OpenFOAM
The OpenFOAM Foundation

Coding Style Guide

Table of Contents

- 1. Code
  - 1.1. General
  - 1.2. The *.H* Header Files
  - 1.3. The *.C* Source Files
  - 1.4. Coding Practice
  - 1.5. Conditional Statements
  - 1.6. `for` and `while` Loops
  - 1.7. `forAll`, `forAllIter`, `forAllConstIter`, *etc.* loops
  - 1.8. Splitting Over Multiple Lines
    - 1.8.1. Splitting return type and function name
    - 1.8.2. Splitting long lines at an "="
  - 1.9. Maths and Logic
- 2. Documentation
  - 2.1. General
  - 2.2. Doxygen Special Commands
  - 2.3. HTML Special Commands
  - 2.4. Documenting Namespaces
  - 2.5. Documenting Applications
  - 2.6. Orthography
  - 2.7. References

1 Code

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 braces. 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 braces.
- Stream output
  - **<<** is always four characters after the start of the stream, so that the **<<** symbols align, i.e.

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

    so

    ```
    WarningInFunction
        << "Warning message"
    ```

    not

    ```
    WarningInFunction
    << "Warning message"
    ```

- Remove 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 should be centred

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

not

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

1.2 The *.H* Header Files

- Header file spacing
  - Leave two empty lines between sections (as per functions in the *.C* file *etc.*)
- Use `//- Comment` comments in header file to add descriptions to class data and functions do be included in the Doxygen documentation:
  - Text on the line starting with `//-` becomes the Doxygen brief description;
  - Text on subsequent lines becomes the Doxygen detailed description *e.g.*

```
//- A function which returns a thing
//  This is a detailed description of the function
//  which processes stuff and returns other stuff
//  depending on things.
thing function(stuff1, stuff2);
```

  - List entries start with `-` or `-#` for numbered lists but cannot start on the line immediately below the brief description so

```
//- Compare triFaces
//  Returns:
//  -  0: different
//  - +1: identical
//  - -1: same face, but different orientation
static inline int compare(const triFace&, const triFace&);
```

or

```
//- Compare triFaces returning 0, +1 or -1
//
//  -  0: different
//  - +1: identical
//  - -1: same face, but different orientation
static inline int compare(const triFace&, const triFace&);
```

not

```
//- Compare triFaces returning 0, +1 or -1
//  -  0: different
//  - +1: identical
//  - -1: same face, but different orientation
static inline int compare(const triFace&, const triFace&);
```

  - List can be nested for example

```
//- Search for \em name
//  in the following hierarchy:
//  -# personal settings:
//    - ~/.OpenFOAM/\<VERSION\>/
//      <em>for version-specific files</em>
//    - ~/.OpenFOAM/
//      <em>for version-independent files</em>
//  -# site-wide settings:
//    - $WM_PROJECT_INST_DIR/site/\<VERSION\>
//      <em>for version-specific files</em>
//    - $WM_PROJECT_INST_DIR/site/
//      <em>for version-independent files</em>
//  -# shipped settings:
//    - $WM_PROJECT_DIR/etc/
//
//  \return the full path name or fileName() if the name cannot be fou
//  Optionally abort if the file cannot be found
fileName findEtcFile(const fileName&, bool mandatory=false);
```

  - For more details see the Doxygen documentation.

- Destructor
  - When 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* Source 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 to avoid excessive clutter, 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, scalar and other primitive types 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 *e.g.* 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
  )
  ```

2 Documentation

2.1 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.
```

2.2 Doxygen Special Commands

Doxygen has a large number of special commands with a **** prefix.

Since the filtering removes the leading spaces within the blocks, the Doxygen commands 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
```

2.3 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. *e.g.*,

- 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.

2.4 Documenting Namespaces

- If namespaces are explicitly 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. *e.g.*,

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

- If nothing else helps, find some sensible header. *e.g.*,

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

2.5 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 purpose 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.

2.6 Orthography

Given the origins of OpenFOAM, the British spellings (*e.g.*, 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. *e.g.*,

```
myClass.initialize()
```

2.7 References

References provided in the `Description` section of the class header files should be formatted in the APA (American Psychological Association) style *e.g.* from `kEpsilon.H`

```
Description
    Standard k-epsilon turbulence model for incompressible and compressible
    flows including rapid distortion theory (RDT) based compression term.
```

```
Reference:
\verbatim
    Standard model:
        Launder, B. E., & Spalding, D. B. (1972).
        Lectures in mathematical models of turbulence.

        Launder, B. E., & Spalding, D. B. (1974).
        The numerical computation of turbulent flows.
        Computer methods in applied mechanics and engineering,
        3(2), 269-289.

    For the RDT-based compression term:
        El Tahry, S. H. (1983).
        k-epsilon equation for compressible reciprocating engine flows.
        Journal of Energy, 7(4), 345-353.
\endverbatim
```

The APA style is a commonly used standard and references are available in this format from many sources including Citation Machine and Google Scholar.

Date: 2011-2016

Created: 2016-04-20 Wed 10:24