OpenFOAM C++ style guide

Feb 2011

Contents

1 OpenFOAM C++ style guide

1.1 General

- 80 character lines max

- The normal indentation is 4 spaces per logical level.
- Use spaces for indentation, not tab characters.
- Avoid trailing whitespace.
- The body of control statements (eg, `if`, `else`, `while`, etc). is always delineated with brace brackets. A possible exception can be made in conjunction with `break` or `continue` as part of a control structure.
- The body of `case` statements is usually delineated with brace brackets.
- A fall-through `case` should be commented as such.
- stream output
  - « is always four characters after the start of the stream, so that the « symbols align, i.e.

    ```
    Info<< ...
    os  << ...
    ```

    so

    ```
    WarningIn("className::functionName()")
        << "Warning message"
    ```

    NOT

    ```
    WarningIn("className::functionName()")
    << "Warning message"
    ```

- no unnecessary class section headers, i.e. remove

```
// * * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * //

    // Check

    // Edit

    // Write
```

if they contain nothing, even if planned for 'future use'

- class titles are centred

```
/*---------------------------------------------------------------------------*\
                        Class exampleClass Declaration
\*---------------------------------------------------------------------------*/
```

NOT

```
/*---------------------------------------------------------------------------*\
                Class exampleClass Declaration
\*---------------------------------------------------------------------------*/
```

1.2 The *.H* Files

- header file spacing
  - Leave two empty lines between sections (as per functions in the *.C* file etc)
- use `//- Comment` comments in header file
  - add descriptions to class data and functions
- destructor
  - If adding a comment to the destructor - use `//-` and code as a normal function:

    ```
    //- Destructor
    ~className();
    ```
- inline functions
  - Use inline functions where appropriate in a separate *classNameI.H* file. Avoid cluttering the header file with function bodies.

1.3 The *.C* Files

- Do not open/close namespaces in a *.C* file
  - Fully scope the function name, i.e.

    ```
    Foam::returnType Foam::className::functionName()
    ```

    NOT

```
namespace Foam
{
    ...
    returnType className::functionName()
    ...
}
```

EXCEPTION

When there are multiple levels of namespace, they may be used in the $.C$ file, i.e.

```
namespace Foam
{
namespace compressible
{
namespace RASModels
{
    ...
} // End namespace RASModels
} // End namespace compressible
} // End namespace Foam
```

- Use two empty lines between functions

1.4 Coding Practice

- passing data as arguments or return values.
  - Pass bool, label and scalar as copy, anything larger by reference.
- const
  - Use everywhere it is applicable.
- variable initialisation using

```
const className& variableName = otherClass.data();
```

NOT

```
const className& variableName(otherClass.data());
```

- virtual functions
  - If a class is virtual, make all derived classes virtual.

1.5 Conditional Statements

```
if (condition)
{
    code;
}
```

OR

```
if
(
   long condition
)
{
    code;
}
```

NOT (no space between `if` and `(` used)

```
if(condition)
{
    code;
}
```

1.6 `for` and `while` Loops

```
for (i = 0; i < maxI; i++)
{
    code;
}
```

OR

```
for
(
    i = 0;
    i < maxI;
    i++
)
{
    code;
}
```

NOT this (no space between `for` and `(` used)

```
for(i = 0; i < maxI; i++)
{
    code;
}
```

Note that when indexing through iterators, it is often slightly more efficient to use the pre-increment form. Eg, `++iter` instead of `iter++`

1.7 `forAll`, `forAllIter`, `forAllConstIter`, etc. loops

like `for` loops, but

```
forAll(
```

NOT

```
forAll (
```

Using the `forAllIter` and `forAllConstIter` macros is generally advantageous - less typing, easier to find later. However, since they are macros, they will fail if the iterated object contains any commas.

The following will FAIL!:

```
forAllIter(HashTable<labelPair, edge, Hash<edge> >, foo, iter)
```

These convenience macros are also generally avoided in other container classes and OpenFOAM primitive classes.

1.8 Splitting Over Multiple Lines

1.8.1 Splitting return type and function name

- split initially after the function return type and left align

- do not put `const` onto its own line - use a split to keep it with the function name and arguments.

  ```
  const Foam::longReturnTypeName&
  Foam::longClassName::longFunctionName const
  ```

  NOT

  ```
  const Foam::longReturnTypeName&
      Foam::longClassName::longFunctionName const
  ```

  NOR

  ```
  const Foam::longReturnTypeName& Foam::longClassName::longFunctionName
  const
  ```

  NOR

  ```
  const Foam::longReturnTypeName& Foam::longClassName::
  longFunctionName const
  ```

- if it needs to be split again, split at the function name (leaving behind the preceding scoping =::=s), and again, left align, i.e.

  ```
  const Foam::longReturnTypeName&
  Foam::veryveryveryverylongClassName::
  veryveryveryverylongFunctionName const
  ```

1.8.2 Splitting long lines at an "="

Indent after split

```
variableName =
    longClassName.longFunctionName(longArgument);
```

OR (where necessary)

```
variableName =
    longClassName.longFunctionName
    (
        longArgument1,
        longArgument2
    );
```

NOT

```
variableName =
longClassName.longFunctionName(longArgument);
```

NOR

```
variableName = longClassName.longFunctionName
(
    longArgument1,
    longArgument2
);
```

1.9 Maths and Logic

- operator spacing

  ```
  a + b, a - b
  a*b, a/b
  a & b, a ^ b
  a = b, a != b
  a < b, a > b, a >= b, a <= b
  a || b, a && b
  ```

- splitting formulae over several lines

  Split and indent as per "splitting long lines at an =" with the operator on the lower line. Align operator so that first variable, function or bracket on the next line is 4 spaces indented i.e.

  ```
  variableName =
      a*(a + b)
     *exp(c/d)
     *(k + t);
  ```

This is sometimes more legible when surrounded by extra parentheses:

```
variableName =
(
    a*(a + b)
   *exp(c/d)
   *(k + t)
);
```

- splitting logical tests over several lines

  outdent the operator so that the next variable to test is aligned with the four space indentation, i.e.

  ```
  if
  (
      a == true
   && b == c
  )
  ```

1.10 General

- For readability in the comment blocks, certain tags are used that are translated by pre-filtering the file before sending it to Doxygen.
- The tags start in column 1, the contents follow on the next lines and indented by 4 spaces. The filter removes the leading 4 spaces from the following lines until the next tag that starts in column 1.
- The 'Class' and 'Description' tags are the most important ones.
- The first paragraph following the 'Description' will be used for the brief description, the remaining paragraphs become the detailed description.

  For example,

  ```
  Class
      Foam::myClass

  Description
      A class for specifying the documentation style.

      The class is implemented as a set of recommendations that may
      sometimes be useful.
  ```

- The class name must be qualified by its namespace, otherwise Doxygen will think you are documenting some other class.
- If you don't have anything to say about the class (at the moment), use the namespace-qualified class name for the description. This aids with finding these under-documented classes later.

```
Class
    Foam::myUnderDocumentedClass

Description
    Foam::myUnderDocumentedClass
```

- Use 'Class' and 'Namespace' tags in the header files. The Description block then applies to documenting the class.
- Use 'InClass' and 'InNamespace' in the source files. The Description block then applies to documenting the file itself.

```
InClass
    Foam::myClass

Description
    Implements the read and writing of files.
```

1.11 Doxygen Special Commands

Doxygen has a large number of special commands with a $=\bar{p}$refix.

Since the filtering removes the leading spaces within the blocks, the Doxygen commmands can be inserted within the block without problems.

```
InClass
    Foam::myClass

Description
    Implements the read and writing of files.

    An example input file:
    \verbatim
        patchName
        {
            type        myPatchType;
            refValue    100;
            value       uniform 1;
        }
    \endverbatim

    Within the implementation, a loop over all patches is done:
    \code
        forAll(patches, patchI)
        {
            ...  // some operation
        }
    \endcode
```

1.12 HTML Special Commands

Since Doxygen also handles HTML tags to a certain extent, the angle brackets need quoting in the documentation blocks. Non-HTML tags cause Doxygen to complain, but seem to work anyhow.

eg,

- The template with type `<HR>` is a bad example.
- The template with type `\<HR\>` is a better example.
- The template with type `<Type>` causes Doxygen to complain about an unknown html type, but it seems to work okay anyhow.

1.13 Documenting Namespaces

- If namespaces are explictly declared with the `Namespace()` macro, they should be documented there.
- If the namespaces is used to hold sub-models, the namespace can be documented in the same file as the class with the model selector. eg,

  ```
  documented namespace 'Foam::functionEntries' within the
  class 'Foam::functionEntry'
  ```

- If nothing else helps, find some sensible header. eg,

  ```
  namespace 'Foam' is documented in the foamVersion.H file
  ```

1.14 Documenting typedefs and classes defined via macros

. . . not yet properly resolved

1.15 Documenting Applications

Any number of classes might be defined by a particular application, but these classes will not, however, be available to other parts of OpenFOAM. At the moment, the sole purpuse for running Doxygen on the applications is to extract program usage information for the '-doc' option.

The documentation for a particular application is normally contained within the first comment block in a $.C$ source file. The solution is this to invoke a special filter for the "/applications/{solver,utilities}/" directories that only allows the initial comment block for the $.C$ files through.

The layout of the application documentation has not yet been finalized, but foamToVTK shows an initial attempt.

1.16 Orthography

Given the origins of OpenFOAM, the British spellings (eg, neighbour and not neighbor) are generally favoured.

Both '-ize' and the '-ise' variant are found in the code comments. If used as a variable or class method name, it is probably better to use '-ize', which is considered the main form by the Oxford University Press. Eg,

```
myClass.initialize()
```

The word "its" (possesive) vs. "it's" (colloquial for "it is" or "it has") seems to confuse non-native (and some native) English speakers. It is better to donate the extra keystrokes and write "it is" or "it has". Any remaining "it's" are likely an incorrect spelling of "its".