Course Coding Standards

Existing Code Structure

Remember that consistency is more important than standards. That means that you can make different decisions, but it is important to stay consistent. With that side, for better or worse, C code uses **under_score_standard** while C++ code **usesCamelCase**. This is an inconsistency that needs to be fixed because the developers realized that acronyms do not work well with camel case and this was realized too late.

Secondly, FreeRTOS follows significantly different coding standard. Please read more about it here, but the biggest hint is that the first letter, such as "v" in **vTaskDelay** means that it is a void return type.

Goals

- Easily reusable from common multi purpose functions and objects.
- Easily maintainable from consistent coding practices.
- Easily understood code from well described methods and self-documenting variables.

Code Structure

- DO NOT include unused headers in a source file
- ALWAYS use parenthesis arguments in any mathematical expression. Do not rely on operator precedence.
- ALWAYS Prioritize code maintainability over code complexity
- DO NOT use break statements except for in switch statements
- ALWAYS put curly braces must be on their own line

```
int32_t algorithm(int32_t a, int32_t b)
{
    if(a < 0)
    {
        a = -a;
    }
    return (a + b);}</pre>
```

- NEVER omit curly braces
 - **NEVER** single line an if statement

```
int32_t algorithm(int32_t a, int32_t b)
{
    if(a < 0) a = -a; /* NOT ALLOWED */
    if(a < 0)
        a = -a; /* NOR IS THIS ALLOWED */
    if(a < 0) { a = -a; } /* REFRAIN FROM THIS AS WELL, MAKES DEBUGGING CODE LINES AMBIGUOUS */
    return (a + b);}</pre>
```

• REFRAIN from functions that are more than 100 lines. Make code highly modular

• DO NOT have more than one exit (return statement) point from a function

```
/* BAD */
int32_t algorithm(int32_t a, int32_t b)
{
□if(a < 25)</pre>
[]
□return 25;
□}
□else
□{

□return b;

_}
}
/* GOOD */
int32_t algorithm(int32_t a, int32_t b)
{
□int result;
∐if(a < 25)
[]
\squareresult = 25;
_}
□else
[]
\squareresult = b;
□}
[return result}
```

- ALWAYS Use spaces around all operators.
 - o DO: uint32_t counter = 5 + 2;
 - o DO NOT: uint32_t counter=5+2;
- ALWAYS include a default case in a switch statements
- ALWAYS Use 4 spaces for indentations
 Do not use tab characters in your source
- ALWAYS Separate logical blocks with one blank line

```
//Bad Example
uint8_t current[6];
//Good Example
```

Memory

• REFRAIN from dynamically allocating memory

Variables (snake_case)

- **REFRAIN** from creating global variables.
- ALWAYS put constants, enumerations, structures, objects/variables at the top of the scope
 - For global variables that means the top of the file
 - $\circ\,$ For local variables to a function, the top of a function
- ALWAYS use highly descriptive variable names
- ALWAYS use snake_case for variable names
 - ° Example: uint32_t interrupt_counter = 0;

Types (snake_case)

- ALWAYS use <stdint.h> integer types from: uint8_t, int8_t, uint16_t, int16_t, uint32_t, int32_t, etc
- ALWAYS use booleans from stdbool.h.
- NEVER use floats for calculations, use doubles instead. You may store floating point values as floats or doubles.

Macros (CAPS)

- Use of macros should be limited to preprocessor conditionals and that is it
 - Any flags should be checked for as the following: #if (MY_FEATURE)

Inline Functions (CAPS)

• DO use this in place of Macro functions.

```
/* !!BAD!! */
#define HASH(letter) (uint8_t)(letter-'A')
/* GOOD */
inline uint8_t HASH(char letter)
{
```

Constants (CAPS)

- ALWAYS use const or enumerations instead of "magic" numbers
- Use unit names in constants.
 - $\circ\,$ Use constants to convert between units instead of hard coding the conversions.

const uint32_t MILLISECONDS = 1;const uint32_t SECONDS = (1000 * MILLISECONDS);

Functions (camelCase)

- ALWAYS use CamelCase for function names
- Attempt to make functions generic so it can be reused.

Classes/Structures/Objects (CapCamelCase)

- Protected Section
- Member variables as private
- Private Section
 - Not used
- Public Section
 - \circ Methods
 - Setters and getter methods
- · Declare constants within classes as static
- Constructors should not alter or modify hardware/registers.

Revision #10 Created 1 year ago by Admin Updated 11 months ago by Preet Kang