AMDiS C++ code style directive

Thomas Witkowski

May 27, 2010

1 General recommendations

1. The main goal of this document is to assure a unique code style in AMDiS. This should make the code more readable and easier to maintain.
2. The code should be readable independently of any text editor and IDE.
3. Any violation to the guide is allowed if it enhances readability.
4. Not all source files of AMDiS are already converted to this style. If you are bored, you are free to do it.

2 Naming conventions

1. All names should be written in English.
2. Names representing types must be in mixed case string with upper case.

   `FiniteElemSpace, ProblemInstat`

3. Variable names must be in mixed case starting with lower case.

   `refinementManager, fileWriter`

4. Named constants (including enumeration values) must be all uppercase using underscore to separate words.

   `CALL_LEAF_EL, GRD_PHI`

5. Names representing methods or functions must written in mixed case starting with lower case.

   `refineMesh(), getTimestep()`

6. Abbreviations and acronyms must not be uppercase when used as name.

```
getFeSpace()      // NOT: getFESpace()
listDof           // NOT: listDOF
```

7. Do not use underscores to indicate variable scopes.
8. The prefix n should be used for variables representing a number of objects.

```
nPoints, nBasisFcts
```

9. Naming pointers specifically should be avoided.

```
Line *line          // NOT: Line *pLine
Element *el         // NOT: Element *ptrEl
```

3 Files

1. C++ header files should have the extension .h. Source files should have the extension .cc.
2. A class should be declared in a header file and defined in a source file where the name of the files match the name of the class.
3. All definitions should reside in source files. Usually, get- and set-function can be excluded from this rule.
4. File content must be kept within 85 columns.
5. Include statements must be located at the top of a file only.

4 Statements

1. Type conversions must always be done explicitly. Never rely on implicit type conversion. Do not use C-style type conversion.

```
floatValue = static_cast<float>(intValue);
// NOT: floatValue = intValue;
// NOT: floatValue = (float)intValue;
```

2. Variables should be declared where they are used for the first time.
3. Variables should be initialized where they are declared.
4. Use of global variables should be minimized.
5. Class variables should never be declared public.
6. Control variables in for() loops should be declared within the for-construction.

```
for (int i = 0; i < 10; i++)

// NOT:
int i;
for (i = 0; i < 10; i++)
```

7. Complex conditional expressions must be avoided. Introduce temporary boolean variables instead.
8. The conditional should be put on a separate line.

```
if (isOkay)    // NOT: if (isOkay) doIt();
  doIt();
```

9. Executable statements in conditionals must be avoided.

10. Floating point constants should always be written with decimal point and at least one decimal.

```
double total = 0.0;    // NOT: double total = 0;
double speed = 3.0e8; // NOT: double speed = 3e8;
```

5 Layout

1. Block layout should be as follows

```
if (isOkay) {
  doSomething();
  ...
} else {
  doSomethingElse();
  ...
}

for () {
  runCode();
  ...
}
```

2. Single statement if-else, for or while statements can be written without brackets.

```
for (int i = 0; i < 5; i++)
  doSomething();

// NOT:
for (int i = 0; i < 5; i++) {
  doSomething();
}
```

3. The class declarations should have the following form:

```
class SomeClass : public BaseClass
{
public:
  ...
protected:
  ...
private:
  ...
}
```

4. Method definitions should have the following form:

```
void someMethod()
{
  ...
}
```

5. General white spaces:

   - Conventional operators should be surrounded by a space character.
   - C++ reserved words should be followed by a white space.
   - Commas should be followed by a white space.
   - Colons should be surrounded by white space.
   - Semicolons in for statements should be followed by a space character.

```
a = (b + c) * d;     // NOT: a=(b+c)*d
while (true)         // NOT: while(true)
doSomething(a, b, c, d);          // NOT: doSomething(a,b,c,d);
for (int i = 0; i < 10; i++) {    // NOT: for(int i=0;i<10;i++){
```

6. Methods should be separated by two blank lines.

6 Comments

1. Tricky code should not be commented but rewritten!
2. All comments should be written in English.
3. Class and method header comments should follow the JavaDoc conventions.