How to implement a new boundary condition

- The implementations for the boundary conditions are located in `src/finiteVolume/fields/fvPatchFields/`
- To add a new boundary condition, start by finding one that does almost what you want. Copy that to your own implementation of a solver to link it statically to that solver. Compile it with your solver without any modifications before you start modifying it.
- We will now try adding parabolicVelocityFvPatchVectorField to the simpleFoam solver from OpenFOAM-1.4.1-dev to OpenFOAM-1.4.1, and use it for the pitzDaily tutorial (see next slide)

Add parabolicVelocityFvPatchVectorField locally and statically

```
run
cp -r $WM_PROJECT_DIR/tutorials/pitzDaily .
cp -r $WM_PROJECT_DIR/applications/solvers/incompressible/ \
       simpleFoam pitzDaily
cd pitzDaily
cp -r /chalmers/sw/unsup/OpenFOAM/OpenFOAM-1.4.1-dev/src/finiteVolume/ \
       fields/fvPatchFields/derived/parabolicVelocity simpleFoam
mv simpleFoam parabolicInletSimpleFoam
cd parabolicInletSimpleFoam
```

- Add `parabolicVelocityFvPatchVectorField.C` to the second line of `Make/files`, and modify the final line to `EXE = $(FOAM_USER_APPBIN)/parabolicInletSimpleFoam`
- Add `#include "parabolicVelocityFvPatchVectorField.H"` in the header of your `simpleFoam.C` file, so that the solver knows the new boundary condition.
- Compile with `wmake`, update your hash table using `rehash` (if needed)
- The next step is to modify the case so that it uses the new boundary condition.

Use the parabolicVelocityFvPatchVectorField

- Run `blockMesh` on your pitzDaily case.
- Modify the entry for the `inlet` boundary condition in `0/U` to:

```
type                parabolicVelocity;
n                   (1 0 0);
y                   (0 1 0);
maxValue            1;
value               (0 0 0); // Dummy for paraFoam
```

  The contents of this entry must be in accordance with the constructor in the `parabolicVelocityFvPatchVectorField` class. `n` is the direction of the flow, `y` is the coordinate direction of the profile, and maxvalue is the centerline velocity.

- Run the case using `parabolicInletSimpleFoam`. (You might see that there is a bug - try to find it).
- You now have a boundary condition and a case that only work with this solver, and however you modify it you will not destroy anything else in OpenFOAM. When you are sure that it works as it should you can add it globally so that it can be used for any solver in OpenFOAM. We will do that now.

Compile your boundary condition as a new dynamic library

- Again, copy the boundary condition to somewhere outside the OpenFOAM installation:

```
cp -r /chalmers/sw/unsup/OpenFOAM/OpenFOAM-1.4.1-dev/src/finiteVolume/ \
       fields/fvPatchFields/derived/parabolicVelocity .
cd parabolicVelocity
```

- We need a Make/files file:

```
parabolicVelocityFvPatchVectorField.C
LIB = $(FOAM_USER_LIBBIN)/libmyBCs
```

- We need a Make/options file:

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

- Compile the dynamic library:

```
wmake libso
```

Use your boundary condition from the dynamic library

- The boundary condition will not be recognized by any of the original OpenFOAM solvers unless we tell OpenFOAM that the library exists. In OpenFOAM-1.4.1 this is done by adding a line in the `system/controlDict` file:

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

  i.e. the library must be added for each case that will use it, but no re-compilation is needed for any solver. `libmyBCs.so` is found using the `LD_LIBRARY_PATH` environment variable, and if you followed the instructions on how to set up OpenFOAM and compile the boundary condition this should work automatically.

- You can now set up the case as we did earlier and run it using the original `simpleFoam` solver. Note that we never re-compiled the original `simpleFoam` solver, and if you do `ldd ‘which simpleFoam‘` your new library will NOT show up since it is linked at run-time (using dlopen).

A look at the boundary condition

- The `parabolicVelocityFvPatchVectorField` boundary condition consists of two files:

```
parabolicVelocityFvPatchVectorField.C
parabolicVelocityFvPatchVectorField.H
```

- The `.H`-file is the header file, and it is included in the header of the `.C`-file.
- We can see (`.H`) that we create a sub class to the `fixedValueFvPatchVectorField`:

```
class parabolicVelocityFvPatchVectorField:
public fixedValueFvPatchVectorField
```

  i.e. this is for Dirichlet (fixed) boundary conditions for vector fields.

- The class has the private data

```
        //- Peak velocity magnitude
        scalar maxValue_;
        //- Flow direction
        vector n_;
        //- Direction of the y-coordinate
        vector y_;
```

A look at the boundary condition

- The `TypeName("parabolicVelocity")`, used when specifying the boundary condition, is defined.
- There are some public constructors and member functions that are defined in detail in the `.C`-file.
- We used the third constructor when we tested the boundary condition, i.e. we read the member data from a dictionary.
- The actual implementation of the boundary condition can be found in the `updateCoeffs()` member function:

```
boundBox bb(patch().patch().localPoints(), false);
vector ctr = 0.5*(bb.max() + bb.min());
const vectorField& c = patch().Cf();
scalarField coord = ((c - ctr) & y_)/((bb.max() - bb.min())/2 & y_);
vectorField::operator=(n_*maxValue_*(1.0 - sqr(coord)));
```

Here I have made a correction to the small error (the division by 2 on line 4). The boundary condition thus implements the parabolic function, which might have been obvious by its name. You can easily modify the last line to another function.

A look at the boundary condition

- The member function `write` defines how to write out the boundary values in the time directory. The final line, `writeEntry("value", os);` writes out all the values, which is only needed for post-processing.
- Find out more about all the variables by including the following in the end of the `updateCoeffs` member function:

```
Info << "c" << c << endl;
Info << "ctr" << ctr << endl;
Info << "y_" << y_ << endl;
Info << "bb.max()" << bb.max() << endl;
Info << "bb.min()" << bb.min() << endl;
Info << "(c - ctr)" << c - ctr << endl;
Info << "((c - ctr) & y_)" << ((c - ctr) & y_) << endl;
Info << "((bb.max() - bb.min()) & y_)" <<
        ((bb.max() - bb.min()) & y_) << endl;
Info << "coord" << coord << endl;
```