Components of icoFoam (using Doxygen)

Components of icoFoam (using Doxygen)

We will now identify *the components* of `icoFoam` and figure out what they do.
We will also learn about optional use of the objects, by finding all their member functions.
For this **we will use Doxygen**.

This should be useful when you investigate and try to understand other solvers and utilities.

icoFoam: Files and directories

- The `icoFoam` directory (`$FOAM_SOLVERS/incompressible/icoFoam`) consists of the following:

```
createFields.H  Make/  icoFoam.C
```

- The `Make` directory contains instructions for the `wmake` compilation command.
- `icoFoam.C` is the main file, and `createFields.H` is an inclusion file, which is included in `icoFoam.C`.

icoFoam: header include-file fvCFD.H

- In the header of `icoFoam.C` we include `fvCFD.H`, which in turn includes all class declarations that are needed for `icoFoam` (see next slide).
- `fvCFD.H` is included from (see `Make/options`):
  `$FOAM_SRC/finiteVolume/lnInclude`, but that is actually only a link to
  `$FOAM_SRC/finiteVolume/cfdTools/general/include/fvCFD.H`.
- How to find `fvCFD.H` (and other files):
  - Use `find PATH -iname "*LETTERSINFILENAME*"` to find where in `PATH` a file with a file name containing `LETTERSINFILENAME` in its file name is located.
    In this case: `find $WM_PROJECT_DIR -iname "*fvCFD.H*"`
  - Use `locate fvCFD.H` to find all files with `fvCFD.H` in their names. Note that `locate` is much faster than `find`, but is not frequently updated when files are added and removed!
  - **Use Doxygen**.
    `https://www.openfoam.com/documentation/guides/latest/doc/`
    Use the search box in the upper-right corner to search for `fvCFD.H`
    Click on `fvCFD.H`
    See the path to the file in the lower left of the window.
    Click on `Go to the source code of this file.`

icoFoam: header include-file fvCFD.H (version dependent)

```
#ifndef fvCFD_H
#define fvCFD_H

#include "parRun.H"

#include "Time.H"
#include "fvMesh.H"
#include "fvc.H"
#include "fvMatrices.H"
#include "fvm.H"
#include "linear.H"
#include "uniformDimensionedFields.H"
#include "calculatedFvPatchFields.H"
#include "fixedValueFvPatchFields.H"
#include "adjustPhi.H"
#include "findRefCell.H"
#include "constants.H"
#include "OSspecific.H"
#include "argList.H"
#include "timeSelector.H"

#ifndef namespaceFoam
#define namespaceFoam
    using namespace Foam;
#endif

#endif
```

The inclusion files are all class declarations that are used in `icoFoam`. Dig further into the source file to find out what these classes actually do.

At the end we say that we will use all definitions made in `namespace Foam`.

icoFoam: header include-file pisoControl.H

- In the header of `icoFoam.C` we also include `pisoControl.H`, which is the declaration of the class `pisoControl`, which reads the `piso` subdictionary of the `fvSolution` dictionary and provides member functions for manipulating the piso algorithm (number of correctors etc.).
- Use `find $FOAM_SRC -name pisoControl.H` to get:

  ```
  $FOAM_SRC/finiteVolume/cfdTools/general/solutionControl/pisoControl/pisoControl.H
  $FOAM_SRC/finiteVolume/lnInclude/pisoControl.H
  ```

  where the second one is just a link to the first one.

- `picoControl.H` could also have been included through `fvCFD.H`, like the other class declarations. Why isn't it?

icoFoam: Main function and its include-files

- The `icoFoam` code is implemented in the `main` function:

  ```
  int main(int argc, char *argv[])
  ```

  where `int argc, char *argv[]` are the number of parameters, and the actual parameters used when running `icoFoam` (e.g. `-case cavity`).

- The case is initialized by (these include-files are just pieces of code!):

  ```
  #include "postProcess.H"
  #include "addCheckCaseOptions.H"
  #include "setRootCaseLists.H"
  #include "createTime.H"
  #include "createMesh.H"
  pisoControl piso(mesh);
  #include "createFields.H"
  #include "initContinuityErrs.H"
  ```

  where all inclusion files except `createFields.H` (why?) are included from `$FOAM_SRC/OpenFOAM/lnInclude` and `$FOAM_SRC/finiteVolume/lnInclude`.

- There are two more include-files in the main function:

  ```
  #include "CourantNo.H"
  #include "continuityErrs.H"
  ```

  which are both included from `$FOAM_SRC/finiteVolume/lnInclude`

  In the following slides we will have a look at the above lines/files.

icoFoam: postProcess.H

- The file `postProcess.H` contains code that only runs through the functionObjects if the flag `-postProcess` is supplied to `icoFoam`
  We will not have a close look

icoFoam: addCheckCaseOptions.H

- The file `addCheckCaseOptions.H` contains:

```
Foam::argList::addBoolOption
(
    "dry-run",
    "Check case set-up only using a single time step"
);
Foam::argList::addBoolOption
(
    "dry-run-write",
    "Check case set-up and write only using a single time step"
);
```

- This adds the possibility to use the `dry-run` and `dry-run-write` flags. These are available so that you can test that there are no silly mistakes in your case set-up. They fail and give the error message if there are some syntax errors that would make the simulation fail before starting.

icoFoam: setRootCaseLists.H

- The file `setRootCaseLists.H` contains:

```
// This is setRootCase, but with additional solver-related listing

#include "setRootCaseListOptions.H"
#include "setRootCase.H"
#include "setRootCaseListOutput.H"
```

Let's have a look at those...

icoFoam: setRootCaseListOptions.H and setRootCaseListOutput.H

- The file `setRootCaseListOptions.H` contains (e.g.):

```
argList::addBoolOption
 (
     "listScalarBCs",
     "List scalar field boundary conditions (fvPatchField<scalar>)",
     true // advanced
 );
```

- The file `setRootCaseListOutput.H` contains (e.g.):

```
if (args.found("listScalarBCs"))
     {
         Info<< "scalarBCs"
             << fvPatchField<Foam::scalar>::
                dictionaryConstructorTablePtr_->sortedToc()
             << endl;
         listOptions = true;
     }
```

- This adds the possibility to use more flags to `icoFoam`, that give additional information to the user. This is a step towards more user-friendliness. Note that the new flags are not shown by `icoFoam -help`, but by `icoFoam -help-full`.

icoFoam: setRootCase.H

- The file `setRootCase.H` contains:

```
Foam::argList args(argc, argv);
if (!args.checkRootCase())
{
    Foam::FatalError.exit();
}
```

The `args` object is constructed and used to call the `checkRootCase()`, which checks that there is a `controlDict` file.

icoFoam: createTime.H

- The file `createTime.H` contains:

```
Foam::Info<< "Create time\n" << Foam::endl;

Foam::Time runTime(Foam::Time::controlDictName, args);
```

  The main purpose is to construct the object `runTime`, belonging to the class `Foam::Time`.

- We see that the name of the `controlDict` is supplied when constructing the object, and it is in fact the `runTime` object that reads, remembers, checks if it updates, and uses the contents of the `controlDict`. At construction it only reads it.
- Use the `find` command to find `Time.H` and `Time.C` (and other files in the same directory), and after completing this course you should be able to investigate what is happening.
- In the next slide we will use `Doxygen` to figure out a bit about the `Time` class without knowing much about C++.

icoFoam: Use Doxygen to investigate the Time class

- Use `Doxygen` to figure out a bit about the `Time` class without knowing much about C++:
  - Open `https://www.openfoam.com/documentation/guides/latest/doc/`
  - Search for `Time` (in the upper-right corner) and click on `Time` as it appears (twice) in the search result.
  - Scroll down to `Public Member Functions` and find *some of* the functions that can be called at the top-level code for the `runTime` object.
  - Further scroll down to `Public Member Functions inherited from...` and click on the triangle to the left for each entry to make them visible (and searchable in this window). Start from bottom to make it easy for you when they expand.
  - Search for functions using `CTRL-f` and "`functionName<space>(`", e.g. "`loop (`".
  - Click on the function name to see more information.
- In `icoFoam.C` and `createFields` we see calls to:
  - `loop()`: Return true if run should continue and if so increment time. (acc. to `controlDict`)
  - `timeName()`: Return current time name.
  - `write()`: Write using setting from DB (e.g. `IOobject::AUTO_WRITE`, and settings in `controlDict`). (inherited from class `regIOobject`)
  - `printExecutionTime(Info)`: Print the elapsed ExecutionTime (cpu-time), ClockTime.
  - `constant()`: Return constant name. (inherited from class `TimePaths`)

icoFoam: createMesh.H

- The file `createMesh.H` *mainly* contains (if neither `dry-run` or `dry-run-write` are used):

```
Foam::autoPtr<Foam::fvMesh> meshPtr(nullptr);
...
    Foam::Info
        << "Create mesh for time = "
        << runTime.timeName() << Foam::nl << Foam::endl;
...
    meshPtr.reset
    (
        new Foam::fvMesh
        (
            Foam::IOobject
            (
                Foam::fvMesh::defaultRegion,
                runTime.timeName(),
                runTime,
                Foam::IOobject::MUST_READ
            )
        )
    );

Foam::fvMesh& mesh = meshPtr();
```

- Without going into details we just look at the last line and understand that an object named `mesh` is constructed as a reference of the class `fvMesh`, and it is referring to a pointer that points at a memory location where the mesh is stored.

icoFoam: Doxygen to investigate the fvMesh class

- Use the same Doxygen procedure for `fvMesh` as when we investigated the `Time` class.
- In `icoFoam.C` we see a call to:
  - `solver(...)`: Return the solver controls dictionary for the given field. (inherited from `solution`)
  - If you like to check what the function returns you can manipulate your `myIcoFoam` solver by adding before `return 0;`:

```
Info << mesh.solver(p.select(piso.finalInnerIter())) << endl;
```

- In `createFields` we see calls to:
  - `solutionDict()`: Return the selected sub-dictionary of solvers if the "select" keyword is given, otherwise return the complete dictionary (inherited from `solution`)
  - `setFluxRequired(...)`: Sets that the flux is required for the variable that is named as argument (inherited from `fvSchemes`)

icoFoam: pisoControl piso(mesh)

- The line saying `pisoControl piso(mesh);` constructs the `piso` object that belongs to class `pisoControl`, using the `mesh` object as argument. The constructor reads the `PISO` sub-dictionary in `fvSolution`.
- In `icoFoam.C` we see calls to (use Doxygen):
  - `momentumPredictor()`: Flag to indicate to solve for momentum. (from `solutionControl`)
  - `correct()`: Pressure corrector loop control. (from `pimpleControl`)
  - `correctNonOrthogonal()`: Non-orthogonal corrector loop. (from `solutionControl`)
  - `finalInnerIter()`: Return true if in the final inner (PISO) iteration. (overrides the same function inherited from `pimpleControl`)
  - `finalNonOrthogonalIter()`: Helper function to identify final non-orthogonal iteration. (from `solutionControl`)

icoFoam: createFields.H

- `createFields.H` is located in the `icoFoam` directory. It initializes all the variables used *specifically* in `icoFoam` (other solvers may need other variables). In particular:
  - `dimensionedScalar nu`
    Doxygen: Search `dimensionedScalar`, click `dimensionedScalar`, click on line number where it is defined (e.g. `Definition at line 41 of file dimensionedScalarFwd.H`)
    A `dimensionedScalar` is `typedef dimensioned<scalar>`.
    Investigate further in templated class `dimensioned<Type>`:
    Search `dimensioned`, click `dimensioned`, click `dimensioned<Type>`
  - `volScalarField p`
    Doxygen: Search `volScalarField`, click `volScalarField`, click on line number where it is defined (e.g. `Definition at line 54 of file volFieldsFwd.H`)
    It is `typedef GeometricField<scalar, fvPatchField, volMesh>`
    Search `GeometricField`, click `GeometricField<scalar, fvPatchField, volMesh>`
    In `createFields.h`: `p.name()`: Return name. (from class `IOobject`)
  - `volVectorField U`
    Doxygen as for `volScalarField`, but for `vector`
    In `icoFoam.C`: `U.correctBoundaryConditions()`: Correct boundary field. Used when field values are explicitly set without taking note of the boundary condition.
  - Continued...

icoFoam: createFields.H

- - ... continued
  - `surfaceScalarField phi`
    Doxygen: Search `surfaceScalarField`, click `surfaceScalarField`, click on line number where it is defined (e.g. `Definition at line 52 of file surfaceFieldsFwd.H`)
    It is `typedef GeometricField<scalar, fvsPatchField, surfaceMesh>`
    Search `GeometricField`, click `GeometricField<scalar, fvsPatchField, surfaceMesh>`

- `createFields.H` ends by setting the pressure in a reference cell by:
  `setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);`
  Doxygen: Search for `setRefCell`, click on `setRefCell`, click on the option with the same argument types as above. Yields description:

  ```
  If the field needs referencing find the reference cell nearest(in index)
  to the given cell looked-up for field, but which is not on a cyclic,
  symmetry or processor patch and return true, otherwise return false.
  ```

  Perhaps not as descriptive as we hoped... Let's move on!

icoFoam: initContinuityErrs.H

- The file `initContinuityErrs.H` contains:

```
#ifndef initContinuityErrs_H
#define initContinuityErrs_H

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

uniformDimensionedScalarField cumulativeContErrIO
(
    IOobject
    (
        "cumulativeContErr",
        runTime.timeName(),
        "uniform",
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    dimensionedScalar(dimless, Zero)
);
scalar& cumulativeContErr = cumulativeContErrIO.value();
```

The first two lines makes sure that the third line is not done more than once, even if the file happens to be included several times in the top-level code.
The last line constructs and initializes a scalar for the cumulative continuity error.

icoFoam: CourantNo.H

- The file `CourantNo.H` contains:

```
scalar CoNum = 0.0;
scalar meanCoNum = 0.0;

{
    scalarField sumPhi
    (
        fvc::surfaceSum(mag(phi))().primitiveField()
    );

    CoNum = 0.5*gMax(sumPhi/mesh.V().field())*runTime.deltaTValue();

    meanCoNum =
        0.5*(gSum(sumPhi)/gSum(mesh.V().field()))*runTime.deltaTValue();
}

Info<< "Courant Number mean: " << meanCoNum
    << " max: " << CoNum << endl;
```

This simply calculates the mean and max Courant numbers and report them to the terminal window.

icoFoam: continuityErrs.H

- The file `continuityErrs.H` contains:

```
{
    volScalarField contErr(fvc::div(phi));

    scalar sumLocalContErr = runTime.deltaTValue()*
        mag(contErr)().weightedAverage(mesh.V()).value();

    scalar globalContErr = runTime.deltaTValue()*
        contErr.weightedAverage(mesh.V()).value();
    cumulativeContErr += globalContErr;

    Info<< "time step continuity errors : sum local = " << sumLocalContErr
        << ", global = " << globalContErr
        << ", cumulative = " << cumulativeContErr
        << endl;
}
```

This calculates and prints the continuity error, as we see it in the terminal window.

icoFoam: Objects, namespaces and functions

We have already before discussed the PISO algorithm in `icoFoam`.

We will now investigate some of the more important objects and functions used in PISO.

- `fvVectorMatrix UEqn` (and `fvScalarMatrix pEqn`)
  Doxygen: Search `fvVectorMatrix`, click `fvVectorMatrix`
  It is `typedef fvMatrix<vector>`
  Search `fvMatrix`, click `fvMatrix`, click `fvMatrix<Type>`
  `UEqn.A()`: Return the central coefficient.
  `UEqn.H()`: Return the H operation source.
  `pEqn.setReference(...)`: Set reference level for solution.
  `pEqn.solve(...)`: Solve returning the solution statistics.
  `pEqn.flux()`: Return the face-flux field from the matrix.

Continued ...

icoFoam: Objects, namespaces and functions

... continued

- `fvm::ddt(U)` (and all other `fvm::`)
  Search `fvm`, click `fvm`.
  Namespace of functions to calculate implicit derivatives returning a matrix.
  Temporal derivatives are calculated using Euler-implicit, backward differencing or Crank-Nicolson. Spatial derivatives are calculated using Gauss' Theorem.

```
The following have no written descriptions, but you can click on them and investigate the source code.
ddt (const GeometricField< Type, fvPatchField, volMesh > &vf)
div (const surfaceScalarField &flux, const GeometricField< Type, fvPatchField, volMesh > &vf)
laplacian (const dimensioned< GType > &gamma, const GeometricField< Type, fvPatchField, volMesh > &vf)
laplacian (const GeometricField< GType, fvPatchField, volMesh > &gamma, const GeometricField< Type, fvPatchFie
```

- `fvc::grad(p)` (and all other `fvc::`)
  Search `fvc`, click `fvc`.
  Namespace of functions to calculate explicit derivatives.

```
The following have no written descriptions, but you can click on them and investigate the source code.
grad (const GeometricField< Type, fvPatchField, volMesh > &vf)
ddtCorr (const GeometricField< Type, fvPatchField, volMesh > &U, const GeometricField< Type, fvsPatchField, su
```

Continued ...

icoFoam: Objects, namespaces and functions

... continued

- My own descriptions of the functions used in `icoFOAM.C`, related to `phiHbyA` (by searching in Doxygen):
  - `constrainHbyA(rAU*UEqn.H(), U, p))`: Make sure that the `rAU*UEqn.H()` field gets correct boundary conditions (for `HbyA`).
  - `fvc::flux(HbyA)`: Interpolated face-flux field from `HbyA` (interp. scheme in `fvSchemes`).
  - `fvc::interpolate(rAU)`: Interpolate `rAU` to faces (interp. scheme in `fvSchemes`).
  - `fvc::ddtCorr(U, phi)`: Correct `rAU` interpolation according to time scheme that is used (needs to be checked in more detail).
  - `adjustPhi(phiHbyA, U, p)`: Calculates difference of flux (according to `phiHbyA`) in and out and corrects flux out (of `phiHbyA`) to get global continuity.
  - I.e. `phiHbyA` is the face flux field of `UEqn` (which is without the pressure gradient).
  - `constrainPressure(p, U, phiHbyA, rAU)`: Make sure that the pressure gradient at boundaries are consistent with the flux (by `phiHbyA`) at those boundaries.