Linear solver execution path of the laplacianFoam/flange tutorial in OpenFOAM-v1906 – a GDB debugging tutorial

© Håkan Nilsson

September 2019

Contents

Listings

1 Introduction

It is quite often difficult to know exactly which source code is used when running an OpenFOAM executable. OpenFOAM is designed to be flexible for the user, which makes the execution highly depending on the user settings.

This document shows how the debugging tool GDB can be used to show the exact path taken by the code. It is also a basic tutorial in the use of GDB to debug OpenFOAM. For this to work it is assumed that the environment for the Debug version of OpenFOAM-v1906 is activated.

Lines starting with `$` are manual command line entries, and lines starting with `(gdb)` are manual entries when executing GDB. Other lines are terminal output. The most important pieces of code are shown by listings.

2 Starting GDB for laplacianFoam and listing the source code

We set up the `flange` case and start GDB with the `laplacianFoam` solver:

```
$ rm -rf $FOAM_RUN/flange
$ cp -r $FOAM_TUTORIALS/basic/laplacianFoam/flange $FOAM_RUN
$ cd $FOAM_RUN/flange
$ ansysToFoam flange.ans -scale 0.001 > log.ansysToFoam 2>&1
$ gdb laplacianFoam
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from laplacianFoam...done.
```

At this point GDB is prepared to run the `laplacianFoam` solver, and it has access to all the code of both the top-level solver and the OpenFOAM libraries it uses. We can view the top-level code using the `list` command (or its short version `l`). To make sure that GDB does not exclude any commented header we can specify the line number (below: `l 1`). To repeat the command we can simply press Enter to march through the top-level code:

```
(gdb) l 1
1   /*---------------------------------------------------------------------------*\
2     =========                 |
3     \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4      \\    /   O peration     |
5       \\  /    A nd           | Copyright (C) 2004-2011 OpenCFD Ltd.
6        \\/     M anipulation  |
7   -------------------------------------------------------------------------------
8                               | Copyright (C) 2011-2017 OpenFOAM Foundation
9   -------------------------------------------------------------------------------
10  License
.
.
.
96              TEqn.solve();
.
.
.
107     return 0;
108 }
109
110
111 // ************************************************************************* //
```

We see that the linear solver is executed by `TEqn.solve()`, at line 96, so that is where we will start our journey.

Other ways to list the code are (try them):

- `l -` to march backwards when repeatedly pressing Enter. The default forward command is `l +`.
- `l 96` to specify which line you want at the center
- `l 80,100` to show the lines 80-100
- The number of lines that are shown by the `l` command can be changed by `set listsize <count>`, where `<count>` is the number of lines to be shown.

It should be noted that I have experienced problems listing code using line numbers inside templated classes. Then it is better to march back an forth using `l +` and `l -`.

3 Path taken by the TEqn.solve() function

We start by setting a breakpoint (`b`) at the call to the `TEqn.solve()` function, which tells GDB to stop the execution at that line, before executing that line. We run `laplacianFoam` through GDB (`run`), and we see that the execution stops at the breakpoint.

```
(gdb) b laplacianFoam.C:96
Breakpoint 1 at 0x2e100: file laplacianFoam.C, line 96.
(gdb) run
Starting program: /home/oscfd/OpenFOAM/OpenFOAM-v1906/platforms/linux64GccDPInt32Debug/bin/
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 88d188709a-20190806 OPENFOAM=1906 patch=190724
Arch   : "LSB;label=32;scalar=64"
Exec   : /home/oscfd/OpenFOAM/OpenFOAM-v1906/platforms/linux64GccDPInt32Debug/bin/laplacianF
Date   : Sep 19 2019
Time   : 14:36:47
Host   : oscfd-VirtualBox
PID    : 15854
I/O    : uncollated
Case   : /home/oscfd/OpenFOAM/oscfd-v1906/run/flange
nProcs : 1
trapFpe: Floating point exception trapping enabled (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster (fileMo
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0


SIMPLE: no convergence criteria found. Calculations will run for 3 steps.

Reading field T

Reading transportProperties

Reading diffusivity DT

No finite volume options present

Calculating temperature distribution

Time = 0.005


Breakpoint 1, main (argc=1, argv=0x7fffffffd0c8) at laplacianFoam.C:96
96              TEqn.solve();
```

There are several ways to continue the execution. Typing...

- `c` will simply continue the execution until the next breakpoint or the end of the execution. In our case it will simply get back to the same location at the next loop, since we only have that breakpoint. You can try it if you like, since the `c` command will just bring us back to the point where we will continue later.
- `s` will step to the next line of the execution, also entering into functions.
- `n` will step to the next line of the execution, but not enter into functions (i.e. stay in the same file).

Each of the above commands can be followed by a number to repeat the command that number of times (e.g. `c 2`). You can repeat the previous command by just pressing Enter again. If you issued either of the commands `s` or `n` you can use the command `c` to get back to the breakpoint at the next loop. Now make sure that you are at the breakpoint, as at the end of the listing above.

We want to figure out where we end up when the `solve()` function is called, so we type `s` to step into the function:

```
(gdb) s
Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:321
321 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve()
```

We see that we end up at line 321 in `fvMatrixSolve.C`. If we forget where we are we can type `where` to show where we are:

```
(gdb) where
#0  Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:321
#1  0x0000555555582119 in main (argc=1, argv=0x7fffffffd0c8) at laplacianFoam.C:96
```

We see that we are at line 321 in `fvMatrixSolve.C`, and we also see that we came there from line 96 in `laplacianFoam.C`. This is the *call stack*, and the numbers to the left are the *frame* numbers. We can go to another frame and list the lines:

```
(gdb) f 1
#1  0x0000555555582119 in main (argc=1, argv=0x7fffffffd0c8)
    at laplacianFoam.C:96
96              TEqn.solve();
(gdb) l
91               ==
92                   fvOptions(T)
93               );
94
95               fvOptions.constrain(TEqn);
96               TEqn.solve();
97               fvOptions.correct(T);
98           }
99
100          #include "write.H"
(gdb) where
#0  Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:321
#1  0x0000555555582119 in main (argc=1, argv=0x7fffffffd0c8)
    at laplacianFoam.C:96
(gdb) f 0
#0  Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:321
321 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve()
(gdb) l
```

```
316     return solve(fvMat_.solverDict());
317 }
318
319
320 template<class Type>
321 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve()
322 {
323     return this->solve(solverDict());
324 }
325
(gdb) where
#0  Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:321
#1  0x0000555555582119 in main (argc=1, argv=0x7fffffffd0c8)
    at laplacianFoam.C:96
```

We see that the `where` command gives the same output irrespectively in which frame we are, which means that we are only moving between frames to look at the code, and that the execution remains at the same location. We can as well use the commands

- `up` to move up in the call stack
- `down` to move down in the call stack

The `f 0` command can as well be used to reset the line that is shown by the command `l`.

We issue the command `f 0` and use the command `l` to have a look at the `solve()` function that is called, in the listing below.

Listing 1: fvMatrixSolve.C, line 321

```
320 template<class Type>
321 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve()
322 {
323     return this->solve(solverDict());
324 }
```

We see that the `solve()` function calls another `solve` function, using the original object `TEqn` (`this`) and the output of a call to a function `solverDict()`. We are not interested at this point to figure out the details of the function `solverDict()`, so we will just step into it and use the command `fin` to finish that function call (which also gives a long output, showing the return of that function). After that we are still at line 321 in `fvMatrixSolve.C`, since the function `solve` is next to be executed. Then we are interested in where that function is located, so we step into that function.

```
(gdb) s 2
Foam::fvMatrix<double>::solverDict (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrix.C:1013
1013     const Foam::dictionary& Foam::fvMatrix<Type>::solverDict() const
(gdb) fin
Run till exit from #0  Foam::fvMatrix<double>::solverDict (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrix.C:1013
0x0000555555586d11 in Foam::fvMatrix<double>::solve (this=0x7fffffffc910)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:323
323     return this->solve(solverDict());
Value returned is $1 =
    (const Foam::dictionary &) @0x555555af24f0: {<Foam::ILList<Foam::DLListBase, Foam::entr
(gdb) s
Foam::fvMatrix<double>::solve (this=0x7fffffffc910, solverControls=...) at /home/oscfd/Openl
296 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve
```

We end up at line 296 in the same file (`fvMatrixSolve.C`). From the listing below we see that yet another function `solve` is called.

Listing 2: fvMatrixSolve.C, line 296

```
295 template<class Type>
296 Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solve
297 (
298     const dictionary& solverControls
299 )
300 {
301     return psi_.mesh().solve(*this, solverControls);
302 }
```

We step a line in the current file (`n`), so that we are at line 301. Then we make a step (`s`), going into functions, realizing that we end up in the `mesh()` function that we are not interested in, so we finish that function by the `fin` command and again step into the next function call.

```
(gdb) n
301       return psi_.mesh().solve(*this, solverControls);
(gdb) s
Foam::DimensionedField<double, Foam::volMesh>::mesh (this=0x7fffffffc350)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/DimensionedFieldI.H:42
42        return mesh_;
(gdb) fin
Run till exit from #0  Foam::DimensionedField<double, Foam::volMesh>::mesh (this=0x7fffffff
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/DimensionedFieldI.H:42
0x000055555558ba55 in Foam::fvMatrix<double>::solve (this=0x7fffffffc910, solverControls=.
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/finiteVolume/lnInclude/fvMatrixSolve.C:301
301       return psi_.mesh().solve(*this, solverControls);
Value returned is $2 =
  (const Foam::GeoMesh<Foam::fvMesh>::Mesh &) @0x555555a32b40: {<Foam::polyMesh> = {<Foam::
(gdb) s
Foam::fvMesh::solve (this=0x555555a32b40, m=..., dict=...) at fvMesh/fvMesh.C:440
440 {
```

We end up at line 440 in `fvMesh.C`. We see below that we are now about to call a function named `solveSegregatedOrCoupled`.

Listing 3: fvMesh.C, line 440

```
435 Foam::SolverPerformance<Foam::scalar> Foam::fvMesh::solve
436 (
437     fvMatrix<scalar>& m,
438     const dictionary& dict
439 ) const
440 {
441     // Redirect to fvMatrix solver
442     return m.solveSegregatedOrCoupled(dict);
443 }
```

We step into that function to figure out where it is located and where next to go.

```
(gdb) s 2
Foam::fvMatrix<double>::solveSegregatedOrCoupled (this=0x7fffffffc910, solverControls=...)
    at lnInclude/fvMatrixSolve.C:60
60  Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solveSegregatedOrCoupled
```

We end up at line 60 in `fvMatrixSolve.C`. The listing below shows that the kind of solver that will be used is either segregated of coupled, see line 92. We need to figure out which one will be used in our case.

Listing 4: fvMatrixSolve.C, line 60

```
59  template<class Type>
60  Foam::SolverPerformance<Type> Foam::fvMatrix<Type>::solveSegregatedOrCoupled
61  (
62      const dictionary& solverControls
63  )
64  {
65      word regionName;
66      if (psi_.mesh().name() != polyMesh::defaultRegion)
67      {
68          regionName = psi_.mesh().name() + "::";
69      }
70      addProfiling(solve, "fvMatrix::solve." + regionName + psi_.name());
71
72      if (debug)
73      {
74          Info.masterStream(this->mesh().comm())
75              << "fvMatrix<Type>::solveSegregatedOrCoupled"
76                 "(const dictionary& solverControls) : "
77                 "solving fvMatrix<Type>"
78              << endl;
79      }
80
81      label maxIter = -1;
82      if (solverControls.readIfPresent("maxIter", maxIter))
83      {
84          if (maxIter == 0)
85          {
86              return SolverPerformance<Type>();
87          }
88      }
89
90      word type(solverControls.lookupOrDefault<word>("type", "segregated"));
91
92      if (type == "segregated")
93      {
94          return solveSegregated(solverControls);
95      }
96      else if (type == "coupled")
97      {
98          return solveCoupled(solverControls);
99      }
100     else
101     {
102         FatalIOErrorInFunction(solverControls)
103             << "Unknown type " << type
104             << "; currently supported solver types are segregated and coupled"
105             << exit(FatalIOError);
106
107         return SolverPerformance<Type>();
108     }
109 }
```

We set a breakpoint at line 92 and continue to that line. Another way of continuing to line 92 without setting a breakpoint would be to issue the command `u 92`, but I had a problem with that in my installation at this particular point (maybe due to templating, as we will see).

```
(gdb) b fvMatrixSolve.C:92
Breakpoint 2 at 0x7ffff63922b6: fvMatrixSolve.C:92. (4 locations)
(gdb) c
Continuing.

Breakpoint 2, Foam::fvMatrix<double>::solveSegregatedOrCoupled (this=0x7fffffffc910, solver
    at lnInclude/fvMatrixSolve.C:92
92          if (type == "segregated")
```

Now we have two breakpoints, so let's examine the breakpoints we have and delete the newly created one (if you like you can keep it to easily get back to this location):

```
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555582100 in main(int, char**)
    at laplacianFoam.C:96
    breakpoint already hit 3 times
2       breakpoint     keep y   <MULTIPLE>
    breakpoint already hit 1 time
2.1                    y     0x00007ffff63922b6 in Foam::fvMatrix<double>::solveSegreg
        at lnInclude/fvMatrixSolve.C:92
2.2                    y     0x00007ffff63927ca in Foam::fvMatrix<Foam::Vector<double>
        at lnInclude/fvMatrixSolve.C:92
2.3                    y     0x00007ffff6392d30 in Foam::fvMatrix<Foam::SymmTensor<dou
        at lnInclude/fvMatrixSolve.C:92
2.4                    y     0x00007ffff6393296 in Foam::fvMatrix<Foam::Tensor<double>
        at lnInclude/fvMatrixSolve.C:92
(gdb) delete 2
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000555555582100 in main(int, char**)
    at laplacianFoam.C:96
    breakpoint already hit 3 times
```

We can in particular see that the second breakpoint has four options, depending on the content type of the `fvMatrix` (due to templating). That may be the reason for my problem above.

We check (`print`, or `p`) the value of the object named `type`, and see that it is `segregated`. Then we step into that particular function.

```
(gdb) p type
$3 = {<Foam::string> = {<std::__cxx11::basic_string<char, std::char_traits<char>, std::allo
    static null = {<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator
      static null = <same as static member of an already seen type>}}, static typeName = 0x
  static null = {<Foam::string> = {<std::__cxx11::basic_string<char, std::char_traits<char>
    static null = {<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocat
      static null = <same as static member of an already seen type>}}, static typeName =
(gdb) n
94          return solveSegregated(solverControls);
(gdb) s
Foam::fvMatrix<double>::solveSegregated (this=0x7fffffffc910, solverControls=...)
    at fvMatrices/fvScalarMatrix/fvScalarMatrix.C:152
152 {
```

We end up at line 152 in `fvScalarMatrix.C`. We see in the listing below (line 172) that there is a call to a function `New` of the class `solver` of the class `lduMatrix` (`lduMatrix::solver`). The solver to be used is specified by the user, in `system/fvSolution`. That solver is then used to call yet another `solve` function at line 180.

Listing 5: fvScalarMatrix.C, line 152

```
147 template<>
148 Foam::solverPerformance Foam::fvMatrix<Foam::scalar>::solveSegregated
149 (
150     const dictionary& solverControls
151 )
152 {
153     if (debug)
154     {
155         Info.masterStream(this->mesh().comm())
156             << "fvMatrix<scalar>::solveSegregated"
157                "(const dictionary& solverControls) : "
158                "solving fvMatrix<scalar>"
159             << endl;
160     }
161
162     GeometricField<scalar, fvPatchField, volMesh>& psi =
163        const_cast<GeometricField<scalar, fvPatchField, volMesh>&>(psi_);
164
165     scalarField saveDiag(diag());
166     addBoundaryDiag(diag(), 0);
167
168     scalarField totalSource(source_);
169     addBoundarySource(totalSource, false);
170
171     // Solver call
172     solverPerformance solverPerf = lduMatrix::solver::New
173     (
174         psi.name(),
175         *this,
176         boundaryCoeffs_,
177         internalCoeffs_,
178         psi_.boundaryField().scalarInterfaces(),
179         solverControls
180     )->solve(psi.primitiveFieldRef(), totalSource);
181
182     if (solverPerformance::debug)
183     {
184         solverPerf.print(Info.masterStream(mesh().comm()));
185     }
186
187     diag() = saveDiag;
188
189     psi.correctBoundaryConditions();
190
191     psi.mesh().setSolverPerformance(psi.name(), solverPerf);
192
193     return solverPerf;
194 }
```

We go there by setting a breakpoint and continuing the execution. Then we make a step and realize that we end up in `operator->()` that we are not interested in at the moment, so we finish that function using the `fin` command and make the next step. Then we end up in `primitiveFieldRef()`, so we repeat `fin` and step again.

```
(gdb) b fvScalarMatrix.C:180
Breakpoint 3 at 0x7ffff676a827: file fvMatrices/fvScalarMatrix/fvScalarMatrix.C, line 180.
```

```
(gdb) c
Continuing.

Breakpoint 3, Foam::fvMatrix<double>::solveSegregated (this=0x7fffffffc910, solverControls=
    at fvMatrices/fvScalarMatrix/fvScalarMatrix.C:180
180     )->solve(psi.primitiveFieldRef(), totalSource);
(gdb) s
Foam::autoPtr<Foam::lduMatrix::solver>::operator-> (this=0x7fffffffbb80)
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/autoPtrI.H:216
216 inline T* Foam::autoPtr<T>::operator->()
(gdb) fin
Run till exit from #0  Foam::autoPtr<Foam::lduMatrix::solver>::operator-> (this=0x7fffffffb
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/autoPtrI.H:216
0x00007ffff676a833 in Foam::fvMatrix<double>::solveSegregated (this=0x7fffffffc910, solverC
    at fvMatrices/fvScalarMatrix/fvScalarMatrix.C:180
180     )->solve(psi.primitiveFieldRef(), totalSource);
Value returned is $4 = (Foam::lduMatrix::solver *) 0x555555946cc0
(gdb) s
Foam::GeometricField<double, Foam::fvPatchField, Foam::volMesh>::primitiveFieldRef (this=0x
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/GeometricField.C:893
893     if (updateAccessTime)
(gdb) fin
Run till exit from #0  Foam::GeometricField<double, Foam::fvPatchField, Foam::volMesh>::pri
    at /home/oscfd/OpenFOAM/OpenFOAM-v1906/src/OpenFOAM/lnInclude/GeometricField.C:893
0x00007ffff676a851 in Foam::fvMatrix<double>::solveSegregated (this=0x7fffffffc910, solverC
    at fvMatrices/fvScalarMatrix/fvScalarMatrix.C:180
180     )->solve(psi.primitiveFieldRef(), totalSource);
Value returned is $5 =
      (Foam::DimensionedField<double, Foam::volMesh>::FieldType &) @0x7fffffffc430: {<Foam
(gdb) s
Foam::PCG::solve (this=0x555555946cc0, psi_s=..., source=..., cmpt=0 '\000') at matrices/ld
219 {
```

We finally end up at line 219 in the PCG solver, which is listed below, as stated in `system/fvSolution`.

Listing 6: PCG.C, line 219

```
213 Foam::solverPerformance Foam::PCG::solve
214 (
215     scalarField& psi_s,
216     const scalarField& source,
217     const direction cmpt
218 ) const
219 {
220     PrecisionAdaptor<solveScalar, scalar> tpsi(psi_s);
221     return scalarSolve
222     (
223         tpsi.ref(),
224         ConstPrecisionAdaptor<solveScalar, scalar>(source)(),
225         cmpt
226     );
227 }
```

Now we have seen the exact path taken to the solver specified by the user, which concludes the aim of this tutorial.
At this point we can type `quit` to quit GDB.

4 Learn more about GDB

Search the Internet for additional ways to use GDB. It is for instance possible to show and manipulate values of variables, and thus influence the execution of the code.

- See `http://www.gnu.org/software/gdb`
- See `https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`
- There are some interfaces to GDB:
  - See `http://www.gnu.org/software/gdb/links/`
  - `ddd`
  - `emacs`
- Macros for GDB/OpenFOAM: `http://openfoamwiki.net/index.php/Contrib_gdbOF`
- `eclipse` is another alternative (An Integrated Development Environment)
  See: `http://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_Eclipse`
- `Qt` is yet another alternative
  See: `http://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_QtCreator`