BATL Coding Guidelines

Table of Contents:

List of Tables:

BATL Coding Guideline Documentation

Functional Description:

This document outlines the guidelines for documenting firmware within your codebase. Proper code documentation enhances code readability, maintainability and facilitates easier collaboration among developers.

Document Control:

IMPORTANT NOTICE
Confidentiality & Restricted Use

- *Intended Recipient(s): Bajaj Auto Technology Limited – Intelligent Systems Group – EEE Department, Corp. R&D.*
- *Name:*
- *The above-named recipient(s) is/are responsible for the safe keeping of this document and ensuring that the document and information contained in it are used only in accordance with the agreements with Bajaj Auto Ltd., hereinafter referred to as BATL.*
- *This document and the information contained within it are being provided "for information purposes only" and BATL shall not be held liable as to the accuracy or completeness of the information.*
- *All Intellectual Property Rights, including but not limited to Copyright, applying to this document and the information contained therein vest solely and exclusively with BATL. No part of this document may be copied or reproduced, either in part or in full by any means, whether mechanical or electronic, without prior written permission of the authorized signatory of BATL. BATL reserves all rights to deal with violations of this clause in accordance with applicable laws.*
- *Due to the confidential nature of the information in this document, internal dissemination/copying should be made on a "need to know" basis only and this "Document Control" notice should be included with all copies.*
- *This document remains the property of BATL.*

1. Introduction:

This document outlines the guidelines for documenting firmware within the codebase. Proper code documentation enhances code readability, maintainability and facilitates easier collaboration among developers. It covers guidelines for declaring variables, functions and user-defined variables. Additionally, it also proposes naming guidelines for enum members.

2. Targeted Audience:

This document is intended to be shared with BATL Intelligent System group developers.

3. Variable Naming Guidelines:

Variable names should be concise yet descriptive enough to convey the purpose of the variable.

3.1 Naming Pattern:

Follow a consistent pattern for variable names to enhance readability and predictability. The proposed naming patter is as follows:

<Type of variable>< ><Scope indicator><Type indicator><Class indicator><_><Name of variable>

3.2 Components of the Pattern:

3.2.1 Type of variable:

Indicates the data type or purpose of the variable

3.2.2 Scope indicator:

Specifies the scope of the variable (e.g., local, global).

Scope	Scope indicator
Global	g
Static	s
Local	Not Applicable
Static Local	sl

Table 1: Scope indicator

3.2.3 Type indicator:

Indicates the data type (e.g., uint8_t, sint32_t, float).

Type	Type indicator
uint8_t	u8
uint16_t	u16
uint32_t	u32
uint64_t	u64
int8_t	s8

int16_t	s16
int32_t	s32
int64_t	s64
char	c
bool	b
single	f16
double	f32
SampleEnum_E	e
SampleStruct_T	st
SampleUnion_U	u
funcPtr	fpt
void	v

Table 2: Type indicator

3.2.4 Class indicator:

Denotes the classification of the variable (e.g., array, pointer, structure)

Class	Class indicator
Pointer	pt
Array	ar

Table 3: Class indicator

3.2.5 Name of variable:

- The data type qualifier string is followed by an _.
- The variable name should be written in Lower Camel Case.
- The name must always begin with an alphabet.

3.3 Examples:

Variable declaration	Description
static uint8_t su8ar_sampleArray[10];	Private array containing uint8_t type elements
SampleStruct_T gst_sampleStruct;	Public structure
uint16_t *gu16pt_sampleEnum;	Public pointer to uint16_t
SampleEnum_E e_sampleEnum;	Local enum
SampleUnion_U gu_sampleUnion;	Public union
SampleFuncPtr_F *gfpt_sampleCallBack	Global function pointer, note return type of function and parameters to function need not to be specified in naming
static UInt16_T slu16_sampleVar = 0;	Static variable local to a function.

Table 4: Variable naming examples

4. Function Naming Guidelines:

All the functions should be declared with below mentioned guidelines:

4.1 Naming pattern:

Follow a consistent pattern for function names to enhance readability and predictability. The proposed naming patter is as follows:

<Scope indicator><Return type indicator><_><Location indicator><_><Name of function>

4.2 Components of the pattern:

4.2.1 Scope indicator:

Specifies the scope of the variable (e.g., local, global).

Scope	Scope indicator
Global	g
Static	s
Local	Not Applicable

Table 5: Scope indicator

4.2.2 Return type indicator:

Same as variable, the return value of a function can be mentioned at the beginning of function name.

4.2.3 Location indicator:

- Module name where function is defined should also be preferably included in the function name.
- So, for example, it will be easy to identify LIB, LIL, MCAL which hosts the function. As per the location of the function it can be FSM, IPC, BNET, SEC, etc.
- The location naming is only applicable for global functions and private/local functions may not include this field.

4.2.4 Name of function:

- The name of the function should begin in Pascal Case (each word starts with a capital letter), please refer to example below.
- Also please do note that the **return type indicator field** and **location indicator field** should be followed by an underscore '_'.

4.3 Examples:

Function declaration	Description
uint8_t su8_SampleFun1();	Private function returning uint8_t type element
SampleEnum_E e_SampleFun2();	Local function returning SampleEnum_E
void gv_FSM_SampleFun3();	Public function returning void, residing in FSM module

Table 6: Function naming examples

5. User-Defined Data Type Naming Guidelines:

The following guidelines establish a standard approach for naming user-defined data types in your codebase. Consistency in naming user-defined data types enhances code readability, maintainability, and collaboration among developers.

5.1 General rules:

5.2 Naming pattern:

User-defined data type names should be descriptive enough to convey the purpose. Follow a consistent pattern for user-defined data type names to enhance readability and predictability. The proposed naming patter is as follows:

<Type><Name><Suffix>

5.3 Components of the pattern:

5.3.1 Type:

Specifies the kind of data type being defined (e.g., struct, enum, union, typedef).

5.3.2 Name:

- Written in Pascal Case (each word starts with a capital letter)
- Should clearly describe the purpose or structure of the data type.

5.3.3 Suffix:

Data type	Suffix
Structure	_T
Union	_U
Enum	_E
Function pointer	_F

Table 7: Suffix for user-defined data types

5.4 Examples:

Data type declaration	Description
typedef struct {…} SampleStruct_T;	Typedef for structure named SampleStruct
typedef struct {…} SampleEnum_E;	Typedef for enum named SampleEnum
typedef struct {…} SampleUnion_U;	Typedef for union named SampleUnion
typedef void (*SampleCallBack_F)(void);	Typedef for a function pointer named SampleCallBack

Table 8: User-defined data type naming examples

6. Enum Member Naming Guidelines:

The following guidelines establish a standard approach for naming enum members. It will also help developers to distinguish between enums and defines as they both usually are written in block letters. Consistency in naming user-defined data types enhances code readability, maintainability, and collaboration among developers.

6.1 Naming pattern:

Enum member names should be descriptive enough to convey the purpose. Follow a consistent pattern for to enhance readability and predictability. The proposed naming patter is as follows:

<e><Acronym of enum data type ><_><Name of member>

- Name should be in block letters.
- Use '_' to separate words as there are all the block letters.
- To distinguish similar enum members from other enum data type, mention acronym of enum data type.

6.2 Examples:

```
typedef enum
{
    eSE_MEMBER_1,
    eSE_MEMBER_2,
} SampleEnum_E;
```

7. Change log:

Version	Data of change	Modified by	Summary	Section modified
1.0	01/08/2024	SHIVAM CHUDASAMA	First draft.	All
1.1	27/03/2025	YASH GIRAMKAR	- Changes incorporated to include guidelines for creation of static local variables. - Example of the same is also included in corresponding section.	-3.2.2 -3.3