OpenFOAM user directory organization and compilation

OpenFOAM user directory organization and compilation

Prerequisites

- You know the OpenFOAM directory organization
- You have a basic understanding of the compilation process in OpenFOAM, and how it is related to the environment

Learning outcomes

- Set up and use the user directory in an organized way
- Basic copying, renaming and compilation of applications and libraries, as a user
- Compilation procedures, paths and linking, as a user

User directory organization

- The `$WM_PROJECT_USER_DIR` environment variable is set up as a suggested location of the user development and cases (note the similarity to the environment variable `$WM_PROJECT_DIR`, which is the location of the OpenFOAM source code). It is empty from scratch, but remember that we have created some directories to prepare:

```
$ ls $WM_PROJECT_USER_DIR
applications  run  src
```

- You recognize that `applications` and `src` are also found in `$WM_PROJECT_DIR`, and the purpose of creating those directories also in `$WM_PROJECT_USER_DIR` is to use the same directory structure for our developments as in the OpenFOAM source code. This is not mandatory, but it is good practice and you only have to remember one sub-directory structure.
- `applications` will be used for our own developed *solvers* and *utilities*
- `src` will be used for our own developed *libraries*
- `run` will be used for our cases, including running the original tutorials (assumed to be known)

Basic user compilation procedures, applications (1/5)

Here is the procedure of copying and compiling the `icoFoam` solver from its original location in `$WM_PROJECT_DIR` to the corresponding location in `$WM_PROJECT_USER_DIR`, renaming directory, file name and executable name (to `myIcoFoam`), and compiling:

Copy:

```
foam
cp -r --parents applications/solvers/incompressible/icoFoam $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
```

Note that you are now in a directory organized as in the original installation, and that we have exactly the same directories and files as in the original installation:

```
$ tree icoFoam
icoFoam
|-- createFields.H
|-- icoFoam.C
`-- Make
    |-- files
    `-- options
```

Basic user compilation procedures, applications (2/5)

The compilation is given instructions through the files named `files` and `options`. The file named `options` has *include* and *linking* instructions that can be kept as they are at this point.

The file named `files` contains:

```
icoFoam.C
EXE = $(FOAM_APPBIN)/icoFoam
```

This tells the compiler that it should compile the file named `icoFoam.C`, and that the executable should be saved with the name `icoFoam` in the directory `FOAM_APPBIN`.

However, we do not want to overwrite the original executable file, which has exactly that name and is located in exactly that location (since it was compiled with exactly the above files). We should therefore at a minimum change the name of the exectuable. It is *adviced* to also change the location of the executable to a corresponding directory for user-developed applications. As can be understood from the namings of the original directory and files, it is also *adviced* to change those. I.e.:

```
mv icoFoam myIcoFoam
mv myIcoFoam/icoFoam.C myIcoFoam/myIcoFoam.C
sed -i s/icoFoam/myIcoFoam/g myIcoFoam/Make/files
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/g myIcoFoam/Make/files
```

Basic user compilation procedures, applications (3/5)

At this point we can clean up (if any previous compilation left files), and compile:

```
wclean myIcoFoam
wmake myIcoFoam
```

Now you can try the `myIcoFoam` solver on the original `icoFoam/cavity` tutorial:

```
run
rm -r cavity
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity .
blockMesh -case cavity
myIcoFoam -case cavity >& log&
```

Check the top of the log-file, that you were running your `myIcoFoam` solver.

Basic user compilation procedures, applications (4/5)

Now, which files were modified/created when we typed `wmake`?

```
$ tree $WM_PROJECT_USER_DIR/applications/solvers/incompressible/myIcoFoam
$WM_PROJECT_USER_DIR/applications/solvers/incompressible/myIcoFoam
|-- createFields.H
|-- Make
|   |-- files
|   |-- linux64GccDPInt32Opt
|   |   |-- myIcoFoam.C.dep
|   |   |-- myIcoFoam.o
|   |   |-- options
|   |   |-- sourceFiles
|   |   `-- variables
|   `-- options
`-- myIcoFoam.C
```

I.e, there is now a directory `Make/linux64GccDPInt32Opt`, containing the same files as in

`$WM_PROJECT_DIR/build/linux64GccDPInt32Opt/applications/solvers/incompressible/icoFoam`

This means that the intermediate files are located differently for user compilations. However, we do not have to bother much about that.

Basic user compilation procedures, applications (5/5)

One additional file was modified/created when we typed `wmake`, and that is the executable itself. The file named `files` now says that it should be saved with the name `myIcoFoam` in the directory `FOAM_USER_APPBIN`:

```
$ ls -l $FOAM_USER_APPBIN/myIcoFoam
-rwxrwxr-x ... $WM_PROJECT_USER_DIR/platforms/linux64GccDPInt32Opt/bin/myIcoFoam
```

We see that the directory `$WM_PROJECT_USER_DIR/platforms` was created, and from `-rwxrwxr-x` that `myIcoFoam` is an executable file. That file is found when we type the name of that executable, since that path is included in the environment variable `$PATH` (here only important paths, as environment variables):

```
$ echo $PATH
...:
$FOAM_USER_APPBIN:
$FOAM_SITE_APPBIN:
$FOAM_APPBIN:
...
```

The paths are searched in order, until the executable is found the first time. It is only located in `$FOAM_USER_APPBIN`, so it is found there. However, if we wouldn't have changed the name of the executable there would have been one `icoFoam` in `$FOAM_APPBIN` and one in `$FOAM_USER_APPBIN`, and it would use the one in `$FOAM_USER_APPBIN` since that path is first in `$PATH`. I.e. we would override the original executable without touching it. BE CAREFUL!

Basic user compilation procedures, libraries (1/8)

Assume that we want to develop a new boundary condition, based on an existing one. The original boundary condition belongs to the `finiteVolume` library, so let's go there and have a look:

```
cd $WM_PROJECT_DIR/src/finiteVolume
```

We find the following directories (version dependent):

```
cfdTools   finiteVolume     fvMatrices   interpolation   Make          volMesh
fields     functionObjects  fvMesh       lnInclude       surfaceMesh
```

All of them (except `Make` and `lnInclude`) contain several layers of sub-directories, in which there are numerous classes that all belong to the `finiteVolume` library. The boundary conditions are located in `fields/fvPatchFields`, and we are going to copy the one in

```
fields/fvPatchFields/derived/cylindricalInletVelocity
```

An important note here is that the directory name (and the file names inside it) corresponds to the type name of the boundary condition (used to set the boundary condition in the cases). This makes it easy to use the `find` command to find the boundary condition in the directory structure: `find $FOAM_SRC -name cylindricalInletVelocity`.

Before we proceed we will discuss the `Make` and `lnInclude` directories...

Basic user compilation procedures, libraries (2/8)

The `Make` directory tells the compiler how to compile the library, including all its classes. The `Make/files` file lists all the files to be compiled, and where to put the final shared object file (here keeping only lines corresponding to the boundary condition of interest):

```
...
fvPatchFields = fields/fvPatchFields
...
derivedFvPatchFields = $(fvPatchFields)/derived
$(derivedFvPatchFields)/cylindricalInletVelocity/cylindricalInletVelocityFvPatchVectorField.C
...
LIB = $(FOAM_LIBBIN)/libfiniteVolume
```

The `Make/options` file tells the compiler where to find include files and which libraries to link to (version dependent):

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

LIB_LIBS = \
    -lOpenFOAM \
    -lmeshTools
```

There is in fact one more library that is found by default, as discussed in the next slide...

Basic user compilation procedures, libraries (3/8)

The `lnInclude` directory was generated when the source code was compiled. It contains soft links to all the files in the library. This makes the file `Make/options` much shorter, since for the include statements it is only necessary to give the path to the `lnInclude` directory of each library that contains any class that is included.

In the previous slide we see that the `finiteVolume` library uses classes from the libraries `surfMesh` and `OpenFOAM`. What we can't see is that it also uses classes from its own library, `finiteVolume`. However, since it is mostly the case that a library uses classes from its own library the `lnInclude` directory of the present library is searched by default.

When we copy one class of a library to another location we have to remember that it most likely uses other parts of the original library, and we thus have to add the original library to the `Make/options` file.

Basic user compilation procedures, libraries (4/8)

What we need to do is to copy the class that we want to develop, re-name directory, file names *and class name*, and set up a `Make` directory.

Copy and rename the directory and file names:

```
foam
cp -r --parents src/finiteVolume/fields/fvPatchFields/derived/cylindricalInletVelocity $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/src/finiteVolume/fields/fvPatchFields/derived
mv cylindricalInletVelocity myCylindricalInletVelocity
cd myCylindricalInletVelocity/
mv cylindricalInletVelocityFvPatchVectorField.C myCylindricalInletVelocityFvPatchVectorField.C
mv cylindricalInletVelocityFvPatchVectorField.H myCylindricalInletVelocityFvPatchVectorField.H
```

At this point we don't look into the files, but we need to change the class name wherever it occurs in the files by typing

```
sed -i s/cylindricalInletVelocity/myCylindricalInletVelocity/g *
```

We will get back to what this means later.

Basic user compilation procedures, libraries (5/8)

Set up the `Make` directory as discussed before.

```
cd $WM_PROJECT_USER_DIR/src/finiteVolume
```

The `Make/files` file should contain:

```
fvPatchFields = fields/fvPatchFields

derivedFvPatchFields = $(fvPatchFields)/derived
$(derivedFvPatchFields)/myCylindricalInletVelocity/myCylindricalInletVelocityFvPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/libmyFiniteVolume
```

Here we have added 'my' in three places, and changed to `FOAM_USER_LIBBIN`.
The `Make/options` file should contain:

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

LIB_LIBS = \
    -lOpenFOAM \
    -lmeshTools \
    -lfiniteVolume
```

Here we have added the `finiteVolume` library since we have moved out of that library.

Basic user compilation procedures, libraries (6/8)

Finally, compile using

```
cd $WM_PROJECT_USER_DIR/src/finiteVolume
wmake
```

The output in the terminal window should end with (here using environment variable)

```
-o $WM_PROJECT_USER_DIR/platforms/linux64GccDPInt32Opt/lib/libmyFiniteVolume.so
```

We see that it is located in a similar structure as in the original installation.

We also see that there are intermediate files in `Make/linux64GccDPInt32Opt/`

However, contrary to when compiling applications we can not run this file. It should instead be linked to the solver when running cases.

Basic user compilation procedures, libraries (7/8)

It is time to test the boundary condition, and we can read how to use it in the file named `myCylindricalInletVelocityFvPatchVectorField.H`:

```
type            myCylindricalInletVelocity;
axis            (0 0 1);
origin          (0 0 0);
axialVelocity   constant 30;
radialVelocity  constant -10;
rpm             constant 100;
```

Let's adapt this for the movingWall patch of U for a `cavity` case named `myCylindricalInletVelocityCavity`, as

```
type            myCylindricalInletVelocity;
axis            (1 0 0);
origin          (0 0 0);
axialVelocity   constant 1;
radialVelocity  constant 0;
rpm             constant 0;
value           uniform (0 0 0); //Compulsory, but only placeholder!!!
//Line to fix copy-paste-problem
```

I.e. we are actually only assigning the x-component of the velocity, as in the original case. The `value` entry at the end is compulsory to set, but it is overridden by the values determined from our boundary condition.

Basic user compilation procedures, libraries (8/8)

Before running, the new library must be added to the controlDict file of the case, as:

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

Then the solver knows that it should link to the new library. Now try to run the case!

For applications we discussed that the environment variable `$PATH` is used to find the executable files. The environment variable `LD_LIBRARY_PATH` is similarly used to find the libraries when dynamically linking:

```
$ echo $LD_LIBRARY_PATH
...:
$FOAM_USER_LIBBIN:
$FOAM_SITE_LIBBIN:
$FOAM_LIBBIN:
...
```

You can check which libraries the solver is linking to by:

```
ldd `which icoFoam`
```

However, since the solver itself does not know about the newly developed library, you can't see that one in the list.

As for the applications it is also for libraries important that you are using new library names. Otherwise you may be linking to another file than what you think. BE CAREFUL!!!

Compilation using wmake

Let's have a closer look at the terminal output when compiling (here `myIcoFoam`):

```
cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible/myIcoFoam
wclean
wmake
```

This yields in the terminal window (version dependent):

```
Making dependency list for source file myIcoFoam.C
g++ -std=c++11 -m64 -DOPENFOAM_PLUS=1706 -Dlinux64 -DWM_ARCH_OPTION=64 -DWM_DP -DWM_LABEL_SIZE=32 -Wall
-Wextra -Wold-style-cast -Wnon-virtual-dtor -Wno-unused-parameter -Wno-invalid-offsetof -O3
-DNoRepository -ftemplate-depth-100 -I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/finiteVolume/lnInclude
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/meshTools/lnInclude -IlnInclude -I.
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/OpenFOAM/lnInclude
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/OSspecific/POSIX/lnInclude   -fPIC -c myIcoFoam.C
-o Make/linux64GccDPInt32Opt/myIcoFoam.o

g++ -std=c++11 -m64 -DOPENFOAM_PLUS=1706 -Dlinux64 -DWM_ARCH_OPTION=64 -DWM_DP -DWM_LABEL_SIZE=32 -Wall
-Wextra -Wold-style-cast -Wnon-virtual-dtor -Wno-unused-parameter -Wno-invalid-offsetof -O3
-DNoRepository -ftemplate-depth-100 -I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/finiteVolume/lnInclude
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/meshTools/lnInclude -IlnInclude -I.
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/OpenFOAM/lnInclude
-I/home/oscfd/OpenFOAM/OpenFOAM-plus/src/OSspecific/POSIX/lnInclude   -fPIC
-Xlinker --add-needed -Xlinker --no-as-needed Make/linux64GccDPInt32Opt/myIcoFoam.o
-L/home/oscfd/OpenFOAM/OpenFOAM-plus/platforms/linux64GccDPInt32Opt/lib \
    -lfiniteVolume -lmeshTools -lOpenFOAM -ldl  \
     -lm -o /home/oscfd/OpenFOAM/oscfd-plus/platforms/linux64GccDPInt32Opt/bin/myIcoFoam
```

The compilation is done in two steps. The first gives the intermediate files. The second links to libraries. There are lots of flags, of which some are specified in `Make` and many in `wmake/rules`

Additional: Adding new BC to only myIcoFoam

Above we compiled the BC into a library that can be linked to by all solvers. Here we compile it into the solver.

1. Copy the `*.H` and `*.C` files from the BC into the `myIcoFoam` directory.
2. Add the `*.C` file name of the BC in `Make/files`, before `myIcoFoam.C`.
3. Compile, using `wmake`.
4. Run the case that uses the BC.

Note that if the `libs(*)` line is still present in the `controlDict` file of the case, the solver throws a warning that there is a duplicate entry in the runtime table. You should never have duplicate entries, since it is unclear which entry is being used. Just remove the `libs(*)` line to get rid of the problem.

Adding a class to a particular solver is only useful if the class only works for that particular solver. Most of the time, classes should be compiled into libraries.