No C++, please. We’re users!

Case setup and quantitative evaluation using swak4Foam (and a bit of PyFoam)

Bernhard F.W. Gschaider

Penn State
13. June 2011
Outline

1 Starting up
   What it’s about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in ‘special zones’
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

No C++, please. We’re users!
Outline

1. **Starting up**
   - What it’s about
   - Technicalities

2. **A crash course in PyFoam**
   - Description
   - Usage
   - Advertisement

3. **Building a new dam - Case setup**
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. **First numbers - Simple evaluations**
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. **Looking at other entities**
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. **Dynamic boundaries**
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. **Advanced topics**
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. **Going out**
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
Outline

Starting up
- What it's about
- Technicalities

A crash course in PyFoam
- Description
- Usage
- Advertisement

Building a new dam - Case setup
- The original case
- Setting initial conditions
- Variables
- The Conditional expression
- Static boundary conditions

First numbers - Simple evaluations
- A new field
- simpleFunctionObjects
- Complicated Expression
- Working with subsets

Looking at other entities
- Sum of liquid in 'special zones'
- Calculations after the fact

Dynamic boundaries
- Introducing groovyBC
- Finding errors in boundary conditions

Advanced topics
- Storing values
- Calculations on sampledSets
- Switching according to the simulation state
- On the surface
- Manipulating the solution

Going out
- What wasn’t shown
- Merging cases with Mercurial
- Ignaz says goodbye
Aim of this lecture

- Introduction to swak4Foam
- Give a crash-course in the use of the pyFoam-utilities
  - We will see some of the lesser known utilities in action
- After this lecture you should know
  - what swak4Foam can do
  - how to use it yourself
About this presentation

- This presentation is accompanied by a tar with the necessary case files to reproduce the steps described here
  - More about that later
- The slides with the word Movie in the title have an embedded movie
  - This is not directly playable with all PDF-viewers
  - ... for instance with Okular on the stick you’ll have to save it and play it separately
- If you see `<brk>` `<cnt>` in the code examples then this is just a line that was too long for the slide
Meeting Ignaz . . . again

- Ignaz is a CFD-engineer
  - A specialist for the damBreak-case
- We already met him in two presentations on PyFoam
  - Happy foaming with Python (Montreal 2009)
  - Automatisation with PyFoam (Gothenburg 2010)
- We meet during a severe crisis in his life
Ignaz problem

- For a damBreak-project Ignaz needed complex boundary conditions and procedures to evaluate the results
- Ignaz decided to code these in C++
- After encountering a 42-line compiler error message Ignaz suffered a nervous breakdown
- After recovering his doctor forbid him the use of C++
  - ... and to limit his use of Python
- Ignaz is devastated because he does not know how to complete his job
  - He goes on a binge of watching MacGuyver reruns
- ... and then the solution occurs to him:
What is swak4Foam

From http://openfoamwiki.net/index.php/Contrib/swak4Foam

swak4Foam stands for SWiss Army Knife for Foam. Like that knife it rarely is the best tool for any given task, but sometimes it is more convenient to get it out of your pocket than going to the tool-shed to get the chain-saw.

It is the result of the merge of

- funkySetFields
- groovyBC

and has become something mightier and scarier. Something that even the mighty Chutulu does not dare to anger without good reason... but I digress
Outline

1 Starting up
   What it’s about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye
Setting the OpenFOAM-version

- Currently Ignaz uses OpenFOAM 1.7
  - No particular reason. He just likes odd numbers
- Make sure that you’re using it as well
  - If you’re not sure whether your shell is set to it. Just enter
    `startOF171`
on the shell
  - or just click the “OpenFOAM 1.7.1”-button on the desktop
    - You get a terminal with the a zsh that is pre-configured to use OpenFOAM 1.7.1
Advertising: *zsh*

- Ignaz likes the *zsh*
- With a good configuration file (*.zshrc*) it kicks the *bash*’es...
  - ... whatever
    - It is more colorful
    - It has a nice tab-completion
    - It knows about SVN, git or mercurial
Preparing our environment

- Make sure you use OF 1.7
- Go to the directory you plan to work in

    cd $HOME

- Get the contents of the archive with the training material

    tar xzf /cdrom/OFW6/Training/gschaider_material.tgz

- Enter the directory

    cd swakAndPyFoam

- The shell prompt should tell you
  - That you’re in a git-archive
  - That you’re on the default-branch
Going to specific points with git

- We’re going to cover a lot of ground during this lecture
- If you get lost: don’t despair
  - Throughout the presentations there will be commands like

  ```
git checkout 0000_OriginalTutorial -b original
  
  This means:
  - **Got to the tag 0000_OriginalTutorial** (on the zsh entering 000 then pressing TAB will complete the tag name for you. Can you understand why Ignaz likes the zsh?)
  - **Open a new branch original** (that name you can choose for yourself)

- If you did changes that you want to keep for a later review (on the flight home) enter something like (the text of the message is yours)

  ```
git commit -a -m "This works. Cool. But we’re moving to the next stage"
  
  and later you can recall it with

  ```
git checkout original
  ```
Outline

1 Starting up
   What it's about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye
Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye
What is PyFoam

• PyFoam is a library for
  • Manipulating OpenFOAM-cases
  • Controlling OpenFOAM-runs

• It is written in Python
• Based upon that library there is a number of utilities
  • For case manipulation
  • Running simulations
  • Looking at the results

• All utilities start with pyFoam (so TAB-completion gives you an overview)
  • Each utility has an online help that is shown when using the -help-option
  • Additional information can be found
    • on openfoamwiki.net
    • in the two presentations mentioned above
Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider

No C++, please. We’re users!
Case setup

- Cloning an existing case
  
  ```python
  pyFoamCloneCase.py $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily test
  ```

- Decomposing the case
  
  ```bash
  blockMesh -case test
  pyFoamDecompose.py test 2
  ```

- Getting info about the case
  
  ```python
  pyFoamCaseReport.py test --short-bc --decomposition | rst2pdf >test.pdf
  ```

- Clearing non-essential data
  
  ```python
  pyFoamClearCase.py test --processors
  ```

- Pack the case into an archive (including the last time-step)
  
  ```python
  pyFoamPackCase.py test --last
  ```

- List all the OpenFOAM-cases in a directory (with additional information)
  
  ```python
  pyFoamListCases.py .
  ```
Running

- Straight running of a solver
  
  `pyFoamRunner.py interFoam`

- Clear the case beforehand and only show the time
  
  `pyFoamRunner.py --clear --progress interFoam`

- Show plots while simulating
  
  `pyFoamPlotRunner.py --clear --progress interFoam`

- Change `controlDict` to write all time-steps (afterwards change it back)
  
  `pyFoamRunner.py --write-all interFoam`

- Run a different OpenFOAM-Version than the default-one
  
  `pyFoamRunner.py --foam=1.9-beta interFoam`

- Run the debug-version of the current version
  
  `pyFoamRunner.py --current --force-debug interFoam`
Generated files

- Typically PyFoam generates several files during a run (the names of some of those depend on the case-name)

  - **case.foam**  Stub-file to open the case in ParaView
  - **PyFoamRunner.solver.logfile**  File with all the output that usually goes to the standard-output
  - **PyFoamRunner.solver..analyzed**  Directory with the results of the output analysis
  - **pickledPlots**  A special file that stores all the results of the analysis
  - **PyFoamHistory**  Log with all the PyFoam commands used on that case
Plotting

- Any logfile can be analyzed and plotted
  
  `pyFoamPlotWatcher.py --progress someOldLogfile`

- A number of things can be plotted
  - Residuals
  - Continuity error
  - Courant number
  - Time-step

- User-defined plots can be specified
  - Specified in a file `customRegexp`
  - Data is analyzed using regular expressions
  - We will see examples for this later

- The option `-hardcopy` generates pictures of the plots
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
What else can PyFoam do for me?

- Write and read dictionaries from the command line
- Display the `blockMeshDict`
- Generate plots of Surfaces and sample lines
- Interact with `paraView`
- Control OpenFOAM-runs over the net
Outline

1 Starting up
   What it’s about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

No C++, please. We’re users!
Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye
Description of the damBreak case

- Ignaz is a specialist on the damBreak case
  - We already said that
- The case can be found in the FOAM_TUTORIALS
- This is how the case looks like

```
git checkout 0000_OriginalTutorial -b original
```
Movie: damBreak original
Clearing out unused stuff

- Ignaz prepares the case for his purposes:
  - Remove the unneeded `setFieldsDict`
  - Make sure that all the files touched by a utility end with `.gz`

`git checkout 0010_StartingPoint -b step1`
Starting up
A crash course in PyFoam
Building a new dam - Case setup
First numbers - Simple evaluations
Looking at other entities
Dynamic boundaries
Advanced topics
Going out

The original case
Setting initial conditions
Variables
The Conditional expression
Static boundary conditions

Outline

1 Starting up
What it's about
Technicalities

2 A crash course in PyFoam
Description
Usage
Advertisement

3 Building a new dam - Case setup
The original case
Setting initial conditions
Variables
The Conditional expression
Static boundary conditions

4 First numbers - Simple evaluations
A new field
simpleFunctionObjects
Complicated Expression
Working with subsets

5 Looking at other entities
Sum of liquid in 'special zones'
Calculations after the fact

6 Dynamic boundaries
Introducing groovyBC
Finding errors in boundary conditions

7 Advanced topics
Storing values
Calculations on sampledSets
Switching according to the simulation state
On the surface
Manipulating the solution

8 Going out
What wasn’t shown
Merging cases with Mercurial
Ignaz says goodbye
Ignaz wants to play ball

- Ignaz is asked to apply his knowledge in the sports industry:
  - Investigate the possibility to use liquid spheres for games
  - He reuses the damBreak-case but the initial conditions should be different

```
git checkout 0020_throwABall -b step2
```
Preparing the case from now on

In the following sections the case will always be prepared in this way:

- Create the mesh
  blockMesh
- Run script with some additional commands
  ./prepareCase.sh
- Run the solver
  pyFoamRunner.py --clear --progress interFoam
The preparation script

```bash
#!/bin/sh

pyFoamClearCase.py .

rm -f 0/*.gz

cp 0/alpha1.org 0/alpha1
cp 0/U.org 0/U

funkySetFields -time 0
```
What it does

1. Clears out old time-steps (if present)
2. Removes files touched by utilities from the initial time-step
3. Copy the template files
4. And last but not least: call `funkySetFields`
**funkySetFields - basic usage**

- FSF always needs the time for which it should be used to be specified on the command line.
- For the further information two modes are possible:
  1. **expression** and all the other parameters are specified on the command line.
     - This is nice for quickly modifying fields and trying out stuff.
     - Only one expression can be entered at a time.
  2. Additional parameters are specified in a dictionary file.
     - One entry `expressions` with a list of dictionaries is needed.
     - Basic parameters have the same names as the command line options.
The `funkySetFieldsDict` expressions

```plaintext
expressions
{
  initZero
  {
    field alpha1;
    create false;
    expression "0";
    keepPatches true;
  }
  setCircle
  {
    field alpha1;
    expression "1";
    keepPatches true;
    condition "mag(pos() - vector(0.15,0.3,0.007))<0.12";
  }
  setVel
  {
    field U;
    keepPatches true;
    create false;
    expression "alpha1*vector(2,0,0)";
  }
}
```
Movie: Splashing ball
Expressions in *funkySetFields* - general structure

- The basic format of the expressions is the format of C++-expressions
  - The usual operators +-* (also logical operations and && and ||)
  - The if-them-else operators ?:
  - & and ^ have the same meaning they have in OpenFOAM: inner- and cross-vector-product
- All functions which are implemented for fields in OpenFOAM (\sin, \cos etc) can be used
- Components of a vector can be extracted with .x, .y and .z
- There are additional functions. The most important ones are:
  - pos the position of the cell center
  - vol The volume of the cell
- Numbers create a constant field
  - Vector fields are created with the vector “function”
- If an unknown symbol fits an existing field then this field is used (with the correct type)
Starting up
A crash course in PyFoam
Building a new dam - Case setup
First numbers - Simple evaluations
Looking at other entities
Dynamic boundaries
Advanced topics
Going out

The original case
Setting initial conditions
Variables
The Conditional expression
Static boundary conditions

Outline

1 Starting up
   What it's about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We're users!
Making it more maintainable

- Ignaz is asked to investigate a number of different geometries
- But the proportion of the ball to the geometry should always be the same
- Also his intern thinks that the hardcoded center vector(0.15, 0.3, 0.007) is ugly
  - Ignaz would never argue with an intern

```
git checkout 0030_throwABallVariables -b step3
```
Change in the dictionary

```plaintext
setCircle
{
    field alpha1;
    keepPatches true;
    expression "mag(pos()-center)<radius";
    variables (
        "width=max(pos().x)-min(pos().x);"
        "height=max(pos().y)-min(pos().y);"
        "middle=(max(pos().z)+min(pos().z))/2;"
        "center=vector(min(pos().x)+width/4,min(pos().y)+height"<brk>
            "<cont>*3/5,middle);"
        "radius=width/5;"
    );
}
```
Things you should know about variables

- Variables are defined in a surprisingly appropriately named list called `variables`.
  - Used to be a single unreadably long string.

- Syntax is:
  1. Variable name
  2. `=`
  3. Expression (can contain previously defined variables)
  4. A closing `;`

- A variable takes precedence over a field of the same name (be careful!)

- Variables are only available in FSF if being used in dictionary-mode.

No C++, please. We’re users!
Movie: Slightly different splash
Alternative to the old setFields-expression

- Ignaz is asked to investigate a slightly more boring type of wave
  - Can reuse the existing case
- Because he doesn’t want to overwrite the amazing ball-condition in funkySetFieldsDict he uses an alternate dictionary funkySetFieldsDict.alternate:
  
  funkySetFields -time 0 -dictExt alternate

- This case can be found in
  
  git checkout 0040_boringAlternative -b boring
The expression

```python
setLower
{
    field alpha1;
    keepPatches true;
    expression "((pos().y<height/2) ? 1 : (pos().y<height*3/5 && pos().x>width/2) ? 1 : 0)"
    variables ("width=max(pos().x)-min(pos().x);" "height=max(pos().y)-min(pos().y);"
    }
```
The conditional expression

- Common on the C/C++/Java-family of programming languages
- Is basically a if-then-else in one line
  - \( a \ ? \ b \ : \ c \) means if \( a \) is true then use \( b \) else use \( c \)
  - \( a \) has to be a logical expression
  - \( b \) and \( c \) have to evaluate to the same type (scalar or vector)
- Careful: \( b \) and \( c \) are always completely evaluated
  - So something like \( x==0 \ ? \ 0 \ : \ 1/x \) will fail with a division by zero
- For the non-programmers: in C++
  - \&\& means logical and
  - || is logical or
Movie: A gentler damBreak
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

No C++, please. We're users!
Filling from below

- Ignaz would like to investigate whether there are faster ways to fill the geometry than throwing balls
  - Liquid is inserted on the right
  - Height of the insertion depends on the geometry size
- But Ignaz is too lazy to rewrite the `blockMeshDict`
  - He only changes the boundary condition on the `rightWall` to `mixed`
- This is what he did:
  ```
git checkout 0050_waterFromTheRight -b staticBC
```
  ```
  He sets the static boundary condition with
  ```
  ```funkySetBoundaryField -time 0```
alpha1 before

```plaintext
boundaryField
{
    leftWall
    {
        type zeroGradient;
    }

    rightWall
    {
        type mixed;
        value uniform 0;
        refValue uniform 0;
        refGradient uniform 0;
        valueFraction uniform 0;
    }
}
```
Excerpt from `funkySetBoundaryFields`

```python
blowIn {
    field alpha1;
    expressions
    {
        target refValue;
        patchName rightWall;
        expression "1";
    }
    {
        target valueFraction;
        patchName rightWall;
        variables ("maxY=max(pts().y);"
                    "thres=0.2*maxY;"
        );
        expression "pos().y<thres?1:0";
    }
}
```
alpha1 after

rightWall
{
    type mixed;
    value uniform 0;
    refValueOld uniform 0;
    refGradient uniform 0;
    valueFractionOld uniform 0;
    refValue uniform 1;
    valueFraction nonuniform List<scalar>
    <cont>>
    50
    (1
    1
    1
    1
}

No C++, please. We’re users!
Executing the solver

- The version 1.7.1 of the interFoam has a strange bug when using mixed boundary conditions
  - This is fixed in 1.7.x
  - But on the stick we only have 1.7.1

- Nevertheless we can execute these examples with another solver

  pyFoamRunner.py --clear --progress interFlux

- But what is this interFlux-solver?
- It is part of ...
pythonFlu

- pythonFlu is a Python-binding for OpenFOAM
  - by Alexey Petrov
- Makes it possible to access all the OpenFOAM-functionality from Python
  - With all the speed of C++
  - All the comfort of Python
- So interFlux is a solver written in Python
  - With the speed of interFoam
  - and the same functionality
Movie: Liquid from the right

Bernhard F.W. Gschaider  No C++, please. We’re users!
Outline

1. Starting up
   What it’s about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
Outline

1. Starting up
   - What it's about
   - Technicalities
2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement
3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions
4. First numbers - Simple evaluations
   - A new field
     - simpleFunctionObjects
     - Complicated Expression
     - Working with subsets
5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact
6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions
7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution
8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
Only the velocity that matters

- Ignaz is not interested in the velocity in the air
- He wants to have the pure liquid velocity for post-processing

```bash
git checkout 0060_expressionField -b exprfield
```
Function objects

- `functionObjects` are plugins that can be loaded at run-time
- They can be executed at every time-step and do various things
  - Post-process
  - Manipulate the solution
  - etc
- A number of `functionObjects` comes with OpenFOAM
- To use a `functionObject`
  - The library that defines it has to be loaded in the `libs-list` in the `controlDict`
  - It has to have an entry in the `functions-list`
    - An entry is a dictionary with various entries
      - `type` Determines which `functionObject` this is
      - `outputControl` when the `functionObject` is executed
    - All other entries depend on the `functionObject-type`
Entry in the controlDict

```plaintext
libs (  
    "libswakFunctionObjects.so"
);

functions (  
    liquidVel {  
        type expressionField;  
        outputControl outputTime;  
        fieldName Uliquid;  
        expression "U*alpha1";  
        autowrite true;
    }
);
```
The `expressionField`

- Create a new field from an expression
  - `fieldName` Name under which this field will be known
  - `expression` What should be in the field
  - `autowrite` Whether the field should be written to disc at the usual write times (otherwise the field is only of use for other expressions)
Two velocities - Before

Figure: Overall velocity
Two velocities - After

Figure: Liquid velocity

Bernhard F.W. Gschaider

No C++, please. We're users!
Outline

1. **Starting up**
   - What it's about
   - Technicalities

2. **A crash course in PyFoam**
   - Description
   - Usage
   - Advertisement

3. **Building a new dam - Case setup**
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. **First numbers - Simple evaluations**
   - A new field
     - `simpleFunctionObjects`
   - Complicated Expression
   - Working with subsets

5. **Looking at other entities**
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. **Dynamic boundaries**
   - Introducing `groovyBC`
   - Finding errors in boundary conditions

7. **Advanced topics**
   - Storing values
   - Calculations on `sampledSets`
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. **Going out**
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

---

*Bernhard F.W. Gschaider*

*No C++, please. We’re users!*
How fast is it filling

• Ignaz wants to see during the simulation how fast the geometry is filling with liquid

• This is what he does to the case

```bash
git checkout 0070_simpleFunctionObject -b simpleFO
```

• This is how he runs it

```bash
pyFoamPlotRunner.py --progress --clear interFlux
```
What is `simpleFunctionObjects`

- `simpleFunctionObjects` is a collection of `functionObjects`
  - Mostly for post-processing
    - Calculates and outputs extremes, averages and integrals on
      - volumes
      - patches
  - Other stuff: writing additional fields, tracking dictionaries etc
- Some of the functionality is now present in OpenFOAM
- But it is kept in `simpleFunctionObjects` for backward compatibility
Addition to controlDict

```plaintext
libs (  
    "libswakFunctionObjects.so"
    "libsimpleFunctionObjects.so"
);

functions (  
    totalLiquid {  
        type volumeIntegrate;  
        fields (  
            alpha1
        );  
        verbose true;
    }
    liquidVel {  
        type expressionField;  
        outputControl outputTime;  
        fieldName Uliquid;  
        expression "U*alpha1";
        autowrite true;
    }
);
```

Bernhard F.W. Gschaider

No C++, please. We’re users!
What `simpleFunctionObjects` usually do

- Data gets written into a directory that is named after the name of the function object.

- Usual fields:
  - `verbose` Also write the data on the standard output (this is necessary for the next step).
  - `fields` List of fields on which the FO should work.
  - `patches` If the FO works on patches these are specified in this list.
The `customRegexp`

- Every entry in the `customRegexp` file is one data set

  `expr` A regular expression that the output of the solver is scanned for
  
  - Each group enclosed in () is one data item
  
  - `%f%` is an abbreviation for the regular expression that matches a floating point number

  `theTitle` Title of the plot

  `titles` Names of the data sets

```
liquid {
  expr "Integral\_of\alpha1=%f%";
  theTitle "Total\_liquid";
  titles ( total );
}
```
Total amount of liquid

- Bernhard F.W. Gschaider
- No C++, please. We’re users!
Starting up
A crash course in PyFoam
Building a new dam - Case setup
First numbers - Simple evaluations
Looking at other entities
Dynamic boundaries
Advanced topics
Going out

A new field
simpleFunctionObjects
Complicated Expression
Working with subsets

Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
The tide is high

- The amount of liquid is interesting, but Ignaz wants to know more:
  - What is currently the highest level of the liquid?
  - Definition of liquid is: $\alpha > 0.5$

- We go to:
  
  ```bash
  git checkout 0080_liquidHeight -b liquidHeight
  ```
General expressions

• Adding

"libsimpleSwakFunctionObjects.so"

gives us a combination of the simpleFunctionObjects and swak-expressions:

• Calculating arbitrary expressions on any entity swak knows about

  type  Always swakExpression

  valueType  Type of the entity. Other entities than internalField might need additional specifications

  expression  The expression to be calculated

  accumulations  The calculated values have to be distilled into a single value. This is a list of specifications how this should be done. Possible values are min, max, average and sum (Finding out what they do is left to the reader as an exercise)
Maximum height of the liquid

```python
maxHeight {
    type swakExpression;
    valueType internalField;
    expression "(alpha1>0.5)?pos().y:0";
    accumulations (max
        max
    );
    verbose true;
}
```
Average height of the liquid

Note: if this gives you a floating point exception, probably the expression \( \text{sum}(\text{alpha}1*\text{vol}()) \) is 0 in the beginning. Modification is left to the reader as an exercise

```cpp
liquidHeight {
    type swakExpression;
    valueType internalField;
    expression "pos().y*alpha1*vol()/sum(<brk>
        <cont>alpha1*vol())";
    accumulations (sum
            sum
    );
    verbose true;
}
```
Calculating on patches

- For patches there is a specialized FO patchExpression
  - But you can also use swakExpression with valueType set to patch
  - But patchExpression makes working with more than one patch at a time possible
- The entry patches lets you specify a list of patches
Height on the walls

```python
wallHeight {
    type patchExpression;
    expression "(alpha1>0.5)?pos().y:0";
    patches (
        leftWall
        rightWall
        lowerWall
    )
    accumulations (max)
    )
    verbose true;
}
```
Joining plots into one

- An expression in `customRegexp` can only match one line of the output
  - Sometimes we want information from more than one line in a plot
  - To achieve this we have two types of entries:

  **master** By default every plot is a master (it has its own plotting window). Noting has to be done

  **slave** This entry doesn’t get his own window but it appends its data to another window. The following entries have to be added:

    **type** This is set to slave

    **master** Name of the plot that the data is added to
Master and slave in customRegexp

```plaintext
height {
    expr "Expression\textsubscript{liquidHeight}:\text{\textsum}=%.\text{\text\textcircled{f}}\%";
    theTitle "Liquid\textsubscript{height}";
    titles ( average );
}
maxheight {
    expr "Expression\textsubscript{maxHeight}:\text{\textmax}=%.\text{\text\textcircled{f}}\%";
    titles ( max );
    type slave;
    master height;
}
```
Liquid height

![Graph of liquid height](image)

- average
- max
Dynamic plots

- Sometimes a number of similar data-sets should be joined into a plot
  - The pattern is similar. They are only distinguished be their name

- This kind of plot can be set using these entries:
  - `type` is set to `dynamic` (a dynamic plot can’t be a slave)
  - `idNr` The number of the pattern that contains the name
    (according to the regular expression convention this the number of the opening ( of a ()-group)
Don’t know how many names we’ve got

1. wallheight {
   theTitle "Height on walls";
   expr "Expression wallHeight on (.+): <brk>
   <cont>max=(%f%)";
   titles ( wall );
   type dynamic;
   idNr 1;
}

Bernhard F.W. Gschaider No C++, please. We're users!
Liquid height on the walls

![Graph showing liquid height on the walls with three different lines representing different walls. The x-axis ranges from 0 to 1, and the y-axis ranges from 0 to 0.3. The lines are labeled as follows: leftWall_wall, lowerWall_wall, and rightWall_wall.]
BTW: debugging the customRegexp

- The Plot-utilities have a number of command line options that help with debugging/working with customRegexp:

  - *dump* Output the data as internally stored (helps finding problems with missing ; or similar)
  - *exclude-regexp-fitting* Exclude all plots that fit a regular expression (used more than once. Processed in the order of occurrence)
  - *include-regexp-fitting* To switch on excluded plots again (by default all are included)
  - *list-custom-Regexp* List all the plots. Shows whether they are switched on or off by exclude/include
Outline

1. Starting up
   - What it's about
   - Technicalities
2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement
3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions
4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets
5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact
6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions
7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution
8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

No C++, please. We’re users!
How much liquid is in the 'pools'

- A swimming pool manufacturer approaches Ignaz because he thinks the damBreak-case might help him to find optimal filling strategies for pools

- For a first calculation Ignaz defines the cavities to the left and the right of the dam as pools
  - He wants to define them as cellSets

```bash
git checkout 0090_setCalculations -b setCalc
```
Creating cellSet’s

- Since old times (but known by few) it is possible to define blocks as cellSet’s in blockMeshDict
  - Just insert a word in the block-specification

```plaintext
blocks
{
  hex (0 1 5 4 12 13 17 16) leftPool (23 8 1) <brk>
  simpleGrading (1 1 1)
  hex (2 3 7 6 14 15 19 18) rightPool (19 8 1) <brk>
  simpleGrading (1 1 1)
  hex (4 5 9 8 16 17 21 20) (23 42 1) simpleGrading (1 1 <brk>
  simpleGrading (1 1 1)
  hex (5 6 10 9 17 18 22 21) (4 42 1) simpleGrading (1 1 <brk>
  simpleGrading (1 1 1)
  hex (6 7 11 10 18 19 23 22) (19 42 1) simpleGrading (1 <brk>
  simpleGrading (1 1 1)
};
```
Defining a cellSet depending on a position

- The cellSet-utility that comes with OpenFOAM allows one to create a cellSet
  - By adding "libswakTopoSources.so"
    to the libs entry it can be extended to use Swak
- The type of the new creator is expressionToCell and all cells for which the logical expression evaluates to true are added
- Ignaz uses this to create a cellSet at the starting position of the liquid
Starting position

```cpp
// Name of set to operate on
name inBall;

// One of clear/new/invert/add/delete | subset/list
action new;

topoSetSources
(
    expressionToCell
    {
        expression "alpha1 > 0.5";
    }
);
```

Bernhard F.W. Gschaider
Defining a `faceSet`

- Similarly there is a creator `expressionToFace` that creates a `cellSet`.

- `swak` can’t return expressions defined on faces

  - To work around this the definition is: “a face belongs to a `faceSet` if the expression evaluates to `true` in one of its cells and to `false` in the other”
Amount of liquid in the starting position

```python
inBall {  
    type swakExpression;
    valueType cellSet;
    setName inBall;
    expression "(alpha1*vol())/sum(vol())";
    accumulations (  
        sum
    );
    verbose true;
}
```
The `flip()`-function

- This gives back the `flipMap (orientation)` of a `faceZone`
  - Is needed to get the right sign of `phi`
- For a `faceSet` `OpenFOAM` does not define a `flipMap`
  - By default `flip()` gives back 1 for the whole set
  - If a `cellSet` exists that has the same name as the `faceSet` plus `SlaveCells` then `flip()` is calculated from this
  - This is the same convention as the `setsToZones-utility` of `OpenFOAM`
Flow through the starting position

```plaintext
flowBall {
  type swakExpression;
  valueType faceSet;
  setName aroundBall;
  expression "phi*flip()\*alpha1";
  accumulations (sum);
  verbose true;
  autoInterpolate true;
  warnAutoInterpolate false;
}
```
An alternate way to calculate the flow through a faceSet would be to calculate $\phi$ by hand:

$\text{expression } \text{"}(U \& \text{Sf())*flip()}*\text{alpha1}\text{";}$
Collecting the results

```plaintext
inPools {
  theTitle "Liquid\textsubscript{in} regions";
  expr "Expression\textsubscript{in (.+)}:\textsubscript{regions} average=(\%f%) min=(\%f%) max=(\%f%) <brk> <cont>";
  titles ( average min max);
  type dynamic;
  idNr 1;
}

inBall {
  expr "Expression\textsubscript{in Ball} sum=(\%f%)";
  titles ( "In\textsubscript{ball}"");
  type slave;
  master inPools;
}

flowBall {
  theTitle "Flows";
  expr "Expression\textsubscript{flow (.+)} sum=(\%f%)";
  type dynamic;
  idNr 1;
}
```
Normalized amount in regions
What goes where

Flows

- Pools
- Out on atmosphere
- Out on lowerWall
- Out on leftWall
- Out on rightWall
- Bell

Bernhard F.W. Gschaider
No C++, please. We're users!
Outline

1. Starting up
   - What it’s about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

Bernhard F.W. Gschaider

No C++, please. We’re users!

100/176
Outline

1 Starting up
   What it's about
   Technicalities
2 A crash course in PyFoam
   Description
   Usage
   Advertisement
3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions
4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets
5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact
6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions
7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution
8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye
Preparing for the sum

- Ignaz wants to have a quick overview how much liquid is in all the pools.

- Alternatives would be:
  - Calculate this by hand from the available results.
  - Create a new `cellSet` that contains both pools.

- But instead Ignaz does this:

  ```
  git checkout 0100_remoteVariables -b remote
  ```
Syntax of remote variables

- A variable is a remote variable if between the name and the = there is a string enclosed by {} that tells us where the variable is to be calculated.
- The string is split into two parts by a ‘’ and has an optional part separated by /:
  
  **type of the entity** this is one of the types known to swak (like internalField, cellSet, patch). If no type is given patch is assumed.
  
  **name of the entity** speaks for itself. If the type is internalField then no name is given.
  
  **mesh region** This is optional for multi-region meshes. The entity is now sought on another mesh with that name.

- The weird syntax is used because these are the only characters that can not be part of a valid entity name.
Total sum

```
inTotal {
    type swakExpression;
    valueType cellZone;
    zoneName leftPool;
    accumulations (average
    );
    expression "(sum(alpha1*vol())+alphaBall+alphaRight)/(sum(vol())<brk>
        +volBall+volRight)";
    variables (
        "alphaRight{cellZone'rightPool}=sum(alpha1*vol());"
        "volRight{cellZone'rightPool}=sum(vol());"
        "alphaBall{cellSet'inBall}=sum(alpha1*vol());"
        "volBall{cellSet'inBall}=sum(vol());"
    );
    verbose true;
}
```
Collecting the sum

```plaintext
inTotal {
  expr "Expression\_\_inTotal\_\_:\_average=(\%f<\text{cont}>\%)");
  titles ( "Overall\_average");
  type slave;
  master inPools;
}
```
Normalized amount in regions with average

![Graph showing liquid in regions with different markers and lines for different values.](image)
Outline

1 Starting up
   What it's about
   Technicalities

2 A crash course in PyFoam
   Description
   Usage
   Advertisement

3 Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4 First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5 Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6 Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7 Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8 Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
Just give me a single number

- Ignaz would like a short “report” for the last time-step
  - With stuff that he already calculated
    - Liquid flow through the ball-region
    - Stuff that hasn’t been calculated yet
      - Pressure difference between top and bottom
  - He sets the case to
    
    ```
    git checkout 0110_funkyDoCalc -b doCalc
    ```
  - And starts editing a dictionary `checkSimulation`
Something we already calculated

```plaintext
flowBall {
    valueType faceSet;
    setName aroundBall;
    expression "phi*flip()*alpha1";
    accumulations ( sum );
    autoInterpolate true;
    warnAutoInterpolate false;
}
```
Pressure difference

```python
pressureDiff {
    valueType patch;
    patchName lowerWall;
    variables (
        "topP{patch'atmosphere}=average(p);"
    );
    expression "p-topP";
    accumulations (min max average );
}
```
Doing the calculation

```
funkyDoCalc -time 0.05: checkSimulation
```

1. Time = 1
   totalLiquid : sum=0.000525861
2. liquidHeight : sum=0.0596091
   maxHeight : max=0.181999
3. flowBall : sum=−5.16526e−06
   inTotal : average=0.292424
4. pressureDiff : min=119.405 max=1163.75
   <brk>
   <cont>average=536.356
End
Outline

1. Starting up
   What it’s about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
Things at the inlet are not always the same

- The pool-builder finds the initial results quite interesting
  - “But we have a different filling procedure. The amount of water changes”
- So Ignaz has to find a way to set changing boundary conditions
- He sets the case to
  
  ```
git checkout 0120_groovyBC -b groovyBC
  
  and starts to use groovyBC
  ```
What is groovyBC

- It is the built-in boundary condition mixed on steroids
  - All relevant fields can be set by expressions
    - `valueExpression` the value if the BC is to be considered a Dirichlet-conditions. This always has to be set
    - `gradientExpression` Value of the gradient if the condition is a Neuman-condition. Defaults to 0 (or equivalent for vectors and tensors)
    - `fractionExpression` Whether this is a Dirichlet condition (1) or a Neuman (0). Defaults to 1
  - To use the groovyBC the entry
    "libgroovyBC.so"

has to be added to the libs-list
Changing velocities

```plaintext
rightWall
{

type groovyBC;
value uniform (0 0 0);
variables (
    "top=max(pts().y)/5;"
    "s=pos().y/top;"
);
valueExpression "-normal()*inVel";
timelines (  
    {  
        name inVel;
        outOfBinds clamp;
        fileName "$FOAM_CASE/inVelocity.data";
    }
)
};
```
Additional fields in a `groovyBC`

**timelines** List of timeline data. They can be used under the name in expression and will return the value at the current time.

- They use the same mechanism for reading timelines as `timeVaryingUniform`. Therefore the format is similar and if you have a recent incarnation of 1.7.x that has the right patches you can also read CSV-files.

**lookuptables** Like timelines but the name can be used as a function where the value is looked up.
The time-line data

- A list of pairs with
  1. time (these have to be sorted)
  2. value

```
1
(0 0)
(0.1 1)
(0.3 1)
(0.5 1.5)
(0.8 1.5)
(0.9 1)
```

Bernhard F.W. Gschaider

No C++, please. We’re users!
Non-homogeneous inlet

rightWall
{
  type groovyBC;
  value uniform 0;
  variables ( 
    "top=max(pts().y)/5;"
    "s=pos().y/top;"
    "switchOff=1.5;"
  );
  fractionExpression "s<=1?1:0";
  valueExpression "(time()<switchOff?1:0)*profile(s)";
  lookuptables ( 
    { 
      name profile;
      outOfBounds clamp;
      fileName "$FOAM_CASE/inProfile.data";
    } 
  );
}
The profile data

(0.3 1)
(0.7 0.8)
(0.9 0.5)
(1 0.1)
Movie: Switched off inlet
The one who doesn’t do anything makes no mistakes

- Sometimes Ignaz has a problem with one of his `groovyBC` s
  - A switch occurs at the wrong time
  - He got the coefficients of some polynomials wrong
  - ...
- Usually this things only reveal themselves when the simulation is finished
- To check out these things there is the utility `replayTransientBC`
  - Also works with other transient boundary conditions

```
git checkout 0130_replayTransientBC -b replayBC
```
The `replayTransientBCDict`

- Only one entry
- List of all the fields which
  1. should be loaded in the beginning
  2. the boundary conditions are evaluated
  3. the fields get written

```plaintext
fields (  
  U
  alpha1
);  
```
Some BCs have to be modified

- Boundary conditions that depend on variables that are calculated by the simulation
  - Like phi
  - Affected is also the whole inletOutlet-family
- Of course if the BC depends on the solution provided by the solver, replayTransientBC can’t help too
Movie: Only the conditions
Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

No C++, please. We’re users!
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
     - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
How much liquid entered the system?

- Ignaz wants to check the conservation in his simulation
  - To do this he wants to know the accumulated amount of liquid that entered the system
  - But all currently known function objects only know about the current state
- But the solution is there: stored variables

```
git checkout 0140_storedVariableInflow -b storedVars
```
Using stored variables

- Stored variables are declared in a list `storedVariables` where each element has two entries
  
  ```
  name  name of the variable (surprising, isn’t it?)
  initialValue  value of the variable in the beginning
  ```

- The variable can be used like any other variable

- When the variable is assigned a value then this is the new value
  
  - For accumulations you’ve got to explicitly add it to the old value
  - The value from the `last` time-step is used
    
    - Not the last one from the current time-step (if the variable is for instance evaluated multiple times during a PISO-loop)
Accumulated liquid

```plaintext
accumulatedInflow {
    type patchExpression;
    patches (  
        rightWall
        atmosphere
    );
    variables (  
        "patchFlow=patchFlow-deltaT()*alpha1*phi;"
    );
    storedVariables (  
        {  
            name patchFlow;  
            initialValue "0";
        }
    );
    expression "patchFlow";
    accumulations (  
        sum
    );
    verbose true;
}
```
Getting the values

```plaintext
rightIn
{
    expr "Expression\accumulatedInflow\on<brk>
        <cont>rightWall:sum=(%f%)");
    titles ( inflow );
    type slave;
    master liquid;
}
```
Accumulated amount of liquid is the same
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

No C++, please. We’re users!
Getting the height at a special position

- The pool builder informs Ignaz that the simulations look nice but there is no way that he can compare the calculated height to the measurements
  - The liquid height sensor only measures the height at one location in the pool
- Ignaz asks for the location of the sensor and decides to use a sampledSet there

```bash
git checkout 0150_sampledSet -b sampledSets
```
Using sampled sets in swak4Foam

- `sampledSet` is a facility in OpenFOAM that allows sampling fields at specified locations
  - Usually lines but points are also possible

- Essential entries in the dictionary are:
  - `setName` specifies the name of the sampled set (under this name it can be accessed)
  - `set` Dictionary that specifies the `sampledSet` the way it is specified for the `sample`-utility
    - swak4Foam introduces a new type `swakRegistryProxy`. This reuses a previously defined `sampleSet` (identified by the `setName`)

- The usual operations can be done on the `sampledSet`
  - For fields the sampled values are used
Maximum height

```plaintext
fillHeight {
    type swakExpression;
    valueType set;
    verbose true;
    setName lineUp;
    set {
        type uniform;
        axis y;
        start (0.2 0 0.007);
        end (0.2 0.585 0.007);
        nPoints 200;
    }
    expression "((alpha1>0.5)?pos().y:0)";
    accumulations (max
    );
    interpolate true;
    interpolationType cellPoint;
}
```
All-time high

```plaintext
maxFillHeight {
  type swakExpression;
  valueType set;
  verbose true;
  setName lineUpCopy;
  set {
    type swakRegistryProxy;
    axis y;
    setName lineUp;
  }
  expression "currentMax";
  accumulations ( max
    );
  variables ( "currentMax=(alpha1>0.5)?pos().y:currentMax;"
    );
  storedVariables ( 
    { 
      name currentMax;
      initialValue "0";
    } 
  );
}
```
Recording the values

```plaintext
fillHeight {
    theTitle "Filling\_height";
    expr "Expression\_fillHeight\_max=(\%f\%)";
    titles ( current );
    progress "h:\$0";
}

maxFillHeight {
    expr "Expression\_maxFillHeight\_max=(\%f\%)" <brk>
        <cont>;
    titles ( maximum );
    type slave;
    master fillHeight;
    progress "max:\$0";
}
```
Filling height and maximum

![Graph showing filling height over time with two lines: one for current and one for maximum values.](image)
Getting the processed data

- Ignaz re-generates the pictures from the pickled data:

  `pyFoamRedoPlot.py --pick PyFoamRunner.interFoam.analyzed/pickledPlots`

- But the customer wants hard numbers to compare them in his spreadsheet program

  `pyFoamRedoPlot.py --csv --file=sampledSets --pick PyFoamRunner.interFoam.analyzed/pickledPlots`
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
The higher it gets, the less comes in

- The customer informs Ignaz that the data looks nice, but not realistic:
  - There are three sensors installed. If one of them gets wet then the inflow-rate is reduced by a third

- Ignaz says that can do this

  `git checkout 0160_switchingBySamples -b switchSamples`
The grand scheme

- Ignaz needs two things to interact because the set can not be created in the BC:
  
  1. The sampledSet threePoints. This is created by a function object of the type `createSampledSet`
  2. The boundary condition. This reads the values at the sensor positions via an external variable and acts accordingly
Probes at three points

```
switchPoints {
  type createSampledSet;
  outputControl timeStep;
  outputInterval 1;
  setName threePoints;
  set {
    type cloud;
    axis x;
    points (
      (0.1 0.3 0.007)
      (0.25 0.3 0.007)
      (0.4 0.3 0.007)
    );
  }
}
```
The switching inlet

```plaintext
rightWall
{
    type groovyBC;
    value uniform 0;
    variables
    {
        "current{set'threePoints}=average(alpha1>0.5?1:0);"
        "maxVal=maxVal<current?current:maxVal;"
        "top=max(pts().y)/5;"
        "s=pos().y/top;"
    }
    storedVariables
    {
    
    
    }
    fractionExpression "s<=1?1:0"
    valueExpression "(1-maxVal)*profile(s)"
    lookuptables
    {
        profile
        outOfBounds clamp
        fileName "$FOAM_CASE/inProfile.data"
    }
}
```
Movie: Running it
Comparing the data

- Ignaz now wants to compare the new data:
  
  ```
  pyFoamRedoPlot.py --pick --csv --file=switching PyFoamRunner.interFoam.analyzed/pickledPlots
  ```

- He merges this data and the one from the old run into a common CSV
  
  ```
  pyFoamJoinCSV.py sampledSetsfillHeight.csv switchingfillHeight.csv joinedFillHeight.csv
  ```

- Which he opens with OpenOffice:
  
  ```
  kde-open joinedFillHeight.csv
  ```

- and produces a nice comparison:
Comparing the filling height
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
Looking at the interface of the liquid

- The pool maker has one last request:
  1. He wants to know the flow through some tilted plane
     - This can not be represented with a faceSet
  2. He wants to know how big the liquid surface is and how fast it is moving

    git checkout 0170_sampledSurfaces -b surfaces

- **Note:** Due to a problem in 1.7.1 sampledSurfaces won’t work on the stick
  - Changing of the size of the surface is not properly handled
  - This is fixed in 1.7.x
Working with sampled surfaces

Is similar to working with sampled sets.
Just replace set with surface
Control surfaces everywhere

```plaintext
flowSurface {
    type swakExpression;
    valueType surface;
    verbose true;
    surfaceName tiltedPlane;
    surface {
        type plane;
        basePoint (0.2 0.2 0);
        normalVector (1 5 0);
        interpolate true;
    }
    expression "(U^&^Sf())*alpha1";
    accumulations (sum);
}
```
There can be too much information
Velocity of the liquid surface

```c
surf
surfaceVel {

type swakExpression;
valueType surface;
verbose true;
surfaceName liquidSurf;
surface {

type isoSurface;
isoField alpha1;
isoValue 0.5;
interpolate true;
}
expression "mag(U)";
accumulations ( 

max 

average

); 
}
```

Bernhard F.W. Gschaider  No C++, please. We’re users!
How fast is the interface

![Graph of velocity of the surface](image_url)

---

**Starting up**
- A crash course in PyFoam
- Building a new dam - Case setup
- First numbers - Simple evaluations
- Looking at other entities
- Dynamic boundaries
- Advanced topics
- Going out

**Storing values**
- Calculations on sampledSets
- Switching according to the simulation state
- On the surface
- Manipulating the solution

---

Bernhard F.W. Gschaider

No C++, please. We’re users!
How big is the interface?

```plaintext
surfaceSize {
    type swakExpression;
    valueType surface;
    verbose true;
    surfaceName liquidSurf2;
    surface {
        type isoSurface;
        isoField alpha1;
        isoValue 0.5;
        interpolate true;
        // type swakRegistryProxy; // doesn’t work due to <brk>
        <cont> bug
        //surfaceName liquidSurf;
        expression "area() / 0.0146";
        accumulations (sum
        );
    }
}
```
How big is the interface
Outline

1. Starting up
   What it's about
   Technicalities

2. A crash course in PyFoam
   Description
   Usage
   Advertisement

3. Building a new dam - Case setup
   The original case
   Setting initial conditions
   Variables
   The Conditional expression
   Static boundary conditions

4. First numbers - Simple evaluations
   A new field
   simpleFunctionObjects
   Complicated Expression
   Working with subsets

5. Looking at other entities
   Sum of liquid in 'special zones'
   Calculations after the fact

6. Dynamic boundaries
   Introducing groovyBC
   Finding errors in boundary conditions

7. Advanced topics
   Storing values
   Calculations on sampledSets
   Switching according to the simulation state
   On the surface
   Manipulating the solution

8. Going out
   What wasn’t shown
   Merging cases with Mercurial
   Ignaz says goodbye

No C++, please. We’re users!
Violating the solution

- Of course this was not the last request

- The last question is: “What is the effect of filling the pools from above (will the rubber ducks survive?)”
  - A preliminary study is needed in 10 minutes

  ```
git checkout 0180_manipulateSolution -b manipulating
  ```

- Ignaz decides to do something ugly to the solution (although his conscience objects)
Producing liquid out of thin air

- He used a function object that manipulates the solution at the end of every time-step:

```python
letItRain {
    type manipulateField;
    outputControl timeStep;
    outputInterval 1;
    fieldName alpha1;
    expression "1";
    mask "pos().x>0.05&&pos().x<0.25&&
pos().y>0.4&&pos().y<0.5";
}
```
Movie: The last picture show
Outline

1. Starting up
   - What it’s about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye
Topics we didn’t touch

Have a look at the Wiki-page and the examples

- Differential operators on fields
- Plugging expression source terms into a solver
  - This was promised in the abstract. Sorry. Anybody wants to stay for an improvised session on that?
- Groovifying boundary conditions
  - Look it up in the dictionary. The meaning of this word is “Extending an existing boundary condition to use the swak4Foam-machinery instead of constant parameters”
- Other stuff that I’ve forgotten

Bernhard F.W. Gschaider

No C++, please. We’re users!
Final remarks

As a figure of American literature (hint: it is not Huckleberry Finn) said:

With great power comes great responsibility

When using swak4Foam always consider:

**Physical correctness** swak4Foam allows you to do things that are not physical (some of the examples do that). Try not to do this

**Performance** Certain things swak4Foam does (especially the sampled-stuff) are computationally expensive. Try to avoid such operations
Outline

1. Starting up
   - What it’s about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn’t shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

Bernhard F.W. Gschaider

No C++, please. We’re users!
Trying different stuff at the same time

- Sometimes Ignaz does different things with a case at the same time
  - Improve the numerics
  - Test different physical parameters
- Merging the results of these without forgetting something can be a nerve wrecking experience
  - And there is always the issue “Where did this come from”
- For this task PyFoam supports us with its Mercurial-support
  - Other VCS-systems are possible
Setting up two different cases

- Make sure the master is under version control
  
  cd masterCase
  pyFoamInitVCSCase . --commit-message="Initial commit"

- Create two working cases (pyFoamCloneCase.py knows about Mercurial)
  
  cd ..
  pyFoamCloneCase.py masterCase numericsTests
  pyFoamCloneCase.py masterCase parameterTests
Working on the cases

cd numericsTests

    hg branch numericsBranch

• Work on the numerics
• Review the changes (Mercurial-style)

    hg diff

• Review the changes (pyFoam-style)

    hg foamdiff .

• Commit the changes

    hg commit -m "Now the numerics seem to work"

• Push to the master case

    hg push --new-branch

• Do the same in the physics-case
Getting all the changes

- Got to the master
  
  `cd ../masterCase`

- Merge in the numerics
  
  `hg merge numericsBranch`

- Merge in the physics:
  
  `hg merge physicsBranch`

- All the wisdom in one case
  
  `hg log`
Why Mercurial?

- It can do the same things as git but it is simpler and more consistent
- It is written in Python
  - So interaction with PyFoam is easy
- It is easily extendable
  - Writing a plugin like foamdiff for git is no fun
- If the case is in a git-controlled directory, then the two do not collide
  - Which they would if PyFoam used git for that
Outline

1. Starting up
   - What it's about
   - Technicalities

2. A crash course in PyFoam
   - Description
   - Usage
   - Advertisement

3. Building a new dam - Case setup
   - The original case
   - Setting initial conditions
   - Variables
   - The Conditional expression
   - Static boundary conditions

4. First numbers - Simple evaluations
   - A new field
   - simpleFunctionObjects
   - Complicated Expression
   - Working with subsets

5. Looking at other entities
   - Sum of liquid in 'special zones'
   - Calculations after the fact

6. Dynamic boundaries
   - Introducing groovyBC
   - Finding errors in boundary conditions

7. Advanced topics
   - Storing values
   - Calculations on sampledSets
   - Switching according to the simulation state
   - On the surface
   - Manipulating the solution

8. Going out
   - What wasn't shown
   - Merging cases with Mercurial
   - Ignaz says goodbye

Bernhard F.W. Gschaider
No C++, please. We’re users!
We’re leaving Ignaz

- For all his trouble Ignaz got a nice pool installed in his garden
- He lies in that pool thinking “Who needs C++?”
- We leave him there
  - Wondering: “Will he come back? Or will he lie in the sun forever doing studies on heat transfer by radiation?”
Bring your bugs

A program without bugs wasn’t worth writing in the first place

I’m sure as you start using swak4Foam you will encounter bugs

- Bugs in swak4Foam (my ego can stand these)
- Bugs in OpenFOAM (a couple of these was already found during the development of swak4Foam)

If you find a bug please don’t grumble and gossip about it, but report it on the Mantis-Bugtracker of the openfoam-extend-project
Bring your examples

- This is the call for a competition: **Sickest use of swak4Foam**

- The example should:

  - Demonstrate the use of swak4Foam (no C++-programming should be required)
  - Do something astonishing that nevertheless makes sense (either on a physical or on a spiritual level)
  - Be simple
    - Run on a state-of-the-art PC in under an hour
    - Have a blockMesh-geometry (or something else that comes with OF)
  - Should be shown to me at least after the end of the last session

- The winner

  - will be announced after the conference dinner at the bar (if there is one) by a committee (which consists of me, Fritz and Wolfgang)
  - will receive a refreshing beverage of choice (alcoholic optional)