,              /* IN:  OLE-compliant serial day number           */
              int32s s32EndDate,                /* IN:  OLE-compliant serial day number           */
              int16s s16DayCount,               /* IN:  0=30/360(ICMA), 1=30/360(SIFMA),
                                                 *      2=act/360, 3=act/365, 4=act/act           */
              mfsbool bEoM,                     /* IN:  flag for end-of-month convention          */
              float64 *pf64Result,              /* OUT: function result                           */
              string *pstrErr);                 /* OUT: error message if function returns failure  */
/*-------------------------------------------------------------------------------------------------*/
```

4.1.2 Prologue of Function Implementations

Use this template for function implementations:

```
/*--|10 exp|-------------------------------------------------------------------------------------------
 *  Function:       mfs__year_fraction
 *
 *  Purpose:        Calculates the year fraction as a decimal number representing the number
 *                  of days between two dates.
 *
 *  Comments:       None
 *
 *  Date            Developer           Action           Remarks
 *-------------------------------------------------------------------------------------------------
 *  28 Aug 09       Armin Kurt          Created          None
 *-------------------------------------------------------------------------------------------------*/
int16s mfs__year_fraction (int32s s32StartDate, int32s s32EndDate, int16s s16DayCount, mfsbool bEoM,
                           float64 *pf64Result, string *pstrErr)
{
  /* function body left out */
}
```

5 Appendix B

5.1 Header File Template

```
/*
                                          ******************
********************************************* C HEADER FILE *********************************************
**                                          ******************                                           **
**                                                                                                       **
** project   : xxx                                                                                       **
** filename  : (...).h                                                                                   **
** version   : xxx                                                                                       **
** date      : xxx                                                                                       **
**                                                                                                       **
***********************************************************************************************************
**                                                                                                       **
** Copyright (c) yyyy Minoris Financial Software                                                         **
** All rights reserved.                                                                                  **
**                                                                                                       **
***********************************************************************************************************

VERSION HISTORY
---------------

Version     : xxx
Date        : xxx
Revised by  : xxx
Description : xxx

*/

#ifndef _(...)_INCLUDED
#define _(...)_INCLUDED

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                             MODULES USED                                              **/
/**                                                                                                       **/
/***********************************************************************************************************/

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                        DEFINITIONS AND MACROS                                         **/
/**                                                                                                       **/
/***********************************************************************************************************/

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                        TYPEDEFS AND STRUCTURES                                        **/
/**                                                                                                       **/
/***********************************************************************************************************/

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                         EXPORTED VARIABLES                                            **/
/**                                                                                                       **/
/***********************************************************************************************************/

#ifndef _(...)_C_SRC
#endif

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                         EXPORTED FUNCTIONS                                            **/
/**                                                                                                       **/
/***********************************************************************************************************/

#endif  /** _(...)_INCLUDED **/

/***********************************************************************************************************/
/**                                                                                                       **/
/**                                            END OF FILE                                                **/
/**                                                                                                       **/
/***********************************************************************************************************/
```

5.2 Source File Template

```
/*
                                          ******************
********************************************* C SOURCE FILE ********************************************
**                                        ******************                                            **
**                                                                                                      **
** project   : xxx                                                                                      **
** filename  : (...).c                                                                                  **
** version   : xxx                                                                                      **
** date      : xxx                                                                                      **
**                                                                                                      **
**************************************************************************************************************
**                                                                                                      **
** Copyright (c) yyyy Minoris Financial Software                                                        **
** All rights reserved.                                                                                 **
**                                                                                                      **
**************************************************************************************************************

VERSION HISTORY
---------------

Version     : xxx
Date        : xxx
Revised by  : xxx
Description : xxx

*/

#define _(...)_C_SRC

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                          MODULES USED                                                   **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                       DEFINITIONS AND MACROS                                            **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                       TYPEDEFS AND STRUCTURES                                           **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                    PROTOTYPES OF LOCAL FUNCTIONS                                        **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                        EXPORTED VARIABLES                                               **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                         GLOBAL VARIABLES                                                **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                        EXPORTED FUNCTIONS                                               **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                          LOCAL FUNCTIONS                                                **/
/**                                                                                                         **/
/*************************************************************************************************************/

/*************************************************************************************************************/
/**                                                                                                         **/
/**                                           END OF FILE                                                   **/
/**                                                                                                         **/
/*************************************************************************************************************/
```