EMBEDDED
SYSTEMS
ACADEMY

Software Style Guide

Version 1.13

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED
SYSTEMS
ACADEMY

Contents

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

0. History

Version 1.13 – AA – modified hexadecimal number example

Version 1.12 – AA – cleaned up, formatted, all notes consolidated

Version 1.10 – OP – notes added

Version 1.02 – AA and CK – notes added

Version 1.01 – PL and OP – notes added

Version 1.00 – AA – first version

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED
SYSTEMS
ACADEMY

1. Introduction

This document describes the styles Embedded Systems Academy, Inc. and Embedded Systems Academy GmbH will recommend for use to author software for internal and external use.

The nature of programming embedded systems may sometimes require deviating from the recommendations set in this document, especially small example programs based on a single or just a few modules do not require to be implemented following this guide. However, if a piece of code gets published, it is strongly suggested to use the recommendations set forth in this document as much as applicable.

As is, the document covers software aimed at embedded firmware. If during projects we realize that variations from these styles are required either by application, customer or different SW environment (like C++ on a windows style OS), we may introduce variations of these styles for these purposes.

These recommendations are based on the C programming language. They can and should be used with other languages where possible.

If a programmer violates a recommendation, he/she must be prepared to "reasonably justify" why the recommendation should not be used in this case.

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

2. Definitions

Routine	a block of code that is invoked for a single purpose
Function	a routine that returns a value
Procedure	a routine that does not return a value
Module	a collection of data and routines that act on the data

EMBEDDED SYSTEMS ACADEMY

3. Comments

3.1 General

- Comments may appear either in blocks or single lines.
- Comments may be indicated using any keywords or symbols provided by the programming language for the purpose of writing comments.
- In the case of C it is recommended that comments ordinarily are indicated using "//" and not "/*" "*/". If comments are indicated using "//" then it makes it easy to comment out whole blocks of code including comments using "/*" and "*/".
- Comment blocks should be formatted as follows:

```
/******************************
  comments
*******************************/
```

  Do not use the following style or styles like it as it is too time consuming to maintain:

```
/*******************************
 * comments                    *
 *******************************/
```

- Comments should be in English
- Use English-like statements that precisely describe specific operations
- Avoid syntactic elements from the target programming language, i.e. comments are a design at a slightly higher level than the code itself.
- Write comments at the level of intent. I.e. describe the meaning of the approach rather than how the approach will be implemented.
- Generating code from comments should be nearly automatic.
- Comments should always precede the code they comment.
  Exception: #ifdef statements are not regarded as "code" – here the comment must be located within the blocks set by #ifdef/#endif. There are pre-processing and editing tools (like the Development Assistant-C) that automatically eliminate or color highlight code blocks affected by current #define and #ifdef/#endif settings.

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

For this reason, any comment belonging to a code block must be within the #ifdef/#endif statements.

Example:

```
#ifdef NMT_HEARTBEAT
// If heartbeat support is required, we have to initiate
// this and that
…
#else
// If heartbeat is not used, we automatically fall back to
// Node Guarding and initiate that in this
…
#endif // NMT_HEARTBEAT
```

- Comments at the end of a line are an exception, see the section on Endline Comments.
- When writing hexadecimal numbers in comments use the "0x" prefix rather than "H" or "h" suffixes. Hexadecimal values should be in uppercase.
- Date formats in comments are always written as DD-MMM-YY, where DD is a 2 digit number for the date, MMM is JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC and YY is a 2 digit number for the year.
  This format is chosen to avoid potential misinterpretations from DD/MM/YY vs. MM/DD/YY formats.

3.2 Functions

Each function must have a header comment (which is a comment block) before the first line of the function.

The header comment should contain:

- What the function does
- Any global data the function modifies, and how
- Interface assumptions – assumptions about the state of global variables, input data, etc, if needed.
- Changes history if any
- Limitations if any

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

Example:

```
/*****************************************************************
DOES:    Searching for a string pattern in another string
GLOBALS: This function does not read or write global variables
RETURNS: location in str1 where str2 was found, or 255 if not
found
*****************************************************************/
unsigned char STR_FindStringInAString
   (
  char[] str1,        // String in which we search
  char[] str2,        // The string we search to find in str1
  unsigned char start // At this element in str1 we start the
                      // search
  )
{
  int local;
}
```

Example:

```
/*******************************************************
DESC: Prints a string to the output if in debug mode. Returns
whether it printed or not.
GLOBALS: Reads gDebug. Does not modify gDebug.
ASSUMPTIONS: Assumes gDebug is 0 or 1.
HISTORY: Fixed bug so that string is actually printed, AA, 05-OCT-
01
LIMITATIONS: Strings must be less than 40 chars in length
*******************************************************/
int Output_PrintString
   (
  char *pdebugstring  // pointer to string to output
  );
```

3.3 Ends of Blocks

Each function should have the function name in a comment immediately following the close of the function, on the same line. For example

```
void foo(void)
{
} // foo
```

The end of each control structure should have a comment immediately following the close of the block. For example

```
// copy input field up to end of string
while (*Val)
{
} // while – copy input field
```

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

3.4 Endline Comments

Endline comments, i.e. comments that are placed on the end of a line of code may only be used in the following situations:

- Annotate data declarations

```
int PointCount; // count of number of points
```

- Maintenance notes for single lines. Example:

```
A = 0x4A; // changed SFR value from 0x45 to 0x4A to turn on
                                  interrupts, AA 5-OCT-01
```

- Mark the ends of blocks (for example the ends of functions as described above)

3.5 File Headers

Each file should start with a block comment containing:

- Name of file/module
- Description of file/module
- Version
- Version history
- Copyright and web page

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

Example:

```
/***************************************************************************
MODULE:     STR
CONTAINS:   Functions working on strings
COPYRIGHT:  Embedded Systems Academy, Inc. - www.esacademy.com
            This software was written in accordance to the guidelines at
            www.esacademy.com/software/softwarestyleguide.pdf
VERSION:    1.0, PF 30-OCT-01
---------------------------------------------------------------------------
HISTORY:    0.1, PF 02-JUN-01, Initial implementation
            0.2, AA 05-JUL-01, Removed all bugs
            0.3, PF 06-JUL-01, Added function STR_IsThisStringEmpty
            1.0, PF 30-OCT-01, First release and publication
***************************************************************************/
```

3.6 Maintenance Notes

Maintenance notes do not have to be given for every modified line of code, but one maintenance note may be given for a block of changed code. Maintenance notes should include the following:

- Change made
- Reason for change
- Initials of person making the change
- Date of change

Example:

```
// changed ISR for CAN message object 15, AA 05-OCT-01
```

Common sense should be used with maintenance notes, for example it may not be desirable to include them in some publicly released code.

A maintenance note is assumed to only affect the next logical block. If the note is in front of a function, it may affect the entire function. If it is in front of any { } block, it only affects this block.

In case that it is not obvious where the code affected ends, an end-of-maintenance comment needs to be inserted at the appropriate place.

EMBEDDED
SYSTEMS
ACADEMY

Example:

```
// End-of-maintenance, AA 05-OCT-01
```

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

4. Function and Procedure Names

Function and procedures should have a clear, unambiguous names and that start with the module name followed by an underscore. If the module name is longer than 6 characters, an abbreviation may be used. It must match any module name abbreviation used for preprocessor constants.

When objects (in an OO language) are not being used, procedure names should have the form:

Verb followed by object

For example: Init_PrintReport, Reg_InitializeUart, Core_FormatAndPrintOutput.

When objects are being used, procedure names should be a verb. For example: Output_Report.Print.

Function names should describe the return value. For example: Color_CurrentPenColor.

In this context, the following words are to be used in their "common sense" interpretation:

Init, Put, Push, Pop, Handle, Read, Write

"Is" is used for functions that in the essence return a Boolean value.

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED
SYSTEMS
ACADEMY

5. Debug Code

A symbol containing "DEBUG" in the name should be used to include debug code in a program. The symbol must be defined to 0 for no debug code or a non-zero value for the inclusion of debug code. For example (from module COM.c):

```
#define COM_DEBUG 1

#if (COM_DEBUG)
…
#endif

#define COM_DEBUG_UART 2

#if (COM_DEBUG_UART > 1)
…
#endif
```

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

6. Parameters

Parameters passed to a function or procedure should be listed with the parameters that the routine modifies first, followed by the parameters that the routine does not modify.

For example:

```
void foo
  (
  char *pType,     // data pointer to XYZ
  char TypeLength // length of the data type
  )
{
// modification of data pointed to by pType
}
```

Exception:

Pointers to error or status information should be listed last in the list of parameters, as their functionality is only incidental to the operation of the routine.

If several routines use the same or similar parameters, list the parameters in the same order for the routines.

For example:

```
void foo
  (
  char *pType,
  int length,
  float precision
  );


void bar
  (
  int length,
  float precision
  );
```

The number of parameters passed to a routine should be limited to seven where possible.

50 Airport Parkway
San Jose, CA 95110

Tel. (408) 910-7899
Fax. (408) 249-4473

www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

7. Data Naming

7.1 Loop Indexes

Can have short names like i, j, k, etc. unless they are used outside of the loop, in which case the names must be more descriptive, or if loops are nested.

7.2 Enumerated Types

Ensure that it's clear that members of the type all belong to the same group by using a group prefix or suffix. For example:

COLOR_RED and COLOR_BLUE

7.3 Pre-processor Constants

Pre-processor constant names should appear all capitalized, the first group of letters representing the name of the module where the #define is made. If the module name is longer than 6 chars, a consistent abbreviation may be used. An underscore "_" is inserted between the module name and the name chosen for this particular define.

```
#define HW_LED_ADDRESS 0xFFA0
```

7.4 Types

- All basic C data types may be used.
- This includes "short int", "int" and "long int" which are assumed to be 16 and 32-bit. In case these data types are not supported as such by a compiler, the alternatives INTEGER8, INTEGER16 and INTEGER32 are to be used.
- User-defined types are in all capital letters.
- The following, additional basic data types may be used: BYTE, WORD, DWORD, QWORD

EMBEDDED SYSTEMS ACADEMY

7.5 Other Variables

Use the following Naming Convention (based on the Hungarian Naming Convention – but simplified) for short, descriptive variable names. Variable names are made up of:

[<one or more prefixes>]<descriptive name>

Prefix	Meaning
g	Global Variable
m	Module-level variable
p	Pointer

Note that if the variable is a global, module level or pointer the prefixes must be used, and cannot be omitted.

The descriptive name is a name in both upper and lower case (no underscores) that describes the variable. For example:

pCenter - pointer to variable called center

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

8. Modules

All routines in a module must be related in functionality or the data used.

A module may consist of one or more source files.

Each module should have a header file containing the public functions and variables.

Each source file should have a private header file _<filename>.h, containing the private functions and variables.

Source files outside of the module will only include the public header file. Source files in the same module will include the private header files of the other source files in the module.

The last two paragraphs only apply if modules are being used with multiple source files. The situation where someone maintaining code will want to create a module out of some of the source files will be very unusual, and they can rename header files to do it.

9. Code Layout

9.1 Grouping

Code should be grouped into paragraphs, with each paragraph containing code where the statements accomplish a single task and are related to each other. Paragraphs of code should be separated from other code with blank lines.

9.2 Indentation

Indent code to show the logical structure. Statements should be indented under the line of code to which they are logically subordinate. To increase the amount of code on the screen and reduce the possibility of scrolling, two space indentations should be used.

Format blocks of code with:

```
statement
{
  statements;
}
```

The above formatting should also be used when there is only one statement in the block to reduce mistakes when adding additional lines to the block, for consistency, and to aid in debugging (the expression is not on the same line as another statement).

9.3 Assignment Alignment

Align the equals sign for groups of assignments. For example:

```
EmployeeName   = InputName
EmployeeSalary = InputSalary
EmployeeDate   = InputDate
```

The alignment of the equals sign shows the statements belong together.

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED SYSTEMS ACADEMY

9.4 Splitting

For complicated expressions put separate conditions on separate lines. For example:

```
if ((Var != 'a') &&
    (Var != 'b') &&
    (Var != 'c'))
```

How many conditions to put on a line are left up to the common sense of the person writing the code. They should balance the need to make the software compact but also consider readability and the likelihood that new conditions might be added or some removed.

For function declarations the following multiple line layout should be used:

```
void DrawLine
  (
  short int x1; // x-position of point 1
  short int y1; // y-position of point 1
  short int x2; // x-position of point 2
  short int y2; // y-position of point 2
  );
```

See also the variable declarations section for additional formatting rules.

If assignments are to be split over multiple lines, indent the additional lines after the = operator. For example:

```
Purchases = Purchases + Sales +
            CustomerSales(PurchaseID);
```

9.5 Variable Declarations

Variable declarations should be indented so the variable names line up. This makes scanning the list of variable names easier.

```
int     foo; // Used to fool the device
BYTE    bar; // Pressure taken in
float   baz; // Return value
```

Place only one variable declaration on a single line. This makes the declarations easier to comment and modify and easier to find declaration errors.

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

Group variable declarations by type. For example:

```
short int foo;
short int bar;
float     baz;
float     foo2;
```

For pointer declarations, put the asterisk next to the variable name. For example

```
short int *pfoo;
```

Apart from grouping variables by type, all the rules in this section should also be applied to function declarations, where the parameters to the function are listed. For example:

```
void DrawLine
  (
  int            x1;
  float         *y1;
  unsigned char  x2;
  POINT          y2;
  );
```

9.6 Comments

Comments should be indented the same as their corresponding code. For example

```
// check point is valid or invalid
if (point.valid)
{

  // make point invalid
  point.valid = 0;
}
else
{

  // make point valid
  point.valid = 1;
}
```

Include a single blank line before each comment to improve readability.

EMBEDDED SYSTEMS ACADEMY

9.7 File/Module Layout

Separate each function in a file by two blank lines.

The following order in each file should be used:

1. File/ Module description comment
2. #include files
3. type definitions/variable declarations/constant definitions
4. function prototypes
5. functions

EMBEDDED
SYSTEMS
ACADEMY

10. Miscellaneous

10.1 Number Notation

- Hexadecimal numbers should be written in uppercase.
- Octal notations should be avoided.
- Decimal notation rather than hexadecimal notation should be used as the default, unless the context requires hexadecimal or using hexadecimal makes the intent and understanding of the code easier. For example use hexadecimal for bit masking, memory addressing, etc, but not for day of the month variables.
- Whenever using hex or bin notations, the total length MUST represent the total number of bits or hex digits available in the used data type. For example assign 0x000A to a word instead of 0x0A. This forces the programmer/reviewer to think about data types used.

10.2 Inline Assembly

Avoid inline assembly – put assembly in it's own module

50 Airport Parkway
San Jose, CA 95110
Tel. (408) 910-7899
Fax. (408) 249-4473
www.esacademy.com

EMBEDDED
SYSTEMS
ACADEMY

11. References

1. "Code Complete", Steve C. McConnell, Microsoft Press, 1st ed., 1993, ISBN 1-55615-484-4

2. "How to Write Unmaintainable Code", Roedy Green, 1997-2001, www.mindprod.com/unmain.html