Style Guide for C Programming

QUTMS Embedded

September 28, 2020

Alex Lossberg
Software Manager

Calvin Johnson
Embedded Lead

General Team Practices

Supported IDEs - Integrated Development Environment

STM32CubeIDE is the only supported IDE by the team for embedded development. You may wish to use your own IDE if you are familiar with another environment such as Visual Studio, however all practices must still be followed and support may not be provided when your environment is causing an error.

STM32CubeIDE is build on the Eclipse framework that you may be familiar with from programming units such as CAB302. It is available for free download at the link provided here.

Code Initialisation

When generating new codebases, STM32CubeMX may be utilised to automate and assist in the configuration of peripherals and pins.

> STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code for the Arm® Cortex®-M core or a partial Linux® Device Tree for Arm® Cortex®-A core), through a step-by-step process.

The first step to create a new code base is to contact either your Lead or Software Manager. All Leads and Managers have the ability to create a new repository on GitHub. It is recommended however that you perform this task in conjunction with your respective Lead at the time, to ensure progress can be tracked on the repository.

Once the repository is live, you may now continue on to use STM32CubeMX. If you have not used CubeMX before, it is recommended to continuing consulting your Embedded Lead at this stage, to ensure you configure the pin layout and clock correctly.

Once the code base has been generated, it is critical that you immediately commit the generated code to GitHub. Do not under any circumstances begin programming the functionality desired for the board. Commit what has been generated, and then proceed. This is to ensure all changes can be track commit-to-commit, without having to regenerate the code base, and to track all changes from the initial development phase of the project.

Line Width

Historically, the maximum length of a line of code is 80 characters long in C (Due to monitor resolutions). Though this is no longer of concern, this is still a hard deadline we will be adhering to. Due to the extensive debugging utilities and panes in CubeIDE, and the often necessary requirement of laptop development in embedded work (Often with low resolution displays), the 80 character width is still enforced. It also simplifies the processes of line-by-line Git reviews, and when you find yourself in a situation like this in the future, you will greatly appreciate this limit. If you find yourself going over 80 characters, it may be time to rethink either your variable names or your overall design!

Example - What to Avoid

```
1 // These next lines of code first prompt the user to give two integer values and th
2 int first_collected_integer_value_from_user = get_int("Integer please: ");
3 int second_collected_integer_value_from_user = get_int("Another integer please: ");
4 int product_of_the_two_integer_values_from_user = first_collected_integer_value_from_
5 }
```

settings.json

If you are using Visual Studio Code, we suggest you amend your settings.json with the following rule, to see a visible restriction.

```
{
    "editor.rulers": [80]
}
```

Indentation & Braces

GNU Style

We follow the GNU Style of curly braces. This is the default setting within the CubeIDE. This style is primarily characterised by the curly braces appearing on new lines following a conditional.

Example

```
for (i = 0; i < num_elements; i++)
{
    foo[i] = foo[i] + 42;

    if (foo[i] < 35)
    {
        printf("Foo!");
        foo[i]--;
    }
    else
    {
        printf("Bar!");
        foo[i]++;
    }
}

int main(void)
{
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();
  /* Infinite loop */
  while (1)
  {
    bar()
  }
}
```

Iterators

Whenever you need temporary variables for iteration, use i, then j, then k, unless more specific names would make your code more readable.

Comments

Within functions, use in-line comments, restricted to one line, else it becomes difficult to distinguish comments from code, even with syntax highlighting. Place the comment above the line(s) to which it applies. No need to write in full sentences, but do capitalize the comment's first word (unless it's the name of a function, variable, or the like), and do leave one space between the // and your comment's first character.

Example

```
// Convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

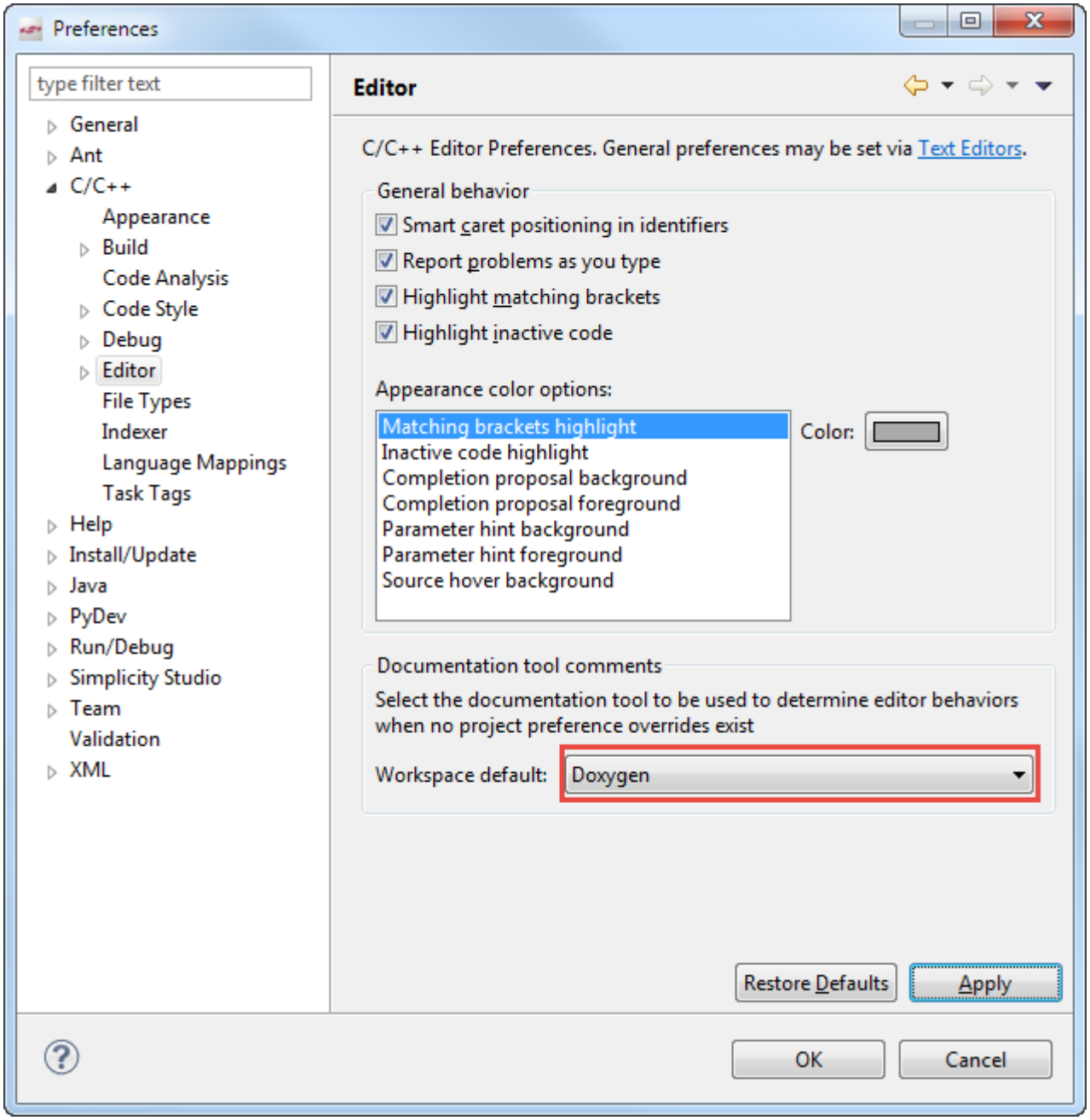
Example - What to Avoid

```
//Convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);

// convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);

float c = 5.0 / 9.0 * (f - 32.0); // Convert Fahrenheit to Celsius
```

Documentation

QUT Motorsport uses Doxygen to generate all our documentation. Doxygen is a utility that parses code & stylised comments to produce written documentation in an external document.

To ensure your codebase is compatible with Doxygen, we strongly recommend you modify your CubeIDE settings to be Doxygen compatible. To integrate Doxygen, launch your workspace or an empty project in CubeIDE. Then select Windows -> Preferences from the Menu bar. Then head to C/C++ -> Editor -> Documentation tool comments -> Workspace default, and change the dropdown to Doxygen. This action is visible in the figure below.



Every projected generate within the CubeIDE environment will now include comments in a Doxygen compatible format. The Doxygen method of commenting requires a developer to append two asterisks (*) to their comment describing the functionality of the method/attribute/variable below.

Example

```
/**
 * Initialise iterator
 */
int i = 0
```

This is commonly referred to as the "Javadoc" style. While Doxygen also supports Qt styling and other operators, please avoid using other styles for the sake of maintainability. The example above may also be shortened to one line, provided the asterisks are placed in an identical format

Example

```
/** Initialise iterator */
int i = 0
```

Once your code is made compatible with Doxygen, you may now proceed to generate the documentation. Doxygen may be downloaded from the link provided here. The extensive Doxygen documentation is also provided here. Doxygen comes packaged with a GUI Utility, however we strongly recommend using the CLI. For this reason, we suggest noting your installed directory, and adding the bin file to your PATH environment variables, to access the utility at anytime.

Once installed, the next step is to configure Doxygen for your repository. Clone the QUTMS_Doxygen repository available here to access the configuration template. Drag the Doxyfile and logo.png into the root folder of your current working directory. Provided Doxygen has been added to your path, or you have access to the file path, you may now execute Doxygen via the CLI.

Before executing the command however, we must modify the existing Doxyfile for your current repository. Open the Doxyfile in your text editor of choice, There are only 3 parameters that require modification.

```
# The PROJECT_NAME tag is a single word (or a sequence of words surrounded by
# double-quotes, unless you are using Doxywizard) that should identify the
# project for which the documentation is generated. This name is used in the
# title of most generated pages and in a few other places.
# The default value is: My Project.

PROJECT_NAME           = QUTMS_TITLE

# The PROJECT_NUMBER tag can be used to enter a project or revision number. This
# could be handy for archiving the generated documentation or if some version
# control system is used.

PROJECT_NUMBER         = Version 0.1

# Using the PROJECT_BRIEF tag one can provide an optional one line description
# for a project that appears at the top of each page and should give viewer a
# quick idea about the purpose of the project. Keep the description short.

PROJECT_BRIEF          = QUTMS_DESC
```

Provided the PROJECT_NAME, PROJECT_NUMBER & PROJECT_BRIEF are modified, you are now ready to generate your documentation.

```
doxygen Doxyfile
```

Source Control

QUT Motorsport currently manages source control through a publicly accessible GitHub organisation. Whilst the repositories are free to access online, the required privileges to collaborate on these repositories is managed by the team. If you require access to a particular repository, or are yet to be added to the organisation, please contact your respective Lead. If they are not available, the Software Manager also has the necessary permissions to grant you access.

We assume prior knowledge of Git from your degree & potentially other side projects. There are minimum expectations involved with using our GitHub. The "golden rules" and expectations are outlined below.

- Commit often
- Always commit to a new branch
- Never mix active issues within an active branch
- Never merge ones own pull request
- Be descriptive

These are the bare minimum guidelines we expect everyone to follow. Yes there are exceptions to these rules, and yes you may think your case requires it, but before doing so, contact your respective team Lead.

Maintainability & Readability

Pointers

When declaring a pointer, write the asterick next to the type name

```
int* p;
```

Type Safety

Avoid using definitions, and opt for constants when applicable.

```
/** Avoid using */
#define A 12

/** Opt for this instead */
const int a = 12;
```