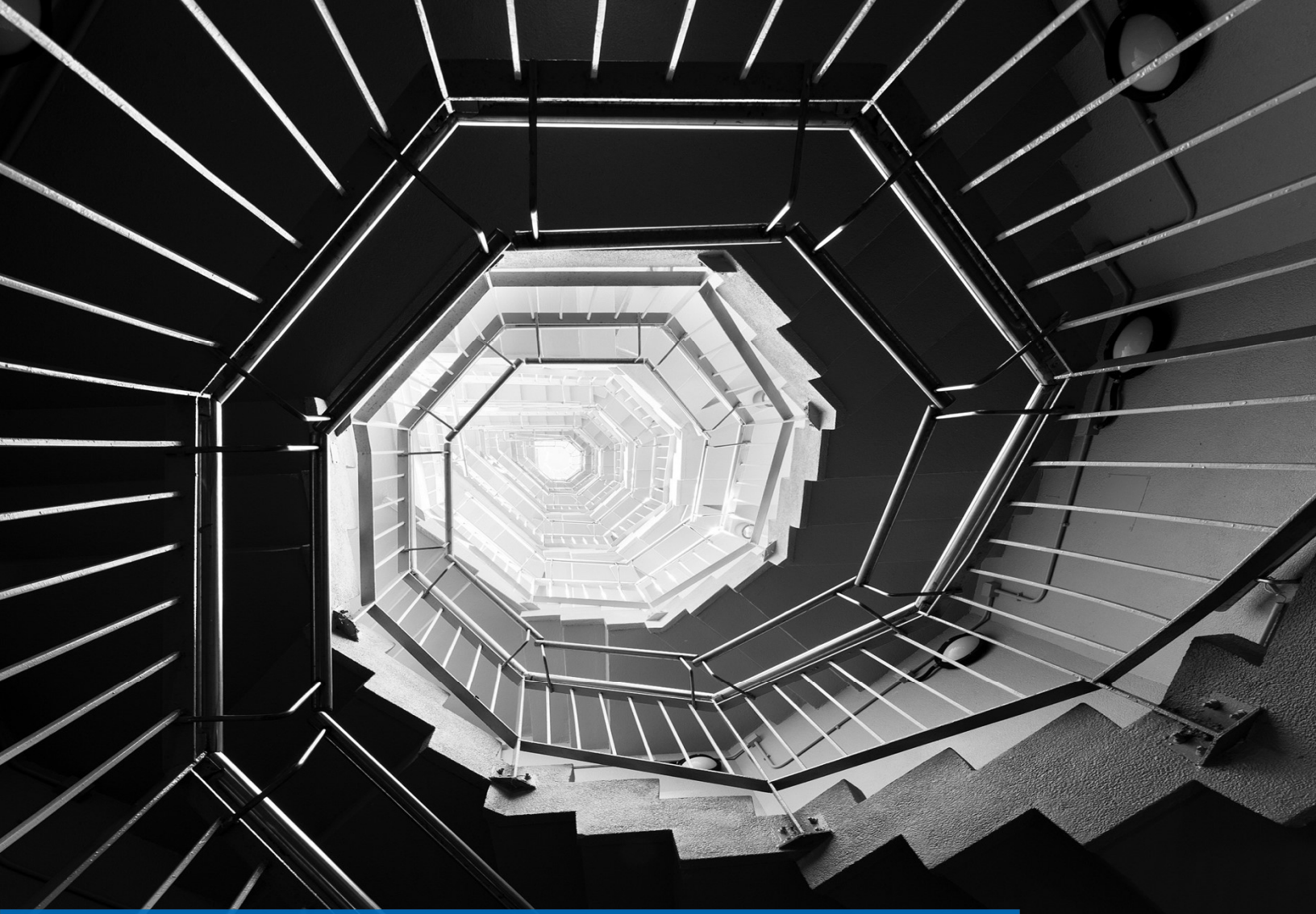


Titan Virtual Corp.

SDK Code Formatting

Contents

About this document

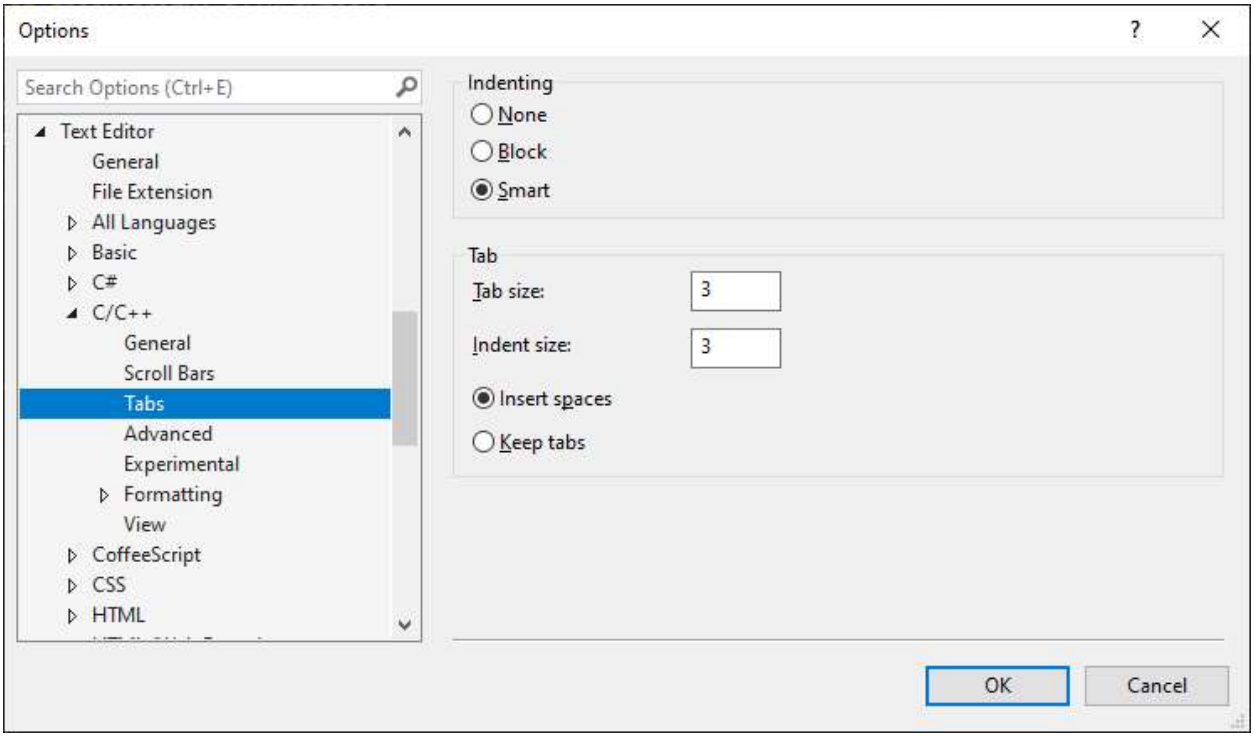
This document describes the formatting of all code developed by Titan Virtual Corp. This document does not address good coding practices nor does it encourage the developer to follow the rules herein. Use this document as a navigation tool to familiarize yourself with our code formatting style. You are encouraged to adopt this formatting should it be useful in your own projects or if you plan on merging your own code with the Titan Virtual Corp. source code repository.

Formatting:

Indentation:

Tabs:

Titan code uses three spaces instead of tabs for indentation. By default, Visual Studio uses tabs and the indentation size is 4. The following settings are used throughout Titan code.



Code blocks:

Methods are separated by exactly one empty line, and the function body is indented using *Allman* style:

```
void MyFunction()
{
   statement;
   statement;
}
```

Code blocks are indented using *Whitesmiths* style:

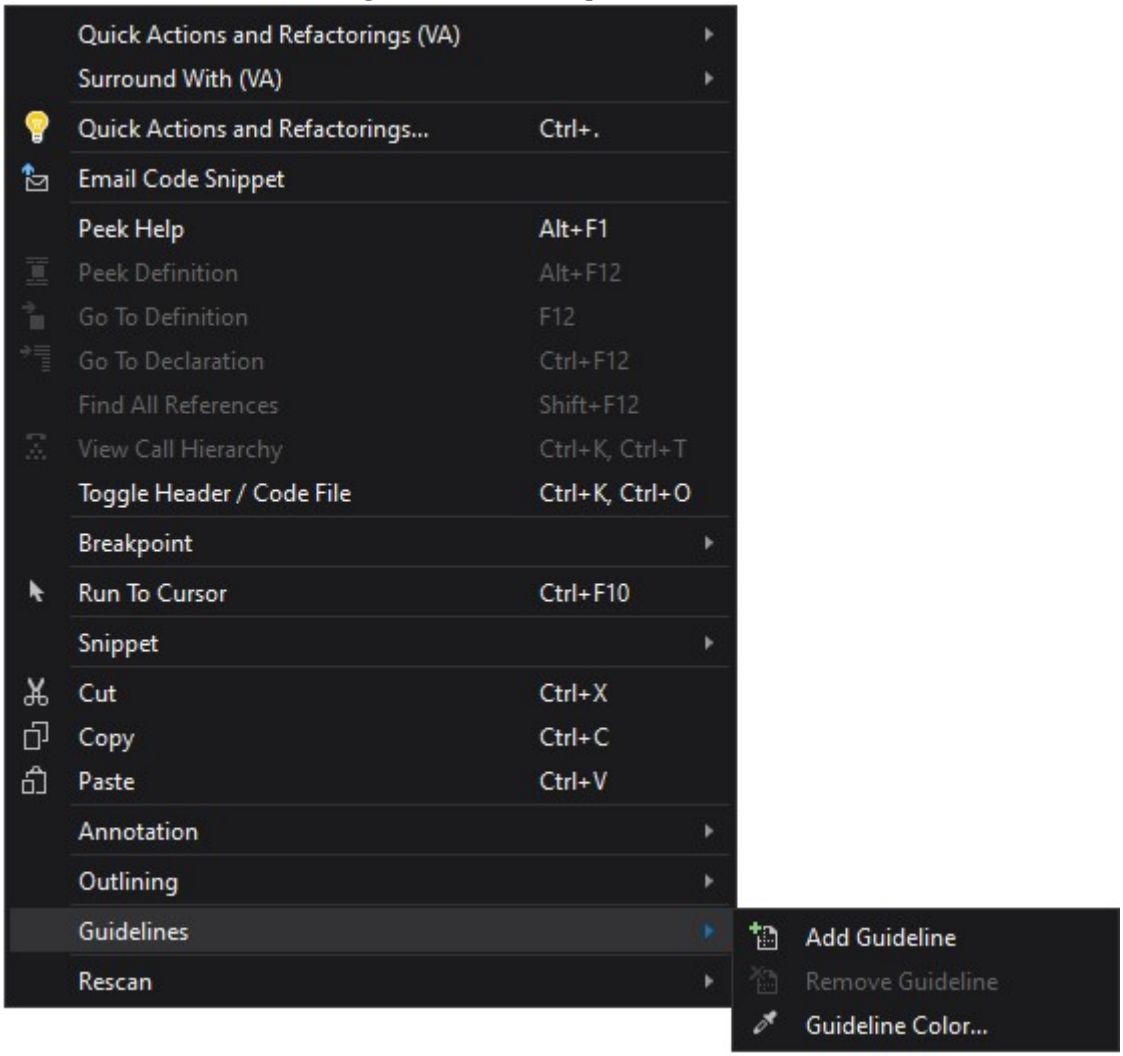
```
if (x)
   {
   statement;
   }
```

```
switch (x)
   {
   case CASE1:
      break;
   case CASE2:
      break;
   };
```

Variables:

Variables are declared on the 55th column, and their in-line descriptive comments begin on 94th column.

Visual Studio offers visual guidelines to help with indentation:



Add a guideline on the 55th column and another on the 94th column if you want to use these features.

The body of all header files is indented by 3 spaces:

```
#ifndef HEADER_H
#define HEADER_H

   #include <header1.h>
   #include <header2.h>

   declaration;
   declaration;

#endif
```

Code Spacing:

All local variables are declared at the top of the method block. Declarations in the middle of a function block are discouraged.

All commas and semicolons are followed by a white space.

```
variable1, variable2, variable3;
```

All operators are spaced by one character on both ends. An exception to this rule applies to the NOT operator (~ or !) and negation (-) operator:

```
variable = 2 + 3;
variable = (2 + 3) * 12;
variable = ~variable;
variable = -2;
```

Pointers and references are attached to the variable, and not the variable type:

```
TITAN_UINT                          *variable;
TITAN_UINT                          &variable;
```

Condition blocks and loops are spaced from their parameters by one character:

```
if (x)
while (x)
for (x; x < y; x++)
switch (x)
```

Function names are not spaced from their parameters:

```
MyFunction(x, y, z);
```

Line spacing:

All code and comments are separated by one line at the most. Statements that reflect a subsection of code are grouped together with no line separating theme:

```
statement_A1;
statement_A2;
statement_A3;

statement_B1;

statement_C1;
statement_C2;
```

End of file:

All files end with the following comment:

```
/*
   End of file
*/
```

Naming convention:

Variable names:

Variables are named using lower case characters and separated by underscore. Variable names are descriptive. It is discouraged to use numbers in variable names unless the numbers represent a meaningful implementation-independent description.

```
TITAN_UINT                          number_of_elements;
TITAN_BITMAP64                      pre_processing_flags;
```

Function names:

Functions names typically have a verb associated with them to describe the action performed by those functions. Function name words are in *PascalCase* (upper camelCase) style, concatenated together where the first character of each word is upper case, and the remaining characters are lower case:

```
TITAN_BOOL OpenFile(…);
TITAN_ULONG GetFileSize(…);
```

Class names:

Class names follow the same notation rule as function names, with the exception that they don't have to contain a verb in their description. Framework methods (constructor, destruction, and initializer) are at the very top of the class declaration body, followed by public, protected, and private methods respectively. The lower part of the class declaration body contains the public, protected, and private variables respectively:

```
// MyObject class declaration
class MyObject
   {
   public:
      // Framework methods:
      MyObject();
      ~MyObject();
      Initialize(…);

   public:
      // Public methods:
      MyPublicMethod1(…);
      MyPublicMethod2(…);

   protected:
      // Protected methods:
      MyProtectedMethod1(…);
      MyPublicMethod2(…);
```

```
   private:
      // Private methods:
      MyPrivateMethod1(…);
      MyPrivateMethod2 (…);

   public:
      // Public variables:
      TITAN_UINT                          m_variable_name;
      TITAN_UINT                          m_variable_name;
      TITAN_UINT                          m_variable_name;
   protected:
      // Protected variables:
      TITAN_UINT                          m_variable_name;
      TITAN_UINT                          m_variable_name;
   private:
      // Private variables:
      TITAN_UINT                          m_variable_name;
      TITAN_UINT                          m_variable_name;
   };
```

Structure and enumerator names:

Both structures and enumerator definitions are suffixed with *_t* for type:

```
struct my_struct_t
   {
   TITAN_UINT                          variable_name;
   TITAN_UINT                          variable_name;
   };

enum my_enum_t
   {
   ENUM_VALUE_NAME,
   ENUM_VALUE_NAME,
   };
```

Numbers:

Floating point values have the f suffix:

```
3.141591f
```

64-bit integers have the L suffix:

```
142927049237L
```

Files:

Code file names follow the same rule as class names:

```
MyFile.cpp
MyFile.h
```

Classes that have very large implementation files are broken into a bundle of smaller (< 200 lines) files with class prefix as a file name and separated by an underscore:

```
MyObject.h
MyObject_Instatiation.h
MyObject_Saving.h
MyObject_Loading.h
MyObject_Process.h
MyObject_ErrorHandling.h
```

Commenting:

Except for the file hear and function/class implementation header, document blocks `/* */` are avoided. Comments inside of function/code blocks use `//` notation.

The general rule about commenting inside our code is the following:

- All classes are described (one line or more).
- All structures are described (one line or more).
- All structure variables are described (one line or more).
- All class variables are described (one line or more).
- All method implementations are described (multi-line).
    - Description are high-level conveying the purpose of the method .
    - Method return conditions are described.
- All definitions #define are described (one line or more).
- All conditions are described (if, for, while, switch, etc.).
- All method calls are described.
- All assignment to non-local variables are described.
- The maximum allowed number of undocumented consecutive code statements is 3.
- Variables inside a method do not have to be described.
- Class function declarations do not have to be described.