VVDN Code Commenting Style (INDx_TRNG_01)

Rev: A0-01

20 June 2013

VVDN Code Commenting Style

'C' program commenting style for software projects

VVDN Contact:

manas.mohanta@vvdntech.com

Revision History:

Date	Rev No.	Description	By
20-06-2013	A0-01	First Draft	Manas

"Difference that makes a professional"

- ***Professional can replicate performance***
- ***Professionals have a plan, and follow the plan***
- ***Professional do what they do better than almost everyone else. "If it is worth doing, it is worth doing well"***
- ***Professional know more about what they are doing than almost everyone else "It is the knowing & doing that counts"***
- ***Professionals will always try to improve by exerting continuous effort to learn more /practice more and get better at what they do***
- ***Professional don't let their feelings interfere with their performance. "The way they feel about doing something is separate from the importance of doing it or the actual doing of it"***

Table of Contents

1 Introduction

This document describes necessary guidelines for commenting in codes. This should be followed by all projects executed at VVDN.

1.1 Need of Comments

Programs are meant to be beautiful as well as understandable. Unless someone tells you something properly, you'd probably do best not to listen to the rest of his advice.

A good program is not only beautiful by the algorithm used, the design, the flow of control - but also in its readability. Often we need to go through older codes or some other codes written by someone else who is not present at that instant. Sometimes the algorithm is too complex to be understood rapidly or completely without some explanation, or the code requires some esoteric function call from the C library that has both a cryptic and a misleading name. And if you ever plan to code for a living, you will almost certainly have to go back several years later to modify the one section of code that you didn't feel like commenting -- or someone else will, and will curse your name.

Finally, commenting code can also lead to a better understanding of the program and may help uncover bugs before testing. For both aesthetic and practical reasons, good commenting is an essential and often overlooked programming skill. Comments are mainly for saying briefly what it the code written is for. What is the source file contains and what each function does. This should give a proper idea about the functionality of the code.

If all of we write our code having a same commenting style for all files, then all files will be in same standard and it will be easier to understand each other’s code, so we will feel comfortable while working together on the same project or modifying an old project.

Another advantage is, if we keep all file headings in an common generic format then we can generate help files and documentation files with a common generic script, which will extract comment fields from file and store it in a file.

1.2 Important guidelines for comments

Comments are good, but there is also a danger of over-commenting. NEVER try to explain HOW your code works in a comment: it's much better to write the code so that the working is obvious, and it's a waste of time to explain badly written code. Generally, you want your comments to tell WHAT your code does, not HOW. Try to avoid putting comments inside a function body: if the function is so complex that you need to separately comment parts of it. You can make small comments to note or warn about something particularly clever (or ugly), but try to avoid excess.

According to coding style there are three types of comments inside code.

1. Startup/heading comments
2. Single line comments
3. Multiple line Comments
4. Function comments

1.2.1 Startup comments.

Startup comments are written at the beginning of the file. Every program should start with a comment saying briefly what it is for. The comment should be at the top of the source in a box like structure.

A template for C source file

```
/**
 * @file       :[required]
 * @brief      :[optional]
 * @Author     :[optional]
 * @Copyright  :[required]
*/
```

Example:

```
/**
 * @file       : thread_sync.c
 * @brief      : This file contains cource code for thread synchronization.
 *               between multiple threads.There are 10 threads which execute
 *               one-by-one depending upon certain conditions.
 * @author     : Sethuraman T.(sethuraman@vvdntech.com)
 *             : Praveen (praveen.raja@vvdntech.com)
 *             : Manas Mohanta (manas.mohanta@vvdntech.com)
 * @Copyright : (c) 2012-2013 , VVDN Technologies Pvt. Ltd.
 *               Permission is hereby granted to everyone in VVDN Technologies
 *               to use the Software without restriction, including without
 *               limitation the rights to use, copy, modify, merge, publish,
 *               distribute, distribute with modifications.
 */
```

1.2.2 Single line comments:

For short and efficient description of any function or variable or any part of the program we use single line comments. It should be like following.

```
/* single line comments look like this */
```

```
/*
 * Important single line comments look like multi-line comments.
 */
```

Put two spaces after the end of a sentence in your comments, Use only /* */ for comments please avoid to use "//" for commenting even for single line also.

Very short comments may appear on the same line as the code they describe, and should be tabbed over to separate them from the statements. If more than one short comment appears in a block of code, then they should all be tabbed to the same tab setting.

Example 1:

```
if (column == 1) {
    ch = to_upper(ch);    /* works only for first column */
} else {
    ch = to_lower(ch);    /* default case */
}
```

Example 2:

```
/* Special section indices.  */
#define SHN_UNDEF       0            /* Undefined section */
#define SHN_LORESERVE   0xff00       /* Start of reserved indices */
#define SHN_LOPROC      0xff00       /* Start of processor-specific */
#define SHN_BEFORE      0xff00       /* Order section before all others
                                        (Solaris) */
#define SHN_AFTER       0xff01       /* Order section after all others
                                        (Solaris) */
#define SHN_HIPROC      0xff1f       /* End of processor-specific */
#define SHN_LOOS        0xff20       /* Start of OS-specific */
#define SHN_HIOS        0xff3f       /* End of OS-specific */
#define SHN_ABS         0xfff1       /* Associated symbol is absolute */
#define SHN_COMMON      0xfff2       /* Associated symbol is common */
#define SHN_XINDEX      0xffff       /* Index is in extra table.  */
#define SHN_HIRESERVE   0xffff       /* End of reserved indices */
```

1.2.3 Multiple line Comments

Multiline comments shall look as follows. Put the opening and closing comment sequences on lines by themselves. Use complete sentences with proper English grammar, capitalization, and punctuation. The comment text should be tabbed or spaced over uniformly. The opening slash-star and closing star-slash are alone on a line.

Example 1:

```
/*
 * Step 9: Process Reset
 * If P.type == Reset,
 * Tear down connection
 * S.state := TIMEWAIT
 * Set TIMEWAIT timer
 * Drop packet and return
 */
```

Example 2:

```
/*
 * Some buggy high speed devices have bulk endpoints using
 * maxpacket sizes other than 512.  High speed HCDs may not
 * be able to handle that particular bug, so let's warn...
 */
```

1.2.4 Functions comment

A function simplifies the complexity of the code both logic-wise as well as code readability. We should comment each function before the function to describe what it is and what it does.

Template:

```
 /**
  * @function    : <Required>
  * @param1      : <Optional>
  * @param2      : <Optional>
  * @param<X>    : <Optional>
  * @return      : <Optional>
  * @brief       : <Required>
  * @caller      : <Optional>
  */
```

Example :

```
/**
 * @function    : area_of_rectangle
 * @param1      : length
 * @param2      : breadth
 * @param3      : area
 * @return      : status
 * @brief       : This function takes length and breadth as input parameter.
 *                Then calculates the area and returns it.
 * @caller      : calc_area, calc_space
 */
int area_of_rectangle (IN float length, IN float breadth, OUT float *area)
{
    if(length == 0 || breadth == 0 ) {
        return INVALID_PARAM;
    }
    *area = length * breadth ;
    return SUCCESS;
}
```

1.2.4 Structure comments

In c program we often use structures .structures represent certain data or object type. So to describe the use of the structure we have to comment the structure name, purpose of use and variables inside the structure with effective description.

TEMPLATE:

```
/**
 *   @structure : structure name
 *   @brief     : details about the structure
 *   @members   :
 *     @mem1    : about variable 1
 *     @mem2    : about variable 2
 *     @mem3    : about variable 3
 */
```

Example 1:

```
/**
 *   @structure : student
 *   @brief     : This is the basic student structure for a student.
 *                Every student variable must have following properties.
 *   @members   :
 *     @name    : name of student
 *     @roll_num: roll number of student
 *     @reg_num : registration number of student
 *     @std     : standard of student
 */
struct student
{
    char name[40];
    int roll_num;
    int reg_num;
    int class;
}
```

1.3 What to avoid in comments:

"When the code and the comments disagree, both are probably wrong." ***- Norm Schryer***

If comments and expression don't have same meaning then it is a bug.

Short comments should be what comments, such as "compute mean value", rather than how comments such as "sum of values divided by n".