FAST C PROGRAMMING STYLE GUIDE

INTRODUCTION

C programming style guideline serves as a set of recommendations and rules for writing consistent, readable, and maintainable code in the C programming language. By adhering to a defined style guide, developers ensure that their code is not only functional but also easy to understand by others and by their future selves. These guidelines cover aspects such as naming conventions, indentation, formatting, commenting practices, error handling strategies, and overall code organization. Consistency in coding style not only enhances code readability but also promotes collaboration and reduces the likelihood of introducing errors during development and maintenance.

INDEX

I. Naming Conventions

Variables			
Subject	**Convention**	**Fix**	**Example**
Variable **Global Variable** **Static Variable**	camelCase	-	`int simpleValue;`
Temporary Variable	camelCase	-	`int var, i, j, k;`
Class Variable	PascalCase	-	`int classValue`
Return Variable	camelCase	-	`int8_t result;`
Variable Pointers	camelCase	Prefix “p”	`int* pValue;`
Function Variable	camelCase	-	`void simpleFunction ( int var, ) { ... }`
Class Function Variable	PascalCase	-	`int8_t FAST_SimpleFunction ( int FunctionVar, ) { ... }`

Definitions			
Subject	**Convention**	**Fix**	**Example**
Struct Definitions	camelCase	Suffix “_s” Type suffix “_t”	`struct simpleStruct_s { ... };`
Union Definitions	camelCase	Suffix “_u” Type suffix “_t”	`union simpleUnion_u { ... };`
Enum Definitons	camelCase	Suffix “_e” Type suffix “_e”	`enum simpleEnum_e { ... };`
Typedef Definitions	camelCase	Type suffix “_t”	`typedef struct { ... } simple_t;`
Class Type Definitions	PascalCase	Prefix “CLASS_” Type suffix “_t”	`typedef struct {...} FAST_SimpleStruct_t;`
Macro Definitions	SCREAMING_ SNAKE_CASE	-	`#define SIMPLE_PI  3.1415f`

Functions			
Subject	**Convention**	**Fix**	**Example**
Functions	camelCase	-	`void simpleFunction (void);`
Construct Functions	camelCase	Prefix “new”	`vec_t newVector(void);`
Init Functions	camelCase	Suffix “Init”	`int8_t simpleInit(void);`
Class Functions	PascalCase	Prefix “CLASS_”	`void FAST_SimpleFunction(void)` `FAST_Handle_t FAST_NewHandle();`

File Names			
Subject	**Convention**	**Fix**	**Example**
Repository Name	PascalCase	-	`SimpleRepository`
Folder Name	PascalCase	-	`SimpleFolder`
Header File	snake_case	-	`simple_code.h`
Source File	snake_case	-	`simple_code.c`

II. Comments

Use the star version of comment type while writing comments. Never use double slash. If there is more than one line, put star for each line of comment and allign the stars like a single line.

If there is a one line, variable comments writen next to variable declaration. Otherwise, write comment top of the declaration and variable name with “@”.

Single Line Variable Comment:

```
uint8_t referenceValue; /* Reference, holds the last used array value */
```

Multi Line Variable Comment:

```
/*
 *  @referenceValue :
 *  Reference, holds the last used array value.
 *  When reference value same with array value,
 *  operation ends.
 *
 */
uint8_t referenceValue;
```

Function Comment: All functions should be explained with input and output values. “@brief” is explanation of the function.

```
/*
 *  @brief This API reads the data from
 *  the given register address
 *
 *  @param address : Address of the register
 *  @param pData   : Pointer of the buffer
 *  @param len     : Read length
 *
 *  @return results of bus communication function
 *  @retval 0 -> SUCCESS
 *  @retval 1 -> ERROR
 *
 */
int8_t dataRead(uint8_t address, uint8_t* pData, uint16_t len);
```

File Comment: Informations about the file. This comment should be top of the file.

```
/*
 *  @File: example.c
 *  @Description: This file contains functions
 *  for performing basic arithmetic operations
 *  and utility functions.
 *
 *  @Author: BerkN
 *  @Created: June 19, 2024
 *  @Last Modified: June 19, 2024
 *
 */
```

III. Spacing

Indentation : Use 4 spaces or one tab for per indentation level**.** The tab character is problematic due to its different behavior on different platforms. Some IDEs can emulate tab as 4 spaces. Each block of code within functions, loops, conditionals, and other constructs should be indented.

```
if (condition) {
    int state1;
    int state2;
}
```

Spaces Around Operators: Use a single space before and after operators. No space is needed between unary operators and their operands.

```
int a = b + c;
int a = -b;
int *ptr = &a;
```

Spaces After Keywords: Use a single space after control flow keywords.

```
for (int i = 0; i < n; i++) {

    ...
}
```

Blank Lines: Use blank lines to separate logical sections of your code, but avoid excessive blank lines.

```
void function() {
    int a = 0;

    /* Separate declarations from statements */
    a = a + 1;

    /* Separate different logical sections */
    if (a > 0) {
        /* Do something */
    }

    for (int i = 0; i < 10; i++) {
        /* Loop through */
    }
}
```

Function Definitions and Declarations: Leave a blank line between the end of one function and the start of another.

```
void function1();

void function2();
```

Inside Parentheses: Do not add spaces inside parentheses, brackets, or braces.

```
array[index] = value;
```