OpenFOAM structure

Gianluca Montenegro

Department of Energy
Politecnico di Milano

Requistes

- OpenFOAM is a CFD tool which runs on Linux operating systems, so having a Linux release already installed is strongly recommended
- The contents of this training course are related to a specific OpenFOAM release, namely OF-1.4.1. Hence having version 1.4.1 of OpenFOAM is another fundamental requisite
- Where do I get the OF release from?
  - www.opencfd.co.uk, download from this site the release developed by Henry Weller and developers from OpenCFD
  - http://powerlab.fsb.hr/ped/kturbo/OpenFOAM, contains the OF-1.4.1-dev release which has additional features developed by Hrvoje Jasak and by the OF community. There is also an svn repository where everybody can give his own contribution adding specific pieces of code such as classes and applications: http://openfoam-extend.svn.sourceforge.net
- None of this two releases are installed onto you laptop?

  Don't worry there is always a plan B

Plan B

You can use the Live DVD containing the CAElinux live evaluation which already includes the OpenFOAM 1.4.1 installation

Follow these steps:

- Boot the PC with the live DVD into your dvd drive
- Let the PC boot via DVD
- Wait until everything is loaded...it may take a while
- Login as caelinux... pw: caelinux
- You should be logged to your new environment in a while
- If you need to configure something login as root... pw:root, however do not waste your time now in PC administration

Try to list the content of your home directory and you will notice the `OpenFOAM` folder containing the OpenFOAM-1.4.1 installation

To begin working with OpenFOAM you need to load the environment variable, this is something you will learn soon

OpenFOAM installation

There are two ways of installing OpenFOAM onto your machines:

1. **local installation**: it is an installation dedicated to the single user. The user can modify the source of the code, create new applications, modify classes and so on without affecting the work of other OpenFOAM users
2. **network installation**: this installation is suitable when a group of people is supposed to use OpenFOAM without the need of any customization. The OpenFOAM version used is the same for all the users Customization is still possible, but it is preferable to do that outside of the main installation directory

OpenFOAM installation

OpenFOAM is distributed with a large set of precompiled applications but users also have the freedom to create their own or modify existing ones

Applications are split into two main categories:

1. **solvers** that are each designed to solve a specific problem in computational continuum mechanics
2. **utilities** that perform simple pre-and post-processing tasks, mainly involving data manipulation and algebraic calculations

OpenFOAM is divided into a set of precompiled libraries that are dynamically linked during compilation of the solvers and utilities

Libraries such as those for physical models are supplied as source code so that users may conveniently add their own models to the libraries

How to install: local installation

1. Chose the folder where you want to install OpenFOAM into: usually `$HOME/OpenFOAM`
2. Create a subfolder named `linux`
3. Download the source archive `OpenFOAM-1.4.1.General.gtgz` and untar it into the `$HOME/OpenFOAM` folder. This contains only the source file, you need to compile it to make it working onto your PC. The whole process can take two hours on a good PC. If you don't want to compile the whole lot, download and unpack also the precompiled version: `OpenFOAM-1.4.1.linuxGccDPOpt.gtgz`
4. Download the binary files for the gcc, pre-processing and post-processing tools and unpack them into the `$HOME/OpenFOAM/linux` folder:
   - gcc-4.2.1.bin.tgz
   - paraview-2.4.4.bin.tgz
   - java....
5. once the unpacking process has been done you only need to set accurately all the environmental variables and to source the correct files at login

How to install: sourcing environmental variables

The environment variable settings are contained in files in a `.OpenFOAM-1.4.1` directory in the OpenFOAM release:
`$HOME/OpenFOAM/OpenFOAM-1.4.1/.OpenFOAM-1.4.1`

1. if running bash or ksh (if in doubt type `echo $SHELL`), source the `.OpenFOAM-1.4.1/bashrc` file by adding the following line to the end of your `$HOME/.bashrc` file:

   ```
   . $HOME/OpenFOAM/OpenFOAM-1.4.1/.OpenFOAM-1.4.1/bashrc
   ```

   Then update the environment variables by sourcing the `$HOME/.bashrc` file by typing in the terminal:

   ```
   . $HOME/.bashrc
   ```

2. if running tcsh or csh, source the `.OpenFOAM-1.4.1/cshrc` file by adding the following line to the end of your `$HOME/.cshrc` file:

   ```
   source $HOME/OpenFOAM/OpenFOAM-1.4.1/.OpenFOAM-1.4.1/cshrc
   ```

   Then update the environment variables by sourcing the `$HOME/.cshrc` file by typing in the terminal:

   ```
   source $HOME/.cshrc
   ```

How to install: network installation

Make a local installation onto the server following the local installation guide, then on your client check the following steps:

1. If you want to customize OpenFOAM environmental variables copy the `$HOME/OpenFOAM/OpenFOAM-1.4.1/.OpenFOAM-1.4.1` directory into your home and source that path from you .cshrc or .bashrc file
2. the host name must be set - to test, type 'uname -a'. The running shell must be tcsh, csh, bash or ksh - to test type `echo $SHELL`
3. The user must be able to 'ping' the host machine itself (`<host>`) - to test, type `ping -c 1 <host>`. Check this by typing `grep <host> /etc/hosts` which should return a single line, typically of the form:
   `<IPaddress> <host>.<domain> <host>`
4. The machine must have one of (or both) remote (rsh) and secure shell (ssh) running on his/her account
5. Check the rsh executable actually exists, e.g. the path to the executable should be returned when typing `which rsh`. Check with the system administrator that rsh is enabled on the user's account
6. The .cshrc (or .bashrc) file should not contain errors that prevent it from executing fully at startup; all error messages during execution of the .bashrc (or .cshrc) file should be investigated and acted upon to eliminate them

Installation test and getting started

To test whether the installation process has been carried out successfully run the `foamInstallationTest` script:

```
/home/gmonte> foamInstallationTest
```

The output will be like this:

```
Checking basic setup...
-------------------------------------------------------------------------------
Shell:                  csh
Host:                   dastardly
OS:                     Linux version 2.6.22.17-0.1-bigsmp
User:                   gmonte
User_config:            /home/gmonte/.cshrc
Foam_config:            /home/gmonte/.OpenFOAM-1.4.1/cshrc sourced correct
-------------------------------------------------------------------------------
```

Testing the installation

Values of environment variables are also plotted such as:

```
Checking main FOAM env variables...
-----------------------------------------------------------------------
Environment_variable Set_to_file_or_directory                Valid
-----------------------------------------------------------------------
$WM_PROJECT_INST_DIR /homeTo check your installation setup, execute th
/gmonte/OpenFOAM                               yes       yes
$WM_PROJECT_USER_DIR /home/gmonte/OpenFOAM/gmonte-1.4.1       yes
$FOAM_JOB_DIR        /home/gmonte/OpenFOAM/jobControl         no
-----------------------------------------------------------------------

$WM_PROJECT_DIR      /home/gmonte/OpenFOAM/OpenFOAM-1.4.1     yes  yes
$FOAM_USER_APPBIN    ...1.4.1/applications/bin/linuxGccDPOpt  yes  yes
$FOAM_APPBIN         ...1.4.1/applications/bin/linuxGccDPOpt  yes  yes
$WM_DIR              ...gmonte/OpenFOAM/OpenFOAM-1.4.1/wmake  yes  yes
$FOAMX_PATH          ...ations/utilities/preProcessing/FoamX  yes   no
$CEI_HOME            /usr/local/ensight/CEI                   no
$JAVA_PATH           .../gmonte/OpenFOAM/linux/j2sdk1.4.2_05  no
$MICO_ARCH_PATH      .../mico-2.3.12/platforms/linuxGccDPOpt  yes  yes
-----------------------------------------------------------------------
```

Environment variables

Sourcing the `$HOME/.OpenFOAM-1.4.1/cshrc` file, OpenFOAM loads all the environmental variables it needs

For example:

- `$WM_PROJECT_INST_DIR` is the directory where OpenFOAM is installed.
- `$WM_PROJECT_USER_DIR` is the directory where the single user can store his/her own applications or where you usually put the `run` folder
- `$WM_PROJECT_DIR` is the directory which contains the OpenFOAM release we want to use

Environment variable: cshrc file

The cshrc (bashrc file for people using the bash shell) initializes all the variables needed for a correct use of OpenFOAM

```
setenv WM_PROJECT OpenFOAM
setenv WM_PROJECT_VERSION 1.4.1
setenv WM_PROJECT_LANGUAGE c++
```

They set respectively the name of the working project, the version used and the programming language

Typical usage is setting the installation directory name:

```
setenv WM_PROJECT_INST_DIR $HOME/$WM_PROJECT
```

It is possible to customize the installation directory by setting a different path, for example:

```
#setenv WM_PROJECT_INST_DIR /locals/mydisk/$WM_PROJECT
```

The OS variable `LOGNAME` is exploited to define the user working environment:

```
setenv WM_PROJECT_USER_DIR $HOME/$WM_PROJECT/
$LOGNAME-$WM_PROJECT_VERSION
```

Environment variable: important variables

In the same cshrc file it is possible to set other variables concerning the precision, the architecture, and so on

```
setenv WM_COMPILER Gcc
setenv WM_COMPILER_ARCH
setenv WM_COMPILER_LIB_ARCH
```

If `WM_COMPILER` is set to `""` the system compiler will be used

```
#setenv WM_PRECISION_OPTION SP
setenv WM_PRECISION_OPTION DP
```

Use `DP` or `SP` for respectively single and double precision

```
setenv WM_COMPILE_OPTION Opt
#setenv WM_COMPILE_OPTION Debug
#setenv WM_COMPILE_OPTION Prof
```

Set the wanted option to compile OpenFOAM with: `Opt`, `Debug` and `Prof` for respectively, optimized, debugging mode and profiling

```
setenv FOAM_SIGFPE
setenv FOAM_SETNAN
```

If they are set the control runtime the exception handling (`FOAM_SIGFPE`) and the usage of uninitialized memory (`FOAM_SETNAN`)

Environment variable: .cshrc file

Form the cshrc file in your `$HOME/.OpenFOAM-1.4.1` directory the following file is sourced:

```
$WM_PROJECT_DIR/.cshrc
```

It contains the definition of frequently used path and environment variables:

- the definition of the gcc compiler version to use

```
setenv WM_COMPILER_DIR $WM_PROJECT_INST_DIR/$WM_ARCH/gcc-4.2.1
```

- definition of path for user application, libraries and so on

```
setenv FOAM_USER_LIBBIN $WM_PROJECT_USER_DIR/lib/$WM_OPTIONS
setenv FOAM_USER_APPBIN $WM_PROJECT_USER_DIR/applications/bin/$WM_(

setenv FOAM_SRC $WM_PROJECT_DIR/src
setenv FOAM_LIB $WM_PROJECT_DIR/lib
setenv FOAM_LIBBIN $FOAM_LIB/$WM_OPTIONS

setenv FOAM_APP $WM_PROJECT_DIR/applications

setenv FOAM_TUTORIALS $WM_PROJECT_DIR/tutorials
setenv FOAM_UTILITIES $FOAM_APP/utilities
```

Environment variable: important variables

```
setenv WM_ARCH solaris
```

It sets the architecture of the machine you are working on. Possible types are: `linux`, `linuxIA64`,`solaris`,`sgiN32`,`sgi64`

Shortcuts to access a specific working directory are created by means of alises:

```
alias src 'cd $FOAM_SRC'
alias lib 'cd $FOAM_LIB'
alias run 'cd $FOAM_RUN'
alias foam 'cd $WM_PROJECT_DIR'
alias foamsrc 'cd $FOAM_SRC/$WM_PROJECT'
alias foamfv 'cd $FOAM_SRC/finiteVolume'
alias app 'cd $FOAM_APP'
alias util 'cd $FOAM_UTILITIES'
alias sol 'cd $FOAM_SOLVERS'
alias tut 'cd $FOAM_TUTORIALS'
```

OpenFOAM directory organization

Inside the WM_PROJECT_DIR it is possible to find all the folders containing the OpenFOAM installation

The user can move to that dyrectoryt by simply typing the foam alias command

The structure of that folder will the following:

```
$WM_PROJECT_DIR
 |------------> applications
 |------------> bin
 |------------> doc
 |------------> lib
 |------------> src
 |------------> tutorials
 |------------> wmake
```

Additional files are present in the WM_PROJECT_DIR folder, the most important one is Allwmake

Running that command it is possible to compile the whole installation, namely, libraries, applications and utilities

The applications folder

Change to the applications folder running the `app` command

```
$WM_PROJECT_DIR/applications
 |------------> bin
 |------------> solvers
 |------------> test
 |------------> utilities
```

Here is a short description of the applications folder contents:

- `bin` contains the binaries generated by compiling the applications (`$FOAM_APPBIN`)
- `solvers` contains different folders grouping solvers for combustion, compressible, DNSandLES, electromagnetics, financial, heatTransfer, incompressible, multiphase and stress analysis problems This folder can be reached with the `sol` shortcut
- `utilities` contains utilities for postprocessing, preprocessing, mesh manipulation and so on
- `test` contains several small applications to test and learn the usage of certain c++ objects

The `Allwmake` will compile all the content of `solvers` and `utilities`

The doc folder

The `doc` folder contains the documentation of OpenFOAM:

- Programmers and User guide of OpenFOAM
- Doxygen generated documentation in html format

The documentation generated by doxygen can be viewed by firefox

Doxygen can also be customized to generate the documentation of customized classes and user generated classes

The generation of the html source guide can be realized by running the `Allwmake` command from inside the `Doxygen` folder:

```
$WM_PROJECT_DIR/doc/Doxygen
```

The bin, lib and tutorials folders

The bin folder contains all the binaries of OpenFOAM related applications such as paraFoam for the post-processing, foamX for the pre-processing or other utilities like the foamNew, foamLog ...

The lib directory contains the binaries generated by compiling the libraries provided by the the installation in the src folder.

The tutorial folder contains OpenFOAM cases to be executed with the associated application.

The tutorial folder can be accessed by typing the tut command in the shell In the same directory there are other files:

- Allrun it generates automatically the mesh for each case and run the case
- Allclean it removes all the files created by the application run and by the post-processing tools

The wmake folder

All the options and scripts to compile the whole OpenFOAM installation or single applications are contained in this directory

The two most important scripts are the `wmake` and `wclean` commands:

- `wmake` allows you to compile every application you create according to the information written in the `files` and `options` files of the `Make` directory
- `wclean` cleans up the `wmake` control directory Make and removes the include directory generated for libraries

Once you have created the application (*.H and *.C files) you can generate the `Make` directory by using the `wmakeFilesAndOptions` script. Libraries are not included automatically but it creates a starting point to work on.

The `wmake` folder contains other script such as `wcleanMachine`, `wcleanAll` which clean the directory dependent on the machine type or all the machine dependent folders respectively

The src folder

This folder contains the source code for all the libraries

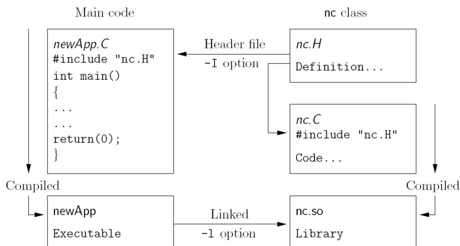
It is divided in different subfolders each of them can contain several libraries

The most relevant are:

- `finiteVolume`, if compiled with the command `wmake libso` it generates the lbrary `libfiniteVolume`. This library provides all the classes needed for the finiteVolume discretization, such as fvMesh, divergence, laplacian, gradient discretization operators and matrices solvers
- `OpenFOAM`,when compiled with the `wmake libso` command it generates the `libOpenFOAM` library, which includes the definitions of the containers used for the operations, the field definitions, the declaration of the mesh and of all the mesh features such as zones and sets
- `turbulenceModels` which contains several turbulence models
- `engine` declaration of classes for engine simulation
- `dynamicMesh` for moving meshes algorithm

OpenFOAM file organization

The generic application/class source code is structured as follows:



The `main` is included in the *.C file. Typically in OpenFOAM application several other files are included containing pieces of code or specific declarations

This file is included at the beginning of any piece of code using the class, including the class declaration code itself

Any piece of .C code can resource any number of classes and must begin with all the .H files required to declare these classes. The classes in turn can resource other classes and begin with the relevant .H files

OpenFOAM file organization

Header files are included in the code using `# include` statements:

```
# include "otherHeader.H";
```

It causes the compiler to suspend reading from the current file to read the file specified

Any self-contained piece of code can be put into a header file and included at the relevant location in the main code in order to improve code readability

For example, in most OpenFOAM applications the code for creating fields and reading field input data is included in a file createFields.H:

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "readThermodynamicProperties.H"
#   include "createFields.H"

    for (runTime++; !runTime.end(); runTime++)
```

Structure of an application

We shall consider the sonicFoam application as an example of application directory

This application can be found at:

```
$FOAM_SOLVERS/compressible/sonicFoam
```

The top level source file takes the application name with the .C extension. The source code for the sonicFoam application would reside is a directory sonicFoam and the top level file would be sonicFoam.C

```
sonicFoam
     |---->compressibleContinuityErrs.H
     |---->createFields.H
     |---->readThermodynamicProperties.H
     |---->readTransportProperties.H
     |---->sonicFoam.C
     |---->Make
               |---->files
               |---->options
```

The directory must also contain a Make subdirectory containing 2 files, options and files, which are needed by the `wmake` command

Structure of an application

Having a look at the sonicFoam application we will find out that several files are included which are not contained in the application folder:

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createMesh.H"
#   include "readThermodynamicProperties.H"
#   include "readTransportProperties.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

    for (runTime++; !runTime.end(); runTime++)
    {
#       include "readPISOControls.H"
#       include "compressibleCourantNo.H"
#       include "rhoEqn.H"
```

Where does OpenFOAM pick those files up?

Compiling the application: the Make folder

The compiler searches for the included header files in the following order, specified with the `-I` option in wmake:

1. the `$WM_PROJECT DIR/src/OpenFOAM/lnInclude` directory;
2. a local lnInclude directory, i.e. `sonicFoam/lnInclude`;
3. the local directory, i.e. `sonicFoam`;
4. platform dependent paths set in files in the `$WM_PROJECT_DIR/wmake/rules/$WM_ARCH/` directory, e.g. `/usr/X11/include` and `$(MPICH_ARCH_PATH)/include`;
5. other directories specified explicitly in the `Make/options` file with the `-I` option.

The Make/options file contains the full directory paths to locate header files using the syntax:

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude
```

The directory names are preceeded by the -I flag and the syntax uses the \ to continue the `EXE_INC` across several lines, with no \ after the final entry

Linking to libraries

The compiler links to shared object library files in the following directory paths, specified with the -L option in wmake:

1. the `$FOAM_LIBBIN` directory
2. platform dependent paths set in files in the `$WM_DIR/rules/$WM_ARCH/` directory, e.g. `/usr/X11/lib`
3. other directories specified in the Make/options file

The actual library files to be linked must be specified using the -l option and removing the lib prefix and .so extension from the library file name, e.g. `libfiniteVolume.so` is included with the flag `-lfiniteVolume`

Linking to libraries

By default, wmake loads the following libraries:

1. the libOpenFOAM.so library from the $FOAM_LIBBIN directory;
2. platform dependent libraries specified in set in files in the $WM_DIR/rules/$WM_ARCH/ directory, e.g. libm.so from /usr/X11/lib and liblam.so from $(LAM_ARCH_PATH)/lib;
3. other libraries specified in the Make/options file

The Make/options file contains the full directory paths and library names using the syntax:

```
EXE_LIBS = \
-L<libraryPath1> \
-l<library1>
```

The directory paths are preceeded by the -L flag, the library names are preceeded by the -l flag

Source files to be compiled

The compiler requires a list of .C source files that must be compiled. The list must contain the main .C file but also any other source files that are created for the specific application but are not included in a class library.

The list only includes the name of the main .C file, e.g. sonicFoam.C

The `Make/files` file includes a full path and name of the compiled executable, specified by the `EXE = syntax`.

OpenFOAM offers two choices for path:

1. standard release applications are stored in `$FOAM_APPBIN`
2. applications developed by the user are stored in `$FOAM_USER_APPBIN`

To develop your own applications, it is recommended to create an applications subdirectory in their `$WM_PROJECT_USER_DIR` directory containing the source code for personal OpenFOAM applications. The Make/files file for our example would appear as follows:

```
sonicFoam.C
EXE = $(FOAM_USER_APPBIN)/sonicFoam
```

Running wmake

The `wmake` script is executed by typing:

```
wmake <optionalArguments> <optionalDirectory>
```

The `<optionalDirectory>` is the directory path of the application that is being compiled

Typically, wmake is executed from within the directory of the application being compiled, in which case `<optionalDirectory>` can be omitted

To build an application executable no `<optionalArguments>` are required

`<optionalArguments>` may be specified for building libraries

Argument	Type of compilation
lib	Build a statically-linked library
libso	Build a dynamically-linked library
libo	Build a statically-linked object file library
jar	Build a JAVA archive
exe	Build an application independent of the specified project library

Removing dependency lists: wclean and rmdepall

On execution, `wmake` builds a dependency list file with a .dep file extension, e.g. soniFoam.dep in our example, and a list of files in a `Make/$WM_OPTIONS` directory

To remove these files the user can run the wclean script by typing:

```
wclean <optionalArguments> <optionalDirectory>
```

The `<optionalDirectory>` is a path to the directory of the application that is being compiled

Typically, `wclean` is executed from within the directory of the application, in which case the path can be omitted and it is needed when the dependencies are changed, for instance when an `# include "file.H"` is added or removed

To remove the dependency files and files from the Make directory no `<optionalArguments>` are required

If lib is specified in `<optionalArguments>` a local lnInclude directory will be deleted also

`rmdepall` removes all dependency .dep files recursively down the directory tree from the point at which it is executed. It is useful when updating OpenFOAM libraries

Compilation example: the sonicFoam application

The code begins with a brief description of the application contained within comments over 1 line (//) and multiple lines (/*...*/)

Following that, the code contains several `# include` statements, e.g. `# include "fvCFD.H"`, which causes the compiler to suspend reading from the current file, `sonicFoam.C` to read the `fvCFD.H`

This file is found at the path `$FOAM_SRC/finiteVolume/lnInclude/fvCFD.H` by means of the path specified in the `Make/option` file

sonicFoam contains only the `sonicFoam.C` source and the executable is written to the `$FOAM APPBIN` directory as all standard applications are

The `Make/files` therefore contains:

```
sonicFoam.C

EXE = $(FOAM_APPBIN)/sonicFoam
```

The user can compile sonicFoam by going to the `$FOAM_SOLVER/compressible/sonicFoam` directory and typing:

```
wmake
```

Compilation example: the sonicFoam application

Executing the `wmake` the output will be something like this

```
compressible/sonicFoam> wmake
Making dependency list for source file sonicFoam.C
SOURCE=sonicFoam.C ;
 g++ -m32 -Dlinux -DDP -Wall -Wno-strict-aliasing -Wextra -Wno-unused-
 -Wold-style-cast -O3  -DNoRepository -ftemplate-depth-40
 -I/home/gmonte/OpenFOAM/OpenFOAM-1.4.1/src/finiteVolume/lnInclude
 -IlnInclude -I.
 ...........
 -lm -o /home/gmonte/OpenFOAM/OpenFOAM-1.4.1/applications/bin/linuxGcc
```

Trying to recompile it will appear a message similar to the following to say that the executable is up to date and compiling is not necessary:

```
make: '/home/gmonte/OpenFOAM/OpenFOAM-1.4.1/applications/bin/
linuxGccDPOpt/sonicFoam' is up to date.
```

Running applications

Each application is designed to be executed from a terminal command line, typically reading and writing a set of data files associated with a particular case. The data files for a case are typically stored in a directory named after the case

For any application, the form of the command line entry can be found by simply entering the application name at the command line, for instance typing sonicFoam returns information

```
Usage: sonicFoam <root> <case> [-parallel]
```

The arguments in angled brackets, `< >`, i.e. `<root>` and `<case>`, are the compulsory arguments, while the arguments in square brackets, `[ ]`, are optional flags

Applications can be run as as a background process, for instance if the user wished to run the sonicFoam example as a background process and output the case progress to a log file (the log file will become useful later on), they could enter:

```
nohup nice -n 19 sonicFoam <root> <case> > log &
```

Equation representation

A central theme of the OpenFOAM design is that the solver applications, written using the OpenFOAM classes, have a syntax that closely resembles the partial differential equations being solved. For example the equation for the turbulence kinetic energy conservation

$$\frac{\partial k}{\partial t} + \nabla \cdot (Uk) - \nabla \cdot [(\nu + \nu_t)\nabla k] = \nu_t \left[\frac{1}{2}(\nabla U + \nabla U^T)\right]^2 - \frac{\epsilon_o}{k_o} k$$

Is written as follows:

```
solve
(
   fvm::ddt(k)
 + fvm::div(phi, k)
 - fvm::laplacian(nu() + nut, k)
== nut*magSqr(symm(fvc::grad(U)))
 - fvm::Sp(epsilon/k, k)
);
```

Correspondence between the implementation and the original equation is clear

The sonicFoam application

SonicFoam is a pressure-velocity coupled solver for compressible transient transonic/subsonic flows

Mass conservation:

```
solve(
        fvm::ddt(rho)
      + fvc::div(phi)
     );
```

Note that mass does not diffuse

Momentum equation

```
fvVectorMatrix UEqn
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
);

solve(UEqn == -fvc::grad(p));
```

The sonicFoam application

Energy equation

```
solve
(
    fvm::ddt(rho, e)
  + fvm::div(phi, e)
  - fvm::laplacian(mu, e)
 ==
  - p*fvc::div(phi/fvc::interpolate(rho))
  + mu*magSqr(symm(fvc::grad(U)))
);

T = e/Cv;
```

The PISO loop

```
for (int corr=0; corr<nCorr; corr++)
{
    volScalarField rUA = 1.0/UEqn.A();
    U = rUA*UEqn.H();
    surfaceScalarField phid =
    (
        (fvc::interpolate(rho*U) & mesh.Sf())
      + fvc::ddtPhiCorr(rUA, rho, U, phi)
    )/fvc::interpolate(p);
    for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix pEqn
        (
            fvm::ddt(psi, p)
          + fvm::div(phid, p, "div(phid,p)")
          - fvm::laplacian(rho*rUA, p)
        );
        pEqn.solve();
        phi = pEqn.flux();
    }
    U -= rUA*fvc::grad(p);
```

The rhoSonicFoam application

rhoSonicFoam is a density-based compressible flow solver:

```
solve
(
    fvm::ddt(rho)
  + fvm::div(phiv, rho)
);
solve
(
    fvm::ddt(rhoU)
  + fvm::div(phiv, rhoU)
 ==
  - fvc::grad(p)
);
solve
(
    fvm::ddt(rhoE)
  + fvm::div(phiv, rhoE)
 ==
  - fvc::div(phiv2, p)
);
```

A deeper look at the src folder: turbulence models

On the basis of what has been shown we shall try to create a new turbulence model and include it in the structure of the code

The implementation of the turbulence models is located in

```
$FOAM_SRC/turbulenceModels
```

It is preferable to create a separate folder when developing new application/libraries rather than modifying the existing ones in the `src` folder

the user should then copy a turbulence model into the user dedicate folder:

```
cd $WM_PROJECT_USER_DIR
cp -r $FOAM_SRC/turbulenceModels/incompressible/kEpsilon .
mv kEpsilon mykEpsilon
```

The user should also modify the `files` and `options` files in the Make directory

```
mykEpsilon.C
LIB = $(FOAM_USER_LIBBIN)/mylibTurbulenceModel
```

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/turbulenceModels/incompressible/lnInclude
```

A deeper look at the src folder: turbulence models

It is preferable to rename the source files according to the name of the turbulence model

```
mv kEpsilon.C mykEpsilon.C
mv kEpsilon.H mykEpsilon.H
```

The user must adapt also the names of the included files and of the class declaration

The customization may be reduced at adding output information inside the constructor:

```
    Info << "This is my new turbulence model" << endl;
```

The new library may be compiled by typing the following command

```
wmake libso
```

This command will generate the dynamic library

The user will find the *.so file inside the `$FOAM_USER_LIBBIN` folder

A deeper look at the src folder: turbulence models

To use the new library for turbulence models the user may include a new line in the `controlDict` file of the application to be used:

```
libs ("libmyTurbulenceModels.so");
```

This tells OpenFOAM which one is the library to be used for the new turbulence model

In this case the user do not need to recompile the application, namely `simpleFoam`

The customized turbulence model is selected in the `turbulenceModels`:

```
turbulenceModels    mykEpsilon;
```

The user will need to adjust accordingly the definition of the coefficients: `mykEpsilonCoeffs` instead of `kEpsilonCoeffs`

By running the application on the desired case the turbulence model is used

```
simpleFoam . pitzDaily
```

A deeper look at the src folder: turbulence models

There is actually another way to implement another turbulence model, but it is more intrusive that the shown one

The user can copy the kEspilon folder inside the `$FOAM_SRC/turbulenceModels/incompressible` folder

Make the same modifications to the files as in the previous case

To compile the new sources the used needs to modify the `Make/files` file, including the new path of the customized turbulence model:

```
RNGkEpsilon/RNGkEpsilon.C
kEpsilon/kEpsilon.C
mykEpsilon/mykEpsilon.C
     ...
```

The library will be generated by running the following command from inside the `$FOAM_SRC/turbulenceModels/incompressible` folder

```
wmake libso
```

The global controlDict file

OpenFOAM provides a system of messaging that is written during runtime, most of which are to help debugging problems encountered during running of a OpenFOAM case

This system is based onto switches which are listed in the `$WM_PROJECT_DIR/.OpenFOAM-1.4.1/controlDict` file

The list of possible switches is extensive and can be viewed by running the `foamDebugSwitches` application

There are some switches that control certain operational and optimization issues:

```
OptimisationSwitches
{
    fileModificationSkew 10;
            ....
    nProcsSimpleSum 0;
}
```

The `fileModificationSkew` keyword is the time in seconds that OpenFOAM will subtract from the file write time when assessing whether the file has been newly modified

When running over a NFS with some disparity in the clock settings on different machines, field data files appear to be modified ahead of time

The global controlDict file

To test the functionality of the debug switches the use may try to change the lduMatrix switch

```
DebugSwitches
{
    lduMatrix       1;
       ....
}
```

Setting this switch to 0: `lduMatrix    0` the runtime information about the solution of the matrix will be hidden

The user need to re-source the OpenFOAM cshrc file to keep the changes made to the controlDict file

Try to run the simpleFoam application (with the customized turbulence model) and see the difference in the output.

Keep in mind that the calculation is performed in the same way of when the switch is on

Update the documentation

When the user customizes a certain class or model it is possible to generate the documentation of the added parts

The documentation will be generated by running the `Allwmake` command in the `Doxygen` folder

The documentation is generated according to the `Make` folders

If the user has added new folders inside or outside the `src` directory, the new path must be added to the list of the input folders

To do this the user will need to access the `Doxyfile` inside the `doc` folder of the `$WM_PROJECT_DIR` directory and modify it

```
INPUT          = $(WM_PROJECT_DIR)/src/OpenFOAM \
                 $(WM_PROJECT_DIR)/src/Pstream \
                 $(WM_PROJECT_USER_DIR)/mykEpsilon
```

The test folder

The test folder contains several demonstration on how to use OpenFOAM classes

We shall have a look at the `mesh` usage demonstration

```
$FOAM_APP/test/mesh
```

It shows how to create the mesh and ow to use some member funcions of the mesh class

```
fvMesh mesh    // creates the mesh
(
    IOobject
    (
        fvMesh::defaultRegion,
        runTime.timeName(),
        runTime,
        IOobject::MUST_READ
    )
);

Info<< mesh.C() << endl;  // returns the centers of all the cells
Info<< mesh.V() << endl;  // returns the volume of all the cells
```

The test folder

First of all the user needs to compile the test application

From inside the application directory the user must run the `wmake` command

It will be created an executable called `meshTest` which should be run on a preexistent OpenFOAM case

We shall use the cavity tutorial for the icoFoam application:

```
/home/gmonte> tut
OpenFOAM-1.4.1/tutorials> cd icoFoam/
tutorials/icoFoam> blockMesh . cavity
tutorials/icoFoam> meshTest . cavity
```

The user will see a list of vector displayed on the shell

To keep track of the output it is possible to redirect the screen output onto a file

```
meshTest . cavity > meshTest.log
```