Coding Standards for C++

Abstract

Coding standards are intended to increase the efficiency and enjoyment of **team-based** C++ development. These coding standards reflect the significant value of uniform coding styles and practices.

1 Naming conventions

The names of all methods/functions, classes, data, variables, etc. follow the ***hump convention***. Hence, all names (other than macros) consist of only alphanumeric characters (**no underscores!**). Names should be as ***descriptive*** as possible, with only minor regard for the length of the name. All characters are lowercase except at word breaks, where uppercase letters are used. The first letter in a name may be upper or lower case depending on its type. For example:

- **Functions and class methods** begin with an uppercase letter, e.g., `GetCenterOfMass()`
- **Local data** (in methods) begin with a lowercase letter (and do *not* begin with **my**), e.g., `otherBody`.
- **Pre-processor macros** do not follow the hump convention and may use underscores. Conventionally, macros use all upper-case letters and underscores, e.g.,
  ```
  #define  __USINGSOCKETS__     0
  ```

2 Comments

Names of classes, functions and data should be as descriptive as possible. More verbose information is contained in comments which must be maintained as code evolves. The preferred style of comments is:[1]

1. Name functions and data as descriptively as possible to minimize comments.
   It is difficult to keep code updated - and even more difficult to keep comments updated.
2. Brief comments describing functions (or methods) appear before prototypes.
   This comment style is also preferred for documenting lines/blocks of code inside functions
   ```
   // Converts from one set of units to another and returns the conversion factor
   double CalculateUnitConversionFactor( const char *unitsA, const char *unitsB );
   ```
3. Short comments may appear at the end of the same line as the instruction they describe, e.g.,
   ```
   int numberOfDogs = 0; // Counts the number of dogs in the kennel
   ```
4. Longer comments describing complicated methods appear immediately proceeding their respective method in `.cpp` files, with each long comment beginning and ending with a dashed line, e.g.,
   ```
   //---------------------------------------------------------------------------
   // This method uses NIST standards for calculating the conversion factor between units.
   // The number 0.0 is returned if the conversion is nonsensical, e.g., converting kg to meter.
   // Otherwise, the conversion factor from  unitsA  to  unitsB  is returned.
   // For example, if unitsA is "inch" and unitsB is "cm", the number 2.54 is returned.
   // This method returns conversion factors involving time, length, mass, angle, charge,
   // velocity, angular velocity, frequency, acceleration, angular acceleration, force,
   // pressure/stress, work, energy, power, area, volume, current, and dimensionless units with
   // prefixes of femto, pico, nano, micro, milli, m, centi, c, kilo, k, mega, giga, tera, peta.
   //---------------------------------------------------------------------------
   double CalculateUnitConversionFactor( const char *unitsA, const char *unitsB )
   ```

[0] Last updated April 13, 2007 by Paul Mitiguy.

[1] Avoid using comments of the form `/* hello */` instead use `// hello`.

3 Horizontal and vertical whitespace

Code should be properly formatted for viewing by text editors and for printing. Ideally, each line is less than **80** characters wide. **Use spaces** for horizontal whitespace, **not tabs**. There are no horizontal spaces between the end of a method and its left parenthesis. Hence `SomeMethod()` is preferable to `SomeMethod ()`. This is also true with `if()`, `for()`, `while()`, etc. Statements inside of braces are indented **three spaces**. A matching pair of braces should be aligned in the **same column**. The first set of braces is in column 0 and align with the definition of a method. Subsequent sets of braces align with the first letter of the statement that created them (e.g., braces are aligned with the first letter in `if`, `for`, `while`, `do`, and `switch` statements). For example,

```
//---------------------------------------------------------------------------
const char*  ConvertStringToDouble( const char *s, double &returnValue, double defaultValue )
{
   // Default return value (in case the string is not a valid number)
   returnValue = defaultValue;

   // Check if  s  is a NULL string or  "abc"  or  "123huh"  or  "(&junk#"  or
   if( s != NULL )
   {
      // Use the standard math function strtod to parse the number
      char *pointerToCharacterAfterNumber = NULL;
      double x = strtod( s, &pointerToCharacterAfterNumber );

      // Ensure the number was not too large (overflow), such as 1.0E+999 or -1.0E-999
      if( errno==ERANGE  &&  x!=0.0 ) return NULL;

      // Ensure the character after the number is a space or '\0', not 'a' or 'z' or ...
      char characterAfterNumber = pointerToCharacterAfterNumber ? *pointerToCharacterAfterNumber : 'z';
      if( characterAfterNumber == '\0' || isspace( characterAfterNumber ) )
      {
         returnValue = x;
         return pointerToCharacterAfterNumber;
      }
   }
   return NULL;
}
```

Single-statement `if`, `for`, `while`, and `do` logic may appear without braces, e.g.,

```
if( myBody == myWorldBody ) position = Vect3(0,0,0);
```

Use a single blank line to separate logical blocks of code.
Use two blank lines proceeding and following functions or methods.
Proceeding each method should be the following 80 character-wide line:

```
//---------------------------------------------------------------------------
```