

Conventions for C Code

Naming Conventions:

1.

Symbol Type	Convention
#define constants	All caps with underscores to separate words.
#define macros	All caps with underscores to separate words followed by ().
Stored constants, and global variables	Mixed case. Multiple words are combined into one word with the first letter of each word capitalized.
Function names	Mixed case. Multiple words are combined into one word with the first letter of each word capitalized.
Local variables, parameter names	All lower case with or without underscores to separate words.
Typedef names	All caps.

2. Data object names should be nouns while routines should describe an action.

3. MCU I/O and control register names must be the same as the name used in Motorola documentation.

4. Names from prewritten modules must be consistent with the module – even if it violates normal naming conventions.

Standard type definitions:

```
typedef unsigned char    INT8U;  
typedef signed char     INT8S;  
typedef unsigned short  INT16U;  
typedef signed short    INT16S;  
typedef unsigned long   INT32U;  
typedef signed long     INT32S;
```

```
#define ISR void __interrupt
```

Signed and unsigned modifiers must be explicitly defined – do not use defaults.

Typedefs must be used to define struct, enum, and union types.

Type Casting:

Explicit type casting is required for mismatches.

Control constructs:

1. Indent 3 to 6 characters.
2. Use comma operator to make assignments separate from expression in conditional expressions.
3. Braces are always used for control structures. Even if the code block consists of a single statement.
4. There must be no overlapping conditions on if-else if-else constructs.

Functions:

1. Entry function is always called main().
2. Function names should start with module name or abbreviation.
3. Task functions should always include the word *task* or *tsk* in the name.
4. Abstract parameter names must be used for documentation.

Header Files:

1. Master header file organization described in Chapters 12 and 14.
 2. Header files should contain items that are used across different modules.
 3. No declarations that actually allocate memory are allowed in a header file. These should 'belong' to a specific module. The header file will then contains an *extern* declaration for public resources.
-

Comments:

1. Comments should make the code easier to read and understand.
2. Comments should be indented with the code so program blocks remain visual.
3. Comments should not be a redundant description of a statement or instruction.
4. Comment headers are required at the start of a module, for each routine or function.