README for swak4Foam

Bernhard F.W. Gschaider

October 18, 2012

Contents

1 Description

A collection of libraries and tools that let the user handle OpenFOAM-data based on expressions

2 About this document

2.1 Scope

This file gives an overview of `swak4Foam` and a history of the features. It is not a canonical documentation.

2.2 Technical

This file is written in the mighty `org-mode` (see http://orgmode.org/) a markup/outline-mode for (X)Emacs. Using this mode it can be easily (using

3 keystrokes ... it's Emacs) to PDF or HTML to make it more readable (and add a table of contents).

Please don't try to "beautify" it with any other text editor as this will surly mess up the markup (and keeping the file *org-compatible* outside of the `org-mode` is a pain in the neck.

3 Contributors etc

3.1 Original Author

Bernhard Gschaider (bgschaid@ice-sf.at)

3.2 Current Maintainer

Bernhard Gschaider (bgschaid@ice-sf.at)

3.3 Contributors

In alphabetical order of the surname

Peter Keller `sprinklerInlet`-case

Andreas Otto fixed the `circulatingSplash`-case

Alexey Petrov `pythonFlu`-integration

Bruno Santos
- Compilation with Intel compiler and Mingw
- Rewrite of `mybison` and `myflex` to allow parallel compilation with `WM_COMPPROCS`

Martin Becker The `potentialPitzDaily`-case (demonstrating a problem with `groovyBC`)

If anyone is forgotten: let me know

Contributors to `simpleFunctionObjects` are listed separately in the `README` of that library

3.4 Documentation

See: http://openfoamwiki.net/index.php/contrib/swak4Foam

4 Installation/Compilation

4.1 Requirements

- Version 2.0 or 2.1 of OpenFOAM (a version that works with 1.6, 1.6-ext and 1.7 is available separately)
  - The `finiteArea`-stuff will probably work with version 2.0-ext (once that is available)
- the compiler generators `bison` and `flex`

  bison `swak4Foam` is known to work with `bison` version 2.4 and higher. Version 2.3 compiles but the plugin-functionality does not work correctly

  flex since the introduction of the plugin functions at least a flex version of `2.5.33` is required (`2.5.35` is the lowest **confirmed** version)

Both of these are mainstream packages (they are for instance needed to compile `gcc`) and should exist on every Linux distribution. Use the package manager of your distribution to install them and **only** if the compilation process of `swak4Foam` complains about too low versions try to install them from other sources.

`swak4Foam` tries to keep the requirements on these as low as possible and sometimes lower versions than the ones reported **may** work. If they do please report so.

The version of bison can be checked with

```
bison -V
```

The version of flex with

```
flex -V
```

4.2 Building

```
wmake all
```

at the base directory should build all the libraries and tools.

Rerun the command to make sure that there was no problem with the compilation (this should be quite fast and only report libraries being created and some administrative stuff)

4.2.1 Additional configuration

Some features (currently only the `Python`-integration may need third party software. The paths to these packages can be configured in a file `swakConfiguration` (an example file `swakConfiguration.example` is provided). If that file is not present these unconfigured features will not be compiled.

Environment variables that can be set in this file are:

SWAK$_{\text{PYTHONINCLUDE}}$ Path to the `Python.h` file of the used `python`-installation

SWAK$_{\text{PYTHONLINK}}$ Options to link the `python`-library to the library for the `python`-integration

SWAK$_{\text{USERPLUGINS}}$ A list of paths separated by semicolons. These are the directories of libraries with function-plugins. They are compiled in the course of the normal `swak`-compilation. This makes sure that they are consistent with the `swak`-release in the case of an update

4.2.2 Possible compilation failure with old 2.0.x-versions

With older versions of 2.0.x (or 2.0 or 2.0.1) it is possible that the compilation of `swakCodedFunctionObject` will fail. In that case remove the last parameter to the `codedFunctionObject`-constructor in `swakCodedFunctionObject.C` (it is clearly marked by a comment)

4.3 Global installation

If the libraries and utilities are considered stable and the should be available to everyone (without having to recompile them) the script `copySwakFilesToSite.sh` in the directory `maintainanceScripts` can be used to copy them to the global site-specific directories.

The script `removeSwakFilesFromSite.sh` in the directory `maintainanceScripts` removes all these files from the global directories. The `removeSwakFilesFromLocal.sh` does the same for the user directories (this makes sure that no self-compiled version *shadows* the global version (which would lead to strange results)

There is a `Makefile` attached. `make globalinstall` compiles `swak4Foam` and installs it into the global directories

Note: Due to the order in which library direcories are searched for with `-L` a global installation might break the compilation. **If you don't know what this means: don't attempt a global installation**

4.4 Packaging

4.4.1 Debian

The command `build dpkg` builds a Debian/Ubuntu package for the **currently enabled** OpenFOAM-package. Note:

- it is assumed that the currently used OF-version was also installed by the package manager
- the `dev` package is built but poorly maintained

Changes in the packaging should be done in the branch `debianPackaging` of the Mercurial-repository and later be merged to the `default`-branch.

Packaging for OpenFOAM 2.x should be done in the branch `debianPackaging_2.x`

Note: Due to the problem described with the global installation it might be necessary to deinstall a previously installed package to successfully build a new package

5 Contents

5.1 Libraries

Collection of Libraries

5.1.1 `swak4FoamParsers`

The basis of `swak4Foam`: the expression parsers with the logic to access the *OpenFOAM* data-structures.

None of the other software pieces compile without it.

Also defines a subclass to `DataEntry` that uses `swak`-expressions and a function object `initSwakFunctionObject` that might be used if this fails

5.1.2 `simpleFunctionObjects`

A collection of function objects that was previously separately available at http://openfoamwiki.net/index.php/Contrib_simpleFunctionObjects.

Provides consistent output of values (on patches and fields) and more.

5.1.3 `groovyBC`

Implements the infamous `groovyBC`. A boundary condition that allows arbitrary expressions in the field-file

5.1.4 `swakFunctionObjects`

Function objects that have no additional requirements. Mainly used for manipulating and creating fields with expressions

addGlobalVariable Adds a variable to a global swak-namespace. Mainly used for debugging and resolving issues where a variable is needed in a BC before it is defined.

expressionField Create a new field from an expression

clearExpressionField Erase a field created with `expressionField`

manipulateField Modify a field in memory

createSampledSet Create a sampled set that can be used by other swak-entities (mainly boundary conditions)

createSampledSurface Create a sampled surface that can be used by other swak-entities (mainly boundary conditions)

swakCoded Child of the `coded`-functionObject that can read and write global variables from and to swak-namespaces

Assumes that the `SWAK4FOAM_SRC` environment variable is set to the `Libraries`-directory of the `swak4Foam`-sources

solveLaplacianPDE Solve the Poisson equation

$$\frac{\partial \rho T}{\partial t} - \nabla \lambda \nabla T = S_{expl} + S_{impl} T \tag{1}$$

for T where ρ, λ and S can be specified

solveTransportPDE Solve the transport equation

$$\frac{\partial \rho T}{\partial t} + \div(\phi, T) - \nabla \lambda \nabla T = S_{expl} + S_{impl} T \tag{2}$$

for T where ρ, λ and S can be specified. Plus the name of the field ϕ

5.1.5 `simpleSwakFunctionObjects`

Function objects based on the `simpleFunctionObjects`-library (which is a prerequisite for compiling it).

Evaluate expressions and output the results

5.1.6 `swakSourceFields`

These classes allow to manipulate the solution. To use these the solver has to be modified.

expressionSource Field that is calculated from an expression. To be used as source-term or coefficient in some solver

forceEquation force an equation to fixed values in selected locations. Has to be used after constructing the equation and before solving

These sources are based on `basicSource` and can be used **without** a modification of the solver (they are only available in the 2.x version):

SwakSetValue sets values according to a mask or the mechanism provided by `basicSource`

SwakExplicitSource Uses the calculated source term on the right hand side of the equation

SwakImplicitSource Uses a calculated scalar-field to add an implicit source term (source is **without** the actual field)

5.1.7 `swakTopoSources`

`topoSources` for `cellSet` and `faceSet`. Can be used with the `cellSet` and `faceSet`-utilities

5.1.8 `swakFiniteArea`

Implements parsers for the `finiteArea`-stuff in 1.6-ext. Also implements `groovyBC` for `areaField` and `expressionField` and other function objects

5.1.9 `groovyStandardBCs`

Collection of boundary conditions that give standard boundary conditions the possibility to use expression for the coefficients

Contributions to this library are explicitly encouraged. Please use the Mercurial-branch `groovyStandardBCs` to *groovyify* standard boundary conditions.

5.1.10 `pythonIntegration`

Embeds a `Python`-interpreter.

`pythonIntegrationFunctionObject` Executes `Python`-code at the usual execution times of functionObjects. The interpreter keeps its state

This library is only compiled if the paths to the Python-Headers are configured in the `swakConfiguration`-file (see above)

5.1.11 `fluFunctionObjectDriver`

Driver for `functionObjects` that implemented entirely in `Python` using the `pythonFlu`-library

5.1.12 `functionPlugins`

Directory with a number of libraries with function-plugins:

swakFacSchemesFunctionPlugin functions with selectable discretization schemes for FAM (only used in 1.6-ext)

swakFvcSchemesFunctionPlugin functions with selectable schemes for FVM

swakLocalCalculationsFunctionPlugin calculations that are local to a cell (Minimum of the face values or so)

swakMeshQualityFunctionPlugin calculate mesh quality criteria like orthogonality, skewness and ascpect ratio

swakRandomFunctionPlugin different random number distributions. Currently only exponential

swakSurfacesAndSetsFunctionPlugin calculates distances from `sampledSurfaces` and `sampledSets` and projects calculated values from these onto a volume field

swakThermoTurbFunctionPlugin Access functions from the thermophysical model and the turbulence model in the current region. Loads the model only if necessary

swakTransportTurbFunctionPlugin Same as above but for incompressible models

swakLagrangianCloudSourcesFunctionPlugin Functions that get informations like source terms from clouds of particles (due to technical reasons this works only for the regular *intermediate* clouds)

swakVelocityFunctionPlugin Functions that work on the flow field (currently only the local Courant-number)

5.2 Utilities

5.2.1 `funkySetFields`

Utility that allows creation and manipulation of files with expressions

5.2.2 `funkySetAreaFields`

Utility like `funkySetFields` for `areaFields` (only works with 1.6-ext)

5.2.3 `funkySetBoundaryField`

Sets any field on a boundary to a non-uniform value based on an expression.

Acts without deeper understanding of the underlying boundary condition

5.2.4 `replayTransientBC`

Utility to quickly test whether a groovyBC gives the expected results. Writes the specified fields with the applied boundary condition but doesn't do anything else.

Can be used for other BCs as well

5.2.5 `funkyDoCalc`

Evaluates expressions that are listed in a dictionary using data that is found on the disc and prints summarized data (min, max, average, sum) to the screen

5.3 Examples

If not otherwise noted cases are prepared by a simple `blockMesh`-call.

Note: All the cases here are strictly for demonstration purposes and resemble nothing from the 'real world'

5.3.1 groovyBC

The old `groovyBC`-Demos

- pulsedPitzDaily

  Solver pisoFoam

  Also demonstrates `manipulateField`, `expressionField` and `clearField` from the `swakFunctionObjects`. `patchExpression` from `simpleSwakFunctionObjects`. `solveLaplacianPDE` and `solveTransportPDE` for solving equations

- wobbler

  Solver solidDisplacementFoam

- circulatingSplash

  Solver interDyMFoam

- movingConeDistorted

  Solver pimpleDyMFoam

  Also demonstrates `swakExpression` with `surface`. Due to a problem described below this currently doesn't work

- average-t-junction

  Solver pimpleFoam

  Mesh preparation Execute the script `prepare.sh` in that directory (requires PyFoam: if not installed change in the `boundary`-file the type of the `defaultFaces` to `wall`)

- delayed-t-junction

  Solver pimpleFoam

  Demonstrates Delayed variables to simulate an inflow that depends on the value of the outflow

- multiRegionHeaterFeedback

  Solver chtMultiRegionFoam

  Mesh preparation Execute the script `prepare.sh` in that directory

Also demonstrated `patchExpression` and `swakExpression` from `simpleSwakFunctionObjects`.

- fillingTheDam

  Solver interFoam

  Also demonstrates Usage of a `sampledSet` defined in the `controlDict` do determine the average filling height. Also stored variables for not switching back once the criterion is reached. Global variables defined by a function object

- sprinklingInlet

  Solver interFoam

  Description Winner of the `swak4Foam`-competition at the 6th OpenFOAM-Workshop (2011). By Peter Keller

- potentialPitzDaily

  Solver potentialFoam

  Description Demonstrates the use of `groovyB` with `potentialFoam` (also a problem connected with that). Provided by Martin Backer

5.3.2 FunkySetFields

Example dictionary for `funkySetFields`

5.3.3 FunkySetBoundaryFields

Example dictionary for `funkySetBoundaryFields`. Sets nonsense boundary conditions for the world famous `damBreak`-case

5.3.4 InterFoamWithSources

Demonstrates usage of `expressionSource`

Due to differences in the original `interFoam`-solver this doesn't work on certain OpenFOAM-versions (most specifically `1.6-ext`). The current solver works with 2.1. For older OF-versions use the sources labeled `_pre2.1`.

The only modifications to the original solver are found at the end of `createFields.H` and in `UEqn.H` (the added source terms).

5.3.5 InterFoamWithFixed

Demonstrates usage of `forceEquation`

Due to differences in the original `interFoam`-solver this doesn't work on certain OpenFOAM-versions (most specifically `1.6-ext`). The current solver works with 2.1. For older OF-versions use the sources labeled `_pre2.1`.

The only modifications to the original solver are found at the end of `createFields.H` and in `UEqn.H` (the fixing of the velocities).

- `interFoamWithSources`
  Slightly modified version of `interFoam`. Adds a source term to the momentum equation. The source term is an expression that is defined at run-time

- `mixingThing`
  Demonstration case for it.

  Preparation Run the script `prepare.sh` to prepare the case

5.3.6 FiniteArea

Demonstration of the `finiteArea`-stuff that works with 1.6-ext

- `swakSurfactantFoam`
  Variation of `surfactantFoam` that adds an `expressionSource`

- `planeTransport`
  Demonstration case

  Preparation Use `blockMesh` and `makeFaMesh`

  Solver `surfactantFoam` (without source term) or `swakSurfactantFoam`

  Demonstrates FAM-specific `swakExpressions` and `groovyBC` (as well as the `expressionSource`)

5.3.7 other

Cases that don't have a `groovyBC`

- `angledDuctImplicit`

  Solver rhoPorousMRFSimpleFoam

  Mesh preparation Execute the `makeMesh.sh`-script in that directory. If you want to run in parallel call the `decomposeMesh.sh`-script with the number of processors as an argument

Demonstrates Usage of the `swakTopoSources`. Compares different approaches to evaluating with the `swakExpression`-functionObject. Also an example dictionary that demonstrates the use of `funkyDoCalc`

- `angledDuctImplicitTransient`

  Solver rhoPorousMRFPimpleFoam

  Mesh preparation Execute the `makeMesh.sh`-script in that directory. If you want to run in parallel call the `decomposeMesh.sh`-script with the number of processors as an argument

  Demonstrates The same as `angledDuctImplicit` but also the output of temporal changes

- `capillaryRise`

  Solver interFoam

  Case preparation run the supplied script `prepareCase.sh`

  Demonstrates Usage of a sampled surface to track the interface in a VOF-simulation

- `mixingGam`

  Solver interFoam

  Case preparation run the supplied script `prepareCase.sh`

  Demonstrates Emulate a "moving gravitation" by using the `manipulateField`-functionObject to recalculate `gh` and `ghf`

5.3.8 PythonIntegration

Demonstrate the integration of `Python`. Mostly using `PyFoam` but also with `pythonFlu`

- `manipulatedPitzDaily`

  Solver simpleFoam

  Demonstrates Usage of `PyFoam` to manipulate the `fvSolution`-file during the run (possible application: unphysical initial conditions cause the run to fail during startup with "normal" relaxation values)

- `findPointPitzDaily`

Solver simpleFoam

Demonstrates Usage of the `pythonFlu`-integration to find the point where the recirculation behind the step ends. Also tries to plot the result using the `matplotlib`-library

- `bed2OfPisa`

  Solver twoPhaseEulerFoam

  Demonstrates Usage of `PyFoam` to read the direction of gravity and feeding it into a `goovyBC` via global variables

  Case preparation Just call `funkySetFields -time 0`

- `multiRegionHeaterBuildFunctionObjects`

  Solver chtMultiRegionFoam

  Demonstrates Building the specification of function objects at runtime via a Python-script

5.3.9 `CodeStream`

Demonstrates working together with the `coded`-stuff in OpenFOAM 2.0

5.3.10 solvePDE

Examples for the `functionObjects` that can solve *Partial Differential equations*

- flangeWithPDE

  Solver laplacianFoam

  Demonstrates The usage of the `functionObject` that solves the laplacian (Poisson) equation and (hopefully) that it gets the same result as the native solver

  Case preparation `Allrun`-script is provided

- pitzDailyWithPDE

  Solver scalarTransportFoam

  Demonstrates Solving additional transport equations

5.3.11 BasicSourceSubclasses

These examples test the source terms based on `basicSource`. They only work with OpenFOAM 2.x and all use the `simpleFoam`-solver

- pitzDailyWithSwirl

  Demonstrates Fixing the values of the velocity in a region with `SwakSetValues`

- pitzDailyWithExplicitPoroPlug

  Demonstrates Implementing a simple porous plug by adding the Darcy-term as a source term with `SwakExplicitSource`

- pitzDailyWithImplicitPoroPlug

  Demonstrates Same as `pitzDailyWithExplicitPoroPlug` but with an implicit source term with `SwakImplicitSource`

5.3.12 Lagrangian

Tests for the functionObjects that create and evolve a cloud of particles

- hotStream

  Solver replayTransientBC

  Mesh preparation `prepareCase.sh`-script

  Demonstrates 3 clouds (kinematic, reacting, solidParticle). Loading of a thermophysical model with a functionObject. Plugin functions for information about the clouds

- angledDuctWithBalls

  Solver rhoPimpleFoam

  Demonstrates Thermo-cloud. Functions for lagrangian particles

5.3.13 tests

Simple test cases for specific features

- `randomCavity`
  Testing of different seeds for the `rand`-function. Also tests the `randFixed`-function

5.4 maintainanceScripts

Undocumented scripts used for maintaining swak4Foam. **If you don't understand them, don't use them**

6 Bug reporting and Development

6.1 Bug reports

The preferred place for bug reports is http://sourceforge.net/apps/mantisbt/openfoam-extend/search.php?project_id=10&sticky_issues=on&sortby=last_updated&dir=DESC&hide_status

A sourceforge-account is required for reporting

6.1.1 Things to do before reporting bug

If you're reporting a bug about the compilation please run `Allwmake` **twice** and only report the messages from the second run. This makes analyzing the log easier as only the unsuccessful commands will be reported.

If the problem seems to be a missing library rerun the compilation to make sure that there wasn't a problem with that.

6.2 Development

Contributions to to swak4Foam are most welcome. If you want to contribute clone the Mercurial archive of the sources

```
hg clone http://openfoam-extend.hg.sourceforge.net:8000/hgroot/openfoam-extend/swak4Fo
```

Change to the branch that you want to improve (usually `default`) and create a new branch

```
hg branch <branchName>
```

where `<branchname>` is an easily identifiable name that makes the purpose of the branch clear (for instance `bugfixWrongRandomFunction` or `featureHyperbolicFunctions`). Don't work on the `default` branch or any other branches that are not "yours". Such contributions will not be merged

Once development on the branch is finished export the relevant changesets with

```
hg export <nodeID>
```

(`nodeID` being the ids of "your" changesets) and send them to the maintainer (or attach them to a bug report on Manits). The changes will be reviewed and merged into the `default` branch (do not attempt to do this yourself). Patches generated with `hg export` make sure that all changes are attributed to the original developer (you).

An alternative would be the `bundle` command. Just do

```
hg bundle <bundlefile>
```

and then send the `bundlefile`. This will include **all** commits that are not in the upstream repository and will allow similar inclusion in the upstream as `export`.

Once you have proven by successfully submitting changesets via `hg export` you can ask for write access to the mercurial repository.

6.2.1 Suggest reading

These topics may be "new" for the average OF-developer:

Mercurial A short tutorial on this can be found at http://mercurial.selenic.com/guide/. If you already know `git` the http://mercurial.selenic.com/wiki/GitConcepts may be enough for you

bison/flex This pair of compiler generator tools generate the parsers for the expressions. Google for a tutorial that looks promising to you.

For a short example that shows how a new function was added to two parsers have a look at this changeset that added the `cpu()`-function to the field and the the patch-parser (usually you'll have to write a new method for the driver too):

```
hg diff -c 8604e865cce6
```

6.2.2 Special branches

Currently the main branches are:

default The **main** branch. This is the brancht that the general public will receive. It compiles under 1.7 and 1.6-ext

`port_2.0.x` The branch that compiles under OpenFOAM 2.0. This will eventually become the `default`-branch

debianPackaging Branch for generating new Debian-packages of `swak4Foam`. If somebody wants to "inherit" this: contact the maintainer

finiteArea In this branch the things for the `finiteArea`-discretization (only present in `1.6-ext`) is developed. Usually gets merged back into the `default`-branch once a feature is completed

6.2.3 Distributed bug-tracking

As an experimental feature distributed bug-tracking was introduced using the *Artemis*-extension for *Mercurial* (see `http://hg.mrzv.org/Artemis/`). An up-to-date version can be installed by

```
hg clone http://hg.mrzv.org/Artemis/
```

somewhere and installing the plugin by editing `.hgrc`.

This is **not** the official bug-tracker for `swak4Foam`. It is used for keeping track of new features that are to be introduced to `swak4Foam` and may be discontinued if the experiment proves to be unsuccessful.

7 Copyright

8 Known bugs

The following list is not complete. If the Artemis extension (see above) is installed then

```
hg ilist
```

gives a more up-to-date list

8.1 Moving meshes and `sampledSurfaces`

It seems that with moving meshes `sampledSurfaces` don't get updated. This seems to be a problem with OpenFOAM itself (the regular `surfaces`-functionObject doesn't get updated. This is currently investigated

8.2 Missing support for interpolation and point-Fields

Apart from patches and internal fields the support for interpolation from cells to faces (and vice versa) is incomplete as well as point fields (although they are supported in the grammar)

8.3 Caching of loaded fields not working

This is especially evident for the `funkyDoCalc`-example

8.4 Possible enhancements of the code

Not really bugs, but stuff that bugs me

8.4.1 Pointers in the driver code

This is necessary because of `bison`. Investigate possibilities to replace these by `tmp` and `autoPtr`

8.5 Possible memory loss

`valgrind` reports some lost memory for stuff that is not directly allocated by `swak4Foam` (in OpenFOAM-sources)

Will investigate. Relevant places are marked by comments in the code. Also the construction of `sampledSet` seems to loose memory

8.6 Non-treatment of the inner product `&` of symmetric tensors

Before OpenFOAM 2.1 the inner product of two symmetric tensors was a symmetric tensor. Since 2.1 it is a general tensor. As the general treatment in the grammar would be confusing currently the this product was removed from the grammar and therefor will not be correctly parsed

8.7 No point-vector construction for Subsets

The same problem that was mentioned in https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=130 is also true for subsets. But as the interpolation is not implemented for most subsets this will be postponed

9 History

9.1 2010-09-13 - version number : 0.1

First Release

9.2 2010-12-18 - version number : 0.1.1

New release Most important changes

9.2.1 Parser for `sampledSurfaces`

Now expressions for the field on a sampled surface can be evaluated. All `sampledSurfaces` offered by OpenFOAM now can be used

9.2.2 Multiline `variables`

The `variables` entry (most notably used in `groovyBC` and `swakExpression`) now can be a list of strings. This allows some kind of "formatting" (one expression per line) and should improve readability

9.2.3 Two maintainance-scripts were added

These can copy the libraries and utilities to the global installation (for those who think that the swak4Foam-stuff is stable enough and want to 'bless' all users at their site with it). Note that any local installation still takes precedence (because `$FOAM_USER_APPBIN` is before `$FOAM_APPBIN` in the `$PATH`

9.2.4 Parsers using 'external variables' are now run-time selectable

This allows the inclusion of other parsers with the regular `swak4Foam` parsers and include them seamlessly with the `variables`-mechanism for 'externals' (in other words: you can add your own parser in a separate library without having to change anything about the overall `swak4Foam`, but it behaves as if it was part of it)

9.3 2011-01-30 - version number : 0.1.2

9.3.1 Support for *Finite Area*-stuff

Now there is support for the `finiteArea`-library found in 1.6-dev. The support is found in a separate library `swakFiniteArea`. It has

- a parser `faField` for `areaFields`
- a parser `faPatch` for patches of `areaFields`
- a variant of `groovyBC` for these patches
- a computed source `faExpressionSource`
- Function-object-variants for `areaFields`: `clearExpression`, `expressionField` and `manipulateField`. These work the same as their `volField`-counterparts

9.3.2 Bugfix for compiling in single precision

See https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=49

9.3.3 New function `nearDist`

See https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=44

9.4 2011-04-20 - version number : 0.1.3

New features and changes are (this list is not complete):

9.4.1 New utility `funkySetAreaField`

Like `funkySetFields` for *finiteArea*. Also writes out a volume field for easier post-processing

9.4.2 Refactoring of `groovyBC` and groovified boundary conditions

Makes it easier to use the `groovyBC`-machinery for other boundary conditions. Two standard boundary conditions were *groovified*. Others may follow

9.4.3 Easier deployment

If no `simpleFunctionObjects` are present they can be downloaded by a script. Also scripts to handle global installations of `swak4Foam`

9.4.4 Force equations

A class to force equations to certain values according to expressions

9.4.5 New utility `funkyDoCalc`

Utility does calculations on saved results and outputs single numbers (min, max, sum, average) to the terminal. Can be used for reporting or validity checks

9.4.6 Debian packaging

Crude packaging for Debian

9.4.7 Lookup-tables

A single-argument function can be specified as a piecewise linear function. Basically works like timelines but the argument can be something else (not only the time)

9.4.8 Stored variables

Variables that store their values between time-steps. Applications are statistics or switches

9.4.9 Sampled sets

Sampled sets can now also be used as en entity on which calculation is possible.

9.5 2011-07-26 - version number : 0.1.4

9.5.1 Port to OpenFOAM 2.0

This is the first release that officially supports OpenFOAM 2.0

Also it is the first release that incorporates the `simpleFunctionObjects`-library

9.5.2 New features:

- Rewrite of `rand` and `randNormal`
  - These two functions now can receive an integer seed that determines the pseudo-randooom sequence generated by these functions

  - Two functions `randFixed` and `randNormalFixed` were added. While the usual `rand` functions generate a different result at every time-steps for these functions the pseudo-random sequence is determined **only** by the seed (not by the timestep)

- Binary `min` and `max`
  Take the bigger/smaller of two fields. Helps avoid `?:`-operations
- Allow writing of only volume-fields in `funkySetAreaFields`
  Application: If the results of the calculation are only needed in ParaView
- Internal changes
  - Use `autoPtr` for sets
  - Update sets that change in memory or on disc

9.5.3 Bug-fixes

- `funkySetAreaFields` did not check for the correct fields
  Fixed by Petr Vita
- `surfaceProxy` uses the wrong geometric data
- Avoid floating point exceptions with division of fields
  Calculated boundaries were 0 and caused a division by zero

9.5.4 Packaging

- Update Debian packaging
  - Packaging information for the currently used OF-version is generated (allows a separate `swak`-package for every OF-version
  - Submission to launchpad added
- Deployment scripts
  Now install to `FOAM_SITE_APPBIN/LIBBIN`

9.6 2011-10-03 - version number : 0.1.5

9.6.1 New features

- `replayTransientBC` now supports multiple regions
  Uses the usual `-region`-option. Looks for a separate dictionary in the `system`-directory of that region

- `replayTransientBC` allows execution of functionObjects
  This can be switched on using the `allowFunctionObjects`-option

- `Python`-embedding
  Allows the execution of Python-Code in a functionObject

  This feature is still experimental and the organization of the libraries is subject to change

- Global variables
  It is now possible to define variables that are 'global': They can be read in any entity.

  Currently these variables can only be uniform.

  To access global variables the specification-dictionary has to have a `wordList` named `globalScopes`. The scopes are searched in that order for the names of global variables. Having scopes allows some kind of separation of the variables

- Using OF 2.0 codeStreams
  Adds a functionObject `swakCoded` that extends the `coded`-functionObject to read and write global variables

- Simplified boundary condition `groovyBCFixedValue`
  Added a boundary condition than allows to only fix the values. This should help to avoid problems with cases that don't like `mixed` (on which the regular `groovyBC` is based)

- Function objects to solve PDEs
  Two function objects that solve *Partial Differential Equations* during a run have been added:

  - one that solves a laplacian (Poisson) equation
  - one that solves the transport equation for a scalar

  The relevant coefficients (including explicit and implicit source terms) can be specified using expressions

9.6.2 Administrative and packaging

- Inject `swak4Foam` into a distro
  Added a script that takes the current sources, copies them into the appropriate places of a `OpenFOAM`-installation and slightly rewrites them

to compile in this place. What happens then (committing them into the repository or just plain compilation) is up to the maintainer

- Absorb `simpleFunctionObjects`
  As many parts of `swak4Foam` depend on it the `simpleFunctionObjects` have now been absorbed into `swak4Foam`. They can still be compiled on their own

9.6.3 Bugfixes

- Variables not working for parallel computations
  If a variable is defined and the patch which it is defined on doesn't have any faces the variable is reported on that processor as *not existing* and the whole run fails

9.7 2012-01-04 - version number : 0.1.6

9.7.1 Cases changed

- `circulatingSplash`
  Fixed according to a suggestion by Andreas Otto. Now runs again (used to crash some time-steps into the beginning)

9.7.2 Infrastructure

- Check for correct `bison`-version
  The `Allwmake`-script now checks for the correct bison-version (and the existence of `bison`) and fails if it doesn't seem to be the right one

- Supply a header with preprocessor-symbols about the used OF-version
  To allow distinguishing different OF-versions as discussed in the bug report http://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=114 the `Allwmake`-script now generates a header file `foamVersion4swak.H` that defines the symbols `FOAM_VERSION4SWAK_MAJOR`, `FOAM_VERSION4SWAK_MINOR` and `FOAM_VERSION4SWAK_PATCH`

9.7.3 Technical

- Refactoring of the FieldDriver
  - now can also produce surfaceFields
  - full support of `tensor`, `symmTensor` and `sphericalTensor`

- Refactoring of the FaFieldDriver
  - now can also produce edgeFields

  No support for tensors yet

- Writing of `storedVariables`
  If necessary (for instance `swakExpression`-functionObject) the `storedVariables` are written to disc (into a subdirectory `swak4Foam` of the timestep) and are reread at the start. This allows consistent restarts (for instance if a flow was summed using the variable) **if the expressions stay the same**.

- `simpleFunctionObjects` now write vectors and tensors without brackets
  Data files can now be written without brackets but each component on its own. The number of entries in the header is not adjusted

- A *default mesh* for the drivers exists
  For drivers that don't have access to a `fvMesh` a default mesh exists. This default mesh is defined by the first `fvMesh` that is used during the construction of **any** driver.

  Definition of the default mesh can be forced using the `initSwakFunctionObject` (see the test case `flowRateAngledDuct`)

9.7.4 New features

- General `phi` in `solveTransportPDE`
  Due to the refactoring of the `FieldDriver` now `phi` can be specified by a general expression (instead of 'only' a field-name)

- `funkySetFields` now also writes surfaceFields
  Can write `surfaceVector` and `surfaceScalar`-Fields. Condition has to be consistent

- Function objects now `surfaceField`-aware
  `expressionField` and `manipulateField` now can create or modify `surfaceFields`

- `funkySetFields` and function objects support tensors
  `funkySetFields` and the function objects `expressionField` and `manipulateField` now also work with the three tensor-types

- Extension of the `expressionToFace` `topoSet`
  If the expression evaluates to a `surfaceField` then this is used as a flag whether or not the face is in the `faceSet`. If the expression evaluates to a `volScalarField` then the old semantic applies (faces are in the set if one cell is `true` and the other is `false`).

  This only works for internal faces

- `addGlobalVariable` allows setting more than one value
  If there is an entry `globalVariables` then this dictionary is used to set the variables

- Function object `calculateGlobalVariables`
  Calculates variables and then pushes them to a global namespace

- Generate a dummy `phi` in `replayTransientBC`
  New option added that generates a `phi` field with value 0 to keep boundary conditions like `inletOutlet` happy

- Function object to dump expression results
  The functionObject `dumpSwakExpression` dumps the complete results of a swakExpression to file at each timestep. This produces huge files and is therefor not endorsed

- Additional options for `funkySetFields`
  Add the options `allowFunctionObjects` and `addDummyPhi` to execute functionObjects and add a `phi`-field (for fields that require these)

- Boundary condition `groovcBCDirection`
  Based on the `directionMixed` boundary condition this allows to set a boundary condition as a Dirichlet-condition only in certain directions while in the other directions it is a gradient-condition

- Boundary condition `groovyBCJump`
  Boundary condition that imposes a jump in the value on a cyclic boundary condition pair (based on `jumpCyclic`). Only works for scalar values

- `simpleFunctionObjects` write CSV-files
  Setting the option `outputFileMode` to `csv` writes CSV-files. The option-value `foam` is the default (old style). The option-value `raw` writes the values delimited by spaces (no brackets for vectors and tensors)

- Submeshes automatically read if `searchOnDisc` specified
  If a submesh is not yet in memory and the option `searchOnDisc` is set, the mesh is automatically read into memory and kept there

- Conditional `functionObjects`
  The `simpleFunctionObjects`-library now has a number of functionObjects that allow the conditional execution of a list of function objects.

  These are

  executeIfExecutableFits if the name of the executable fits a regular expression the function objects are executed

  executeIfObjectExists if a named object exists (or alternatively: doesn't exist) in the registry execute the function objects. Type checking also implemented

  executeIfEnvironmentVariable execute if an environment variable satisfies a certain condition (exists, doesn't exist, fits a regular expression)

  executeIfFunctionObjectPresent execute if a specific functionObject is present. This can help prevent failures if a functionObject is missing for technical reasons

  In addition the `simpleSwakFunctionObjects`-library has

  executeIfSwakObject Evaluates a logical swak-expression. The results are either accumulated using logical *or* (if **one** value is true the result will be true) or logical *and* (all values have to be true)

  The `pythonIntegration`-library has

  executeIfPython Evaluates a Python-code-snipplet that returns a value. If this value is "true" in Pythons standards then the functionObjects are executed

- `functionObject` that reads gravitation
  `simpleFunctionObjects` has an additional function object that reads the direction of gravitation. The purpose is to assist boundary conditions like `buoyantPressure` that rely on it to work. Best used together with conditional function objects ("If `g` is missing ....")

- PDE-functionObjects for `finiteArea`
  Solve transport and laplacian equation

- Subclass to `DataEntry` that uses *swak*-expressions
  This is defined in the `swak4FoamParsers`-library. The class needs a default mesh defined to construct the driver. Definition of the default mesh (if no other driver was constructed in some function-object or by a `groovyBC`) can be forced using the `initSwakFunctionObject` (see the test case `flowRateAngledDuct`)

- `funkySetAreaField` now also writes edgeFields
  Similar to the `surfaceFields` in `funkySetFields`

9.7.5 Bug fixes

- Compilation with Intel-Compiler possible
  The `Utilities` failed with the Intel-compiler. Compilation now falls back to good old `g++`

- Access to tensor-components not working
  Because the tokens were not defined in the `flex`-files getting tensor components with `tensor.xx` did not work. Fixed

- Constants for surfaceFields not working
  Because surfaceFields know no `zeroGradient` the template `makeConstant` did not work

- `snGrad` does not work for patches if the file is on disc
  Change so that the field gets temporarily loaded to calculate the gradient on the patch. Same for `internalField` and `neighbourField`

- `potentialFoam` does not correctly use a `groovyBC`
  The reason is that `groovyBC` usually doesn't get evaluated during construction. The reason is that it is hard to tell whether all required fields are already in memory. The current fix is a workaround: setting `evaluateDuringConstruction` to `true` forces the BC to be evaluated during construction

- Extra evaluation of boundary conditions causes failure
  Extra evaluation of boundary condition that should fix the problem with `calculated` patches causes `funkySetFields` to fail with stock boundary conditions if not all fields are present in memory

9.7.6 Discontinued features

- `groovyFlowRateInletVelocity`
  This boundary condition will be removed in future releases because the base class now supports the more general `DataEntry`-class for which a `swak`-subclass exists

9.8 2012-04-13 - version number : 0.2.0 Friday the 13th

9.8.1 New features

- Region name in `simpleFunctionObject`.outputs
  To distinguish the output of various instances of `functionObjects` from the `simpleFunctionObjects`-library in multi-region cases the screen-output is prefixed with the region name. For the default region nothing changes. Directory names stay the same as they are unambiguous anyway (they contain the name of the functionObject)

- Temporal operators `ddt` and `oldTime`
  For fields (not expressions!) the value at a previous timestep can be gotten via `oldTime(field)` if that information exists (also for `funkySetFields` if the corresponding file `field_0` exists.

  For fields that support it (basically volume-fields) there is also a `ddt`-operator that calculates the explicit time-derivative (if information about the last timestep exists)

  Currently implemented for

  internalFields `oldTime` and `ddt`

  patch only `oldTime`

  cellSet,cellZone only `oldTime`

  sampledSurface,sampledSet only `oldTime`

  faceSet,faceZone `oldTime`

  internalFaFields `oldTime` and `ddt`

  faPatch only `oldTime`

  If there is no old time value stored and in the parser dictionary the parameter `prevIterIsOldTime` is set, then the previous iteration value is used as the old time.

- Boundary condition `groovcBCDirection`
  Based on the `directionMixed` boundary condition this allows to set a boundary condition as a Dirichlet-condition only in certain directions while in the other directions it is a gradient-condition

  Note: this should have been in the last release but was forgotten to merge into the default branch

- Boundary condition `groovyBCJump`
  Boundary condition that imposes a jump in the value on a cyclic boundary condition pair (based on `jumpCyclic`). Only works for scalar values

  Note: this should have been in the last release but was forgotten to merge into the default branch

- Function to return the position of minimum and maximum
  The functions `minPosition` and `maxPosition` return the position of the minimum or the maximum of a scalar field

  This is implemented for all field types

- Support for `pointFields` in the field-parsers
  Now can read and generate pointFields.

  Detailed features (apart from the standard symbols) are:

  - Function `point` generates a constant `pointScalarField`
  - Function `pts()` returns a `pointVectorField` with the point positions
  - Functions `pzone` and `pset` generate logical fields according to existing pointZones or pointSets
  - Functions `interpolateToCell` and `interpolateToPoint` interpolate from pointFields to volFields and from volFields to pointFields

  Utilities and functionObjects affected by this are

  - `funkySetFields`
  - new topoSource `expressionToPoint`
  - `expressionField` and `manipulateField` now can deal with pointFields

- Support for tensors in the `finiteArea`-field parser
  The `faField`-parser now supports tensors, symmetric tensors and spherical tensors.

  Not all operators are supported because the are not instantiated in 1.6-ext

- New convenience-variables for the Python-Integration
  These variables are added in the namespace to ease the writing of Python-code whose output is consistent with OF

  timeName Name of the current time as a string. Allows the construction of directory names

  outputTime Boolean that indicates whether this is a timestep where OpenFOAM will write output

- Additional operators from the `fvc`-namespace
  The missing operators from the `fvc`-namespace have been added to the Field-parser. These are

  `d2dt2` for all volumeFields

  `flux` for all volumFields. Needs a surfaceField as a first argument

  `meshPhi` for volume-vector-fields. Optional with a scalar-field that acts as the density as the first argument. Only works in the context of a solver with a dynamic mesh and hasn't been tested yet

  The only missing operators from the `fvc`-namespace are `volumeIntegrate=/=domainIntegrate`. These have been omitted as they are trivial to implement using other functions

9.8.2 Infrastructure

- Full parallel compilation
  Thanks to patches supplied by Bruno Santos (see http://sourceforge.net/apps/mantisbt/openfoam extend/view.php?id=105) compilation of the libraries is now possible in parallel

- Version numbers and version number reporting
  Releases up to now got a version number. Utilities now report the version number. This should make it easier to find out whether problems are due to an old version

  Still looking for a way to do that for the libraries (so that they will report it if dynamically loaded)

9.8.3 Packaging

- Update of the Debian-Packaging
  `make dpkg` now genrates also a valid package if the current OpenFOAM-installation is **not** installed using the `dpkg`.

9.8.4 Changes in the behavior

- Directory variables in Python-integration
  For parallel runs the content of the `caseDir`-variable changed and a few variables have been added

  caseDir in parallel runs now points to the `FOAM_CASE` instead of the processor subdirectory

  systemDir points to the global `system`-directory

  constantDir points to the global `constant`-directory

  procDir in parallel runs points to the processor-subdirectory of the current CPU

  meshDir The mesh data (of the current processor in parallel runs)

  timeDir Directory where data would be written to at the current time (processor dependent)

- User must acknowledge parallelization in Python-integration
  In parallel runs the user must set the `isParallelized` to `true` if the `parallelMasterOnly` is set to `false`.

  With that he indicates that in his opinion the Python-code has no bad side-effects in parallel runs and that he doesn't blame swak4Foam if anyting bad happens

9.8.5 Bug fixes

- `interFoam`-based example solvers do not compile on 2.1
  As reported in https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=119 due to a change the way the PISO-loop is treated the `interFoamWithSources` and `interFoamWithFixed` don't compile with 2.1 anymore.

  To avoid `#ifdef` in the solver sources there is now a separate set of sources (labeled `pre2.1`) for older versions. The regular sources work with 2.1 (and hopefully the following)

- `-allowFunctionObjects`-option not working for `replayTransientBC`
  Function-objects only work with the `while(runTime.loop())`-construct in 2.1. The utility now uses this.

- Field itself can not be used in `funkySetBoundaryField`
  Bug reported: http://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=124

  An expression like `2*U` did not work for the field `U`. Reason was that the registry already held an object called `U` (the dictionary representation of the field) and therefor refused to load/register another `U`.

  Has been fixed by de-registering the dictionary `U` immediately after loading.

- No gradient for vectors in FieldParser
  The gradient for a vector field (result: a tensor field) was not calculated. It is now part of the grammar

- Some operators for tensors not working in 1.6-ext
  `tr`, `inv` and `det` were not working for some tensort types in 1.6-ext. The parser now fails if such a combination is used. Works OK for other OF-versions

  Also introduced a workaround for certain operators not being properly defined for pointFields (by using the internal fields)

- `x`, `y`, `z` and `xy` etc not available as field names
  These symbols could not be used as field names because they were used for the components of vectors and tensors

  Now these names are only used if the `.`-operator asks for a component. This is implemented for these parsers

  - FieldValues
  - PatchValues
  - SubsetValues (all Zones, sets and samples)
  - finiteArea-Parsers: faPatch and faField

- Missing tensor components for point-Fields in some parsers
  All parsers except the field-parser were missing the access to tensor components in the grammar

- No vector construction possible for point-vectors (and tensors) in PatchField

As mentioned in https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=130 it was not possible to construct a point-vector field using `vector(toPoint(1),toPoint(1),toPoint(1))`. Same for tensors

- Incomprehensible error message in `funkySetFields` if the field is missing
  The error message in `funkySetFields` that was issued when a field is supposed to be created was not very helpful (something about the field currently being an `IOobject`)

- Missing `magSqr` in parsers
  This function was reported missing on the message board

- Wrong size when composing vectors and tensors for point-fields
  The composed objects got their size from the current parser, not the components. This resulted in a segmentation-fault for pointFields

- `icc` does not compile `executeIfExecutableFitsFunctionObject` on Linux
  Preprocessor symbol `linux` unknown. Replaced with `__linux__`

- Enhancement to the `trackDictionary`-functionObject
  Now handles bad or non-existent filenames for dictionaries to track

  Fix provided by Martin Beaudoin

9.9 2012-10-18 - version number : 0.2.1

9.9.1 Requirements

- `flex 2.5.35`
  This version is needed for the reentrant parsers. `2.5.33` **may** work but is untested. Version `2.5.4` which is on some old systems definitely does not work

- `bison 2.4`
  Version 2.3 compiles but there is an offset-problem with the locations that breaks the Plugin-functionality

  Mac-users will have to install `bison` from another source (for instance `MacPorts`)

9.9.2 Bug fixes

- Make sure that `Allwmake` always uses the `bash`
  On Ubuntu `/bin/sh` is something else and the scripts fail. Hardcode to `/bin/bash`

- `downloadSimpleFunctionObjects.sh` still in Makefile
  This script (which was removed long ago) was still referenced in the `Makefile`.

- `grad` in fields added dimensions
  `grad` and other operators from `fvc` added dimensions to values that were supposed to be dimensionless. This has been fixed

- Default `condition` for surface fields wrong size in `funkySetFields`
  Due to a typo the constructed condition field was too short for surface-fields (too long for volume-fields, but that didn't matter)

- `mappedFvPatch` not treated like regular patches
  The field-driver created patch fields there as `calcuated` when `zeroGradient` would have been more appropriate

- `flip()` for `faceSet` not correctly calculated
  A `SortableList` was used which meant that the vector with the flip values was not in the correct order

- `fset()` crashes when `faceSet` has boundary faces
  This problem was due to a non-special treatment of faces on the boundary. Fixed (for `faceZone` as well).

  Also boundary faces are now honored in `expressionToFace` if the expression is a surface-field (for the volume-field logic boundary faces will never work)

- `groovyBC` produced incorrect results with `wallHeatFlux` etc
  The reason was that during construction `refGradient`, `refValue` etc were not read correctly (if present).

  This is now fixed in `groovyBC` and the other BCs (`groovyBC` for pointFields and `groovyBCDirection`)

- Global variables not found in `faField`
  The Lexer correctly identified the variable but the `getField` method did not know how to get it.

  Fixed

- Wrong type of condition field in `funkySetAreaFields`
  If no condition was evaluated the utility generated a pseudo-field of the wrong length

- `calculated`-patches 0 for some operations
  For some operations in the `Field`-driver the `calculated`-patches had the value 0 instead of the number of the next cell. This caused certain calculations to fail (give 0)

  The `FaField`-driver did no fixing of the `calculated`-patches at all.

  This is fixed

- `sqr` of a `vector` should produce a `symmTensor`
  Reported in http://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=150

  Added the `sqr` at the right place to the grammars. Also some other missing tensor operations (`dev` and `symm`).

- `funkySetFields` produced wrong values on processor patches
  Patch fields were not copied any no `correctBoundaryField` was called for technical reasons.

  Fix: values copied by hand

- `sortedToc` does not exist for `1.6-ext`
  Introduced a preprocessor symbol that allows using sortedToc on newer versions

- Wrong `size()` and `pointSize()` reported for `FaField`-driver
  This was due to a strange () (still don't know what happened there)

- Memory leak in the field drivers
  The strings of parsed IDs were not properly deleted. Funnily this was done correctly in the Patch and the Subset-driver. Also for timelines-

  Also fixed a leak with the labels of plugin-functions that was present with all drivers

- Maintenance scripts not working with non-=bash= `/bin/sh`
  Reported by Oliver Krueger that on systems where `/bin/sh` is not a `bash` anymore (newer Ubuntu and SuSE) the sourcing of `theFiles.sh` doesn't work anymore.

  Fixed and moved all the files to the `maintainanceScripts`-folder

- `cof` and `diag` undefined
  Added. Diag had to be reprogrammed as it is not implemented for fields (probably for performance reasons).

  Also some tensor operators were missing (probably lost during copy/-paste)

- No new file created if number of patches for patchAverage changes
  Reported in https://sourceforge.net/apps/mantisbt/openfoam-extend/view.php?id=153

  Fixed by removing all file pointers if the number of patches changes

- `variables` intolerant on spaces
  Spaces in the list of variables made the reading fail because words can't contain spaces. For instance

  ```
  "var =T*2;"
  ```

  Now all the spaces are removed before assigning to variables. This will also remove spaces on the "inside" thus making

  ```
  "v ar =T*2;"
  ```

  the same as the above expression. But it is unlikely that the call will be used in this way

- Missing `div`-operations
  These valid `div`-operations were missing from the grammar:

  - Divergence of a volume-tensor (all three kinds) producing a vector
  - Divergence of a surface-tensor (all three kinds) producing a volume-tensor

- Fields created by `expressionField` written too late
  Fields created by that function object were written with the value from the timestep before because the regular write occurs before the execution of the function objects.

  Fixed

- `storedVariables` did not require a `initialValue`
  Now an initial value is required (instead of the default empty string which caused parser failure)

- Dimension checking makes `expressionField` fail
  Reason is that during the calculation of the variables dimensions are checked.

  Now the functionObject switches the checking off. But a more general solution is desirable

- `expressionField` at last timestep not written
  The functionObject does not write (and calculate) the field at the last timestep.

  Fixed with an one-liner

- `groovyBC` makes `interFoam`-runs fail unpredictably
  Reason was an uninitialized `falueFraction` which sometimes has values that cause a floating point exception. Fixed

- Global variables of different sizes break parallel runs
  Because `size()` was equal to the expected size on some processors. Not on all. Now the branch is taken if the size is equal on **all** processors

- Fields treated with `readAndUpdateFields` were written one timestep too late
  Fields were not written after the boundary condition was updated. Now they are

9.9.3 Enhancements

- Topology-operators now support `variables` etc
  The topology operators `expressionToCell`, `expressionToFace` and `expressionToPoint` now support `variables` and the other supporting keywords if being constructed with a dictionary (for instance from the `topoSet`-utility)

- Fields touched by `manipulateField` being written
  Usually the manipulated version of the fields is not written as the manipulation happens **after** writing. The option `writeManipulated` enforces writing.

  Writing is not the default behavior to avoid side-effects

- Indicator functions `onPatch` and `internalFace` added to field-expressions
  The function `onPatch(name)` returns a surface-field that is 1 on all faces that belong to patch `name`.

The function `internalFace()` is 1 on all internal faces and 0 on all patches

- Non-uniform second argument for `pow`
  Now the second argument to the `pow`-function can be a non-constant

- Added transpose to the tensors
  The expression `A.T()` transposes the tensor `A` (for symmetrical and spherical tensors it leaves them untouched)

- Added unit tensor `I` to parsers
  If no field `I` is defined then this is used as the unit-tensor

- Added the *Hodge dual* operator
  The unary operator `*` calculates for tensors and symmetrical tensors the hodge dual

- `replayTransientBC` can now switch on function-objects via dictionary
  The optional entry `useFunctionObjects` switches on the execution of function objects during the calculation

- `replayTransientBC` can now create a `phi`-field via dictionary
  The optional entry `addDummyPhi` creates a `phi`-field

- `expressionField` now allows the specification of a dimension
  The `dimensions`-entry is now read at the same time the variables are read (this should work for all programs/functionObjects where the parser is not constructed using a dictionary but the dictionary is later searched for the `variables`-entry)

9.9.4 New features

- Allow dynamically loaded plugins that add new functions to parsers
  This allows easy extension of the functionality of `swak4Foam` without modifying the grammar files.

  The way it works is that new functions are added to a runtime-selection table. If the grammar can not resolve a symbol as a built-in function or a field (but only then) it looks up the name in this table and evaluates the function. Parameters are parsed separately and can be:

  primitive data types integer, float, string and word

swak-expression an expression parsed by one of the swak-parsers. The type of this expression does not necessarily have to be the same as the one of the 'main' expression.

The first time a plugin function is searched `swak4Foam` prints a list of all the available functions of this type. Information included is the return type and the types of the parameters (these include the parser used, the expected type and a parameter name).

Libraries with plugin-functions are added via the `libs`-entry in the `system/controlDict`

A number of plugin-libraries are already included covering these topics:

- Evaluation of functions of the turbulence, transport or thermo model
- Different random number distributions
- Functions to "project" `sampledSets` and `sampledSurfaces` onto a volume-field
- Execute explicit discretization functions (like `grad`) but select the used scheme in the function instead of using the value from `fvSchemes`
- Calculations of the mesh quality (same way `checkMesh` does) and return as fields
- Do calculations locally on a cell (like the maximum on its faces)
- Get the source fields and other properties from lagrangian clouds based on the basic intermediate cloud classes (Kinematic, Thermo, Reacting, ReactingMultiphase)

It has been tried to make the names unique instead of short. Usually function names from one library are prefixed with the same short string.

- Dynamically generated lists of `functionObjects`
  The new `dynamicFunctionObjectListProxy` in the `simpleFunctionObjects` can generate a `functionObjectList` from a string and execute them during the run like regular function-objects.

  The string is provided by a special class (the so called `dictionaryProvider`). Current implementations for the provider are:

  fromFileDictionaryProvider reads the text from a dictionary file

stdoutFromCommandProvider executes a program and takes the standard output as the dictionary text

stdoutFromPythonScriptProvider executes a python-script and takes the stdout as the dictionary text

The string must be in the format o a regular OpenFOAM-dictionary with an entry `functions` from which the functionObjects are generated

- Function object `readAndUpdateFields`
  This FO in the `simpleFunctionObjects` reads a number of fields and updates their boundary conditions at every timestep.

  Main purpose is to let `groovyBC` do calculations and use the results for post-processing purposes

  Does not support surface-fields as these don't have a `correctBoundaryConditions`-method.

  Example of the usage in the `angledDuctImplicit`-case (the results are of limited value because of the temperature boundary condition)

- Source terms based on `basicSource`
  Three source terms were added. These source terms are in the `swakSourceFields`-library and can be used with solvers that use the `sourcesProperties`-dictionary. The sources are

  SwakSetValue sets values according to a mask or the mechanism provided by `basicSource`

  SwakExplicitSource Uses the calculated source term on the right hand side of the equation

  SwakImplicitSource Uses a calculated scalar-field to add an implicit source term (source is **without** the actual field)

  These fields are only implemented in the 2.x-version of `swak` because the interface of `basicSource` is very different in 1.7 and a backport was unnecessary

- Function objects that stop a run gracefully
  `simpleFunctionObjects` now has a function object `writeAndEndFieldRange` that stops a run (and writes the last time) if a field is outside a specified range.

  A similar function object `writeAndEndSwakExpression` is in the `simpleSwakFunctionObjects` that stops if a swak-expression evaluates to true. `writeAndEndPython` does the same in `pythonIntegration`.

Note: after the run is written one more timestep is calculated (this seems to be due to the fact that FOs are calculated at the start of a timestep). Also there are issues if the next timestep is a scheduled write-time (this only seem to be an issue with 1.7.x. It all works fine on 2.1.x)

- Function-objects to load thermophysical and turbulence models
  New function objects in the `simpleFunctionObjects` allow the loading of such models for solvers/utilities that don't have such models but where some functionObject (for instance) needs such a model to be in memory

- Function-objects that create and evolve clouds of lagrangian particles
  Added as part of the `simpleFunctionObjects` some functionObjects that create a cloud of particles and evolve them at every timestep.

  The appropriate fields needed by every cloud have to be present (either supplied by the solver or via a `functionObject`)

- Function-object `manipulatePatchField` to manipulate the field on patches
  This function objects allows the manipulation of patch fields like `manipulateField` allows the manipulation of the internal field. Only use if desperate

- Delayed variables to simulate responses
  If a variable is declared in the `delayedVariables`-list then its behavior changes: when a value is assigned to that variable then the value is not immediately used but after the time specified in `delay`. Values are stored at intervals `storeInterval` and interpolated linearly. If no stored data is available then the value of the expression `startupValue` is used.

  This feature allows the modeling of boundary conditions that react with a delay to flow conditions

- Allow preloading of fields in `funkySetFields`
  To satisfy the requirements of certain boundary conditions `funkySetFields` now allows the preloading of fields. This is only available in dictionary mode with the `preloadFields`-entry (for each entry in the `expressions`-list separately)

9.9.5 Infrastructure

- Compilation script checks `SWAK4FOAM_SRC`
  The environment variable `SWAK4FOAM_SRC` is needed for the `swakCoded`-functionObject. The `Allwmake`-script now checks whether this variable exists and prints a message with the correct value if it doesn't. It also checks whether the value is correct and warns if it isn't

- `Allwmake` creates symbolic links in `swakFiniteArea`
  The links to `mybison` and `myflex` are missing when the sources are downloaded as a tarball. The `Allwmake`-script now creates these links if they are missing

- Reformatting of the parser sources
  Sources of the parsers have been completely reformatted to make them more readable and maintainable

- Move non-parser sources in `swak4FoamParsers` into sub-directories
  Make the directory a little bit cleaner