C Coding Standard Guidelines

D:\zigbee_street_light\logo.jpg

Block A - 7th Floor,Safal Profitaire,

Corporate Road, Prahaladnagar,

Ahmedabad-380 015,

Gujarat, India

email: [info@volansys.com](mailto:info@volansys.com)

**Contents**

[Purpose](#30j0zll)

[Scope](#1fob9te)

[Program Organization](#3znysh7)

[Directory Organization](#2et92p0)

[Source and Header files content](#tyjcwt)

[Organization of Header files contents](#3dy6vkm)

[Description of Header File Contents](#1t3h5sf)

[Organization of Source File Contents](#4d34og8)

[Description of Source File Contents](#2s8eyo1)

[Format Style](#17dp8vu)

[Comments](#3rdcrjn)

[Line Length](#26in1rg)

[Indentation](#lnxbz9)

[White Space](#35nkun2)

[Line Continuation](#1ksv4uv)

[The ‘\*’ Character](#44sinio)

[End-of-line](#2jxsxqh)

[Spaces](#z337ya)

[Braces/Scoping](#3j2qqm3)

[Functions](#1y810tw)

[If, Else, For, While](#4i7ojhp)

[Names](#2xcytpi)

[File Names](#1ci93xb)

[Typedefs for structs and enums](#3whwml4)

[Macro Names, #defines and constants](#2bn6wsx)

[Function Names](#qsh70q)

[Function Parameter Names](#3as4poj)

[Variable Names](#1pxezwc)

[Coding Practices](#49x2ik5)

[Parenthesis](#2p2csry)

[Variable Declaration](#147n2zr)

[Constants](#3o7alnk)

[Parameter Passing](#23ckvvd)

[Parameter Checking](#ihv636)

[Return Codes](#32hioqz)

[Enumerated and Predefined Types](#1hmsyys)

[Macros](#41mghml)

[Branching Statements](#2grqrue)

[NULL](#vx1227)

**History**

| Version | Author | Date | Comments |
| --- | --- | --- | --- |
| 0.1 | Amit Solanki | 20-July-2015 | First version |
|  |  |  |  |
|  |  |  |  |

# Purpose

This document presents coding guidelines to be used while developing software in the

C language.

# Scope

All C/C++ source code that is produced at Volansys Technologies Pvt. Ltd. should adhere to

the standards outlined within this document. As with any guidelines, there will be

circumstances where full compliance is not desirable for efficiency, maintainability, or

other reasons.

# Program Organization

## Directory Organization

The source code directory should take the form as shown below:

* common - Contains all module that is shared across different targets
  + common module 1 - Source and private headers of common module 1.
  + common module 2 - Source and private headers of common module 2.
* include - Contains all public header files of common module.
* target 1 - Contains all target specific modules.
  + target 1 module 1 - Contains source file of target 1 module 1.
  + target 1 module 2 - Contains source file of target 1 module 2.
  + include - Contains all public header files of target 1.

## Source and Header files content

A single header file named common.h should be created to define types, constants, and macros that are global in the project. Other header files may be produced for each module. The definition of exported constants, structures, and enumerated types should occur in header files. Use data types defined in stdint.h. Ex. uint8\_t, uint32\_t, int32\_t, etc.

### Organization of Header files contents

Each header file must contain a main comment block, which includes,

the items shown in example below. Each header file also must contain the “guards” which prevent multiple inclusion of a header file as shown below.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Copyright(c) <current\_year>, Volansys Technologies  \*  \* Description:  \* @file code\_style.h  \* @brief (Brief description/summary of the file.)  \*  \* A description of the entire functional block. This description should include  \* design information and details which describe the design in more detail than  \* that found in the requirement.  \*  \* Author - Amit S Solanki  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* History  \*  \* Jul/19/2015, Amit S, Something done to file.  \*  \* Jul/01/2015, Amit S, Created (description)  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #ifndef CODING\_STYLE\_H  #define CODING\_STYLE\_H  /\* Contents Here \*/  #endif /\* '#endif' of CODING\_STYLE\_H \*/ |
| --- |

The contents of a header file should be ordered as follows:

* System include files
* Project include files
* Constants, #defines
* Structures, enumerations, types and externs.
* Function prototypes

Example:

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Includes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <stdio.h>  #include "mydir/myfile.h"  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Defines  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* The number of character transmission times required between serial  messages. \*/  #define INTER\_MESSAGE\_GAP 3  #define DATA\_SIZE 8 /\* Size of data in message \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Structures, Enumerations, Typedefs  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Used in the Inter Processor Communications \*/  typedef struct  {  uint8\_t type; /\* Indicates how to interpret the data field \*/  uint8\_t data[DATA\_SIZE]; /\* Data sent in message \*/  } message\_t;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Global Variables  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* None \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Function Prototypes  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\* Short Description of the function.  \*  \* Detailed description of the function.  \*  \* @param[in] number Explain what is this.  \*  \* @param[out] data Explain what is returned  \*  \* @return Explain what is returned.  \*/  int8\_t SomeFunc(int8\_t number, uint8\_t\* data); |
| --- |

### Description of Header File Contents

The header file should be considered the “user’s manual” for each module. The functions, data structures, and other items should have descriptions. These descriptions should explain what the item is and how it should be used. Comments should include detailed descriptions of data items, relationships of the data to the functions, assumptions about the calling frequency or prerequisites which must be met before calling an operation, and services required/used from other modules. It should do all this WITHOUT making reference to the actual calling function. Information about each function that would be useful to the user, including parameter usage and calling sequence, should be placed with the prototype. When possible, it is best to describe whether a parameter is for input, output, or input/ output in terms of the use of the parameter by the function. Describe function return values in the comment block. All data items should be fully described in terms of their usage and what they represent. Any relationship of the data members should also be described as block comments or discussions as if you are trying to explain it to someone else.

Private header files are meant to include all definitions (private prototypes, private constants, private data types, and private #defines) used by a Module and are not meant for use by any other module. This is intended to keep the module file(s) clean.

### Organization of Source File Contents

Each header file that defines functions should have a source file with the

same file name. If a file exceeds the code inspection criteria for length, it may be split into other files. Files exhibiting this behavior are almost always improperly partitioned – i.e. a functional block exhibiting this behavior most likely isn’t decomposed far enough, use helper functions as necessary. If one function acts

as the main interface for the functional block, it should be presented

toward the top of the source file, with subsequent functions being those

called out from that function. The goal here is to improve readability and

maintainability.

### Description of Source File Contents

Each source file should begin with the following type of comment block

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Copyright(c) <current\_year>, Volansys Technologies  \*  \* Description:  \* @file code\_style.c  \* @brief (Brief description/summary of the file.)  \*  \* A description of the entire functional block. This description should include  \* design information and details which describe the design in more detail than  \* that found in the requirement.  \*  \* Author - Amit S Solanki  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* History  \*  \* Jul/19/2015, Amit S, Something done to file.  \*  \* Jul/01/2015, Amit S, Created (description)  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| --- |

Source files are intended to implement, not describe, the constraints of the design. Any comments about the design of the module should be located in the corresponding header file. If necessary, for complex functions, a comment may be added before the function to describe the complex features and implementation of the function. Ordinarily, the function comment is merely a single line comment to provide visual separation between functions. Note that functions are separated by one blank line.

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Func1Simple( void )  {  << … >>  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void Func2Simple( const int row\_number )  {  << complex algorithm or implementation >>  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* As a supplement to the description in the header file, a description about  \* complex behavior or algorithms may be placed in a comment block such as  \* this above a function.  \*  \* Do not duplicate the comments which are already contained in the header file  \* (to prevent a maintenance problem).  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void Func3Complex( const int column\_number )  {  << complex algorithm or implementation >>  } |
| --- |

Additionally, inline comments may be added near statements within each

function to explain logic details and data usage. However, high level

implementation should not be described within the function; the comment

block immediately above the function should be used instead. Refrain

from producing inline comments that are obvious from the item being

commented.

# Format Style

## Comments

Comment language should be kept to a professional level at all times. Offensive or humorous comments have no place in any source file. The C comment delimiters ‘/\* \*/’ should be used to comment. Brief comments regarding individual statements may appear at the end of the same line, and should be vertically aligned with other comments in the vicinity for readability. Lengthy comments documenting major code sections may begin further left but must maintain the code indention level (i.e., must be aligned with the code). Files should not be checked into the version control system with unused code commented out. If code is not needed, it should be removed from the module or placed within conditional compilation directives.

| if (msg\_received == true) {  /\* If a block of comments describing a major section of code is needed,  the block may be aligned with the code like this. \*/  index = ( input \* LINE\_MAX ) + 1; /\* calculate display index  sum the elements of the array  until the limit is exceeded \*/  int sum = 0;  for ( int i = 0; i < ARRAY\_SIZE; i++ ) {  sum += array[i];  if ( sum > limit ) { /\* find index where sum exceeds limit \*/  printf("Sum has exceeded limit on index\n", i);  }  }  } |
| --- |

Use @todo at the start of a comment to call out a section of code that is incomplete.

| if ( msg\_received == true ) {  /\* @todo handle input message \*/  } |
| --- |

## **Line Length**

Line length for executable statements will be limited to 100 characters. Exceptions are permitted for large table/array definitions and comments.

## **Indentation**

Indentation length should be 4 spaces. Verify that your editor is configured to convert tab key presses to the specified number of spaces. Tab characters create viewing problems when different editors are used that view tabs differently. The function type declaration and the function name are on the same line starting in the first column. The opening bracket is on the next line in the first column. The body of the function is indented one level. The closing bracket is in the first column of the next line after the function body. Preprocessor directives will begin at the first column. An indentation style example is given here.

| /\* getline: read a line into s, return length \*/  int32\_t GetLine( uint8\_t s[], int32\_t lim )  {  int32\_t c, i;  for ( i = 0; i < lim-1 && ( c = getchar( ) )!=EOF && c != '\n'; ++i ) {  s[i] = c;  }  if ( c == '\n' ) {  s[i] = c;  ++i;  }  s[i] = '\0';  return i;  } |
| --- |

## **White Space**

Each function definition in a source file should be separated from the next by one blank line. Each group of data definitions in a source or header file should be separated by at least one blank line.

## **Line Continuation**

Macro definitions must use a line continuation character “\” to continue a macro which will not fit in 100 columns. If a line must be continued, break the line at a logical place such as the end of a sub expression with any operators at the beginning of the next line.

| for ( i = 0; i < MAXLINE - 1  && (c = getchar)) != EOF && c != '\n'; ++i ) {  line[i] = c;  } |
| --- |

## **The ‘\*’ Character**

When used in a variable declaration or function prototype, the ‘\*’ character should be placed next to the data type.

| uint8\_t\* byte; |
| --- |

## **End-of-line**

Appropriate editor options should be used such that all files have the end-of-line character sequence native to the development system (e.g., carriage return followed by a line feed for Windows, line feed only for Unix).

## **Spaces**

A space should be placed just after an open parenthesis and just before a closed parenthesis to aid in readability.

| for ( abc = 0; abc < MAX\_LIMIT; abc++ ) {  /\* Do something. \*/  }  if ( exit == TRUE ) {  // TODO: Do Something  }  HandleMessage( msg );  FunctionCall( param1, param2 ); |
| --- |

Spaces should be placed around binary operators and should not be placed between an arithmetic unary operator and its operand.

| x = data + 1;  pointer = &data;  count++; |
| --- |

## **Braces/Scoping**

### Functions

The opening and closing braces are aligned and located in the 1st column.

### **If, Else, For, While**

Any code (even one line) following an “if”, “else”, ”for”, and “while” will be enclosed in braces. Nesting should go no more than 7 levels. Note: This practice helps avoid coding errors if statements are indented incorrectly or commented out (both of which are not allowed). The opening brace is in the same line (following a space) and and closing are aligned is located in the same column as the first character of the controlling reserved word.

| if ( exit == TRUE ) {  // TODO: Do Something  } |
| --- |

## **Names**

### **File Names**

File names for source and header files should be compact, unique, and easily associated with the code contained within them. File names should be in lower case and seperated by underscore (\_).

### **Typedefs for struc**ts and enums

Type names will be in all lowercase and be short but specific. The name of the typedef must end in “\_t” for structure and “\_e” for enums. If more than one word is needed an “\_” will be used to separate them. Enum values should be in uppercase separated by an “\_”.

| typedef struct {  uint8\_t data;  int8\_t number;  uint8\_t\* name;  } screen\_info\_t;  typdef enum { KEY\_BLUE, KEY\_RED } key\_type\_e; |
| --- |

### Macro Names, #defines and constants

Macro names, #defines, global constants and global enums should be in

uppercase with words separated by an “\_”.

### **Function Names**

Function names should be mixed case with the first character lower case (Camel case). When using acronym in function name, only first character of the acronym should be capital. Functions which return boolean types should be prefaced by a verb of some sort such as “Is”, “Are”, or “Can”.

| UpdateDisplayTable(...)  RewriteLcdDisplay(...)  CalculateTimeLeft(...)  IsPrimeComplete(...) |
| --- |

### **Function Parameter Names**

Parameters to functions should be lower case separated with underscore and should start with an underscore character “\_”.

| int32\_t MyFunction(int32\_t \_parameter\_one) {  uint8\_t local\_var = 10;  return \_parameter\_one + local\_var;  } |
| --- |

### **Variable Names**

Local variables should use lowercase and separated by underscore. A global variable name should use lower case prefixed by "g\_". Words used in variable

names should be short but specific. Local variables can be one character in length as long as the intended use is clear. Static variables will use mixed case prefixed by “s\_” to differentiate it from local variables.

## **Coding Practices**

### **Parenthesis**

Parentheses should be used to indicate operator precedence even though precedence is implicit in the operator. Bit expressions should be enclosed in parentheses.

| x = ((a \* b) + (c / d)) / y;  x = (a & BIT\_MASK) + y; |
| --- |

### **Variable Declaration**

Each variable should be declared on a separate line with a comment on the same line defining its purpose.

| int32\_t truck\_count; /\* count trucks arriving at gate A \*/  int32\_t car\_count; /\* count cars arriving at gate B \*/ |
| --- |

### **Constants**

Any special numeric codes and calculation constants should be declared as const and used by name in the code. However, increment and decrement numbers of 1 and countdown limits of 0 are acceptable. Use const instead of #define, whenever possible. A known issue is the declaration of an array in a class declaration. Use of #define is ok here. This allows the value to be seen in the debugger.

| const float ASPECT\_RATIO = 1.653;  const int EC\_BUF\_SIZE = 100; |
| --- |

### **Parameter Passing**

Any formal parameters for functions should be declared as “const” if possible (including reference parameters) to allow additional parameter usage checking by the compiler. Pass parameters by reference or pointer when possible to avoid construction of a new object

### **Parameter Checking**

For every module interface, input parameters should be checked for validity prior to use. Examples of a validity check are checking for NULL pointers and range checking.

### **Return Codes**

All module interfaces should make use of module returns codes. These return codes should be defined in the module header file. The caller should always check the return code to verify proper module operation and handle module errors appropriately. Special care should be taken to ensure that return code values are not duplicated across modules.

### **Enumerated and Predefined Types**

Enumerated types should be used instead of (#define int) to define a set of related values. This allows the compiler to check for illegal values of an enumerated type. The use of project wide integer types which define actual storage size (e.g., int8\_t & uint16\_t from stdint.h) should be used.

### **Macros**

Parentheses should be placed around macro definitions and around arguments within the definition to prevent precedence rules from changing the meaning of the expression. Inline functions are preferred over macros.

### **Branching Statements**

The **goto** and **continue** statements should not be used. The **break**

statement should only be used within a case statement. Case statements

should have a **default** condition unless the switch item is an enumeration.

### **NULL**

Because the definition of NULL is compiler dependent, it should be used

only with pointer types; use zero (0) instead.