CFDEMcoupling Documentation

1. Contents

The CFDEMcoupling documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so we can improve the CFDEMcoupling documentation.

1.1 About CFDEMcoupling

CFDEM coupling provides an open source parallel coupled CFD-DEM framework combining the strengths of LIGGGHTS DEM code and the Open Source CFD package OpenFOAM(R)(*). The CFDEMcoupling toolbox allows to expand standard CFD solvers of OpenFOAM(R)(*) to include a coupling to the DEM code LIGGGHTS. In this toolbox the particle representation within the CFD solver is organized by "cloud" classes. Key functionalities are organised in sub-models (e.g. force models, data exchange models, etc.) which can easily be selected and combined by dictionary settings.

The coupled solvers run fully parallel on distributed-memory clusters. Features are:

- its modular approach allows users to easily implement new models
- its MPI parallelization enables to use it for large scale problems
- the forum on CFD-DEM gives the possibility to exchange with other users / developers

- the use of GIT allows to easily update to the latest version
- basic documentation is provided

The file structure:

- *src* directory including the source files of the coupling toolbox and models
- *applications* directory including the solver files for coupled CFD-DEM simulations
- *doc* directory including the documentation of CFDEMcoupling
- *tutorials* directory including basic tutorial cases showing the functionality

Details on installation are given on the CFDEMproject WWW Site . The functionality of this CFD-DEM framwork is described via tutorial cases showing how to use different solvers and models.

CFDEMcoupling stands for Computational Fluid Dynamics (CFD) -Discrete Element Method (DEM) coupling.

CFDEMcoupling is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

Core development of CFDEMcoupling is done by Christoph Goniva and Christoph Kloss, both at DCS Computing GmbH, 2012

This documentation was written by Christoph Goniva, DCS Computing GmbH, 2012

(*) OpenFOAM(R) is a registered trade mark of Silicon Graphics International Corp. This offering is not affiliated, approved or endorsed by Silicon Graphics International Corp., the producer of the OpenFOAM(R) software and owner of the OpenFOAM(R) trademark.

1.2 Installation

Please follow the installation routine provided at www.cfdem.com. In order to get the latest code version, please use the git repository at http://github.com (githubAccess).

1.3 Tutorials

General:

Each solver of the CFDEMcoupling is comes with at least one tutorial example, showing its functionality and correct useage. Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell scripts. These scripts perform all necessary steps (preprocessing, run, postprocessing, visualization).

Location:

The tutorials can be found in the directory $CFDEM_PROJECT_DIR/tutorials, which can be reached by typing "cfdemTut"

Structure:

Each case is structured in a directory called "CFD" covering the CFD relevant settings and data, and a dirctory called "DEM" covering the DEM relevant settings and data. This allows to easily expand a pure CFD or DEM simulation case to a coupled case.

Usage:

Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell script, executed by typing "./Allrun.sh". The successful run of the script might need some third party software (e.g. octave, evince, etc.).

Settings:

The main settings of a simulation are done via dictionaries:

The DEM setup of each case is defined by a LIGGGHTS input file located in $caseDir/DEM (e.g. in.liggghts_init). For details on the LIGGGHTS setup, please have a look in the LIGGGHTS manual.

Standard CFD settings are defined in $caseDir/CFD/constant (e.g. transportProperties, RASproperties, etc.) and $caseDir/CFD/system (e.g. fvSchemes, controlDict). You can find more information on that in OpenFOAM(R)(*) documentations (www.openFoam.com)(*).

Settings of the coupling routines are defined in $caseDir/CFD/constant/couplingProperies (e.g. force models, data exchange model, etc.) and $caseDir/CFD/constant/liggghtsCommands (allows to execute a LIGGGHTS command during a coupled simulation).

1.4 "couplingProperties" dictionary

General:

In the "couplingProperties" dictionary the setup of the coupling routines of the CFD-DEM simulation are defined.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, "sub-models & settings" and "sub-model properties".

In "sub-models & settings" the following routines must be specified:

- modelType
- couplingInterval
- voidFractionModel
- locateModel
- meshMotionModel
- regionModel
- IOModel
- dataExchangeModel
- averagingModel
- forceModels
- momCoupleModels
- turbulenceModelType

In "sub-model properties" sub-dictionaries might be defined to specify model specific parameters.

Settings:

Reasonable example settings for the "couplingProperties" dictionary are given in the tutorial cases.

1.5 "liggghtsCommands" dictionary

General:

In the "liggghtsCommands" dictionary liggghts commands being executed during a coupled CFD-DEM simulation are specified.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, first a list of "liggghtsCommandModels" is defined, then the settings for each model must be specified.

Settings:

Reasonable example settings for the "liggghtsCommands" dictionary are given in the tutorial cases.

1.6 Models/Solvers

This section lists all CFDEMcoupling sub-models and solvers alphabetically, with a separate listing below of styles within certain commands.

IOModel	IOModel_basicIO
IOModel_noIO	averagingModel
averagingModel_dilute	cfdemSolverIB
cfdemSolverPiso	cfdemSolverPisoScalar
clockModel	clockModel_noClock
clockModel_standardClock	dataExchangeModel
dataExchangeModel_noDataExchange	dataExchangeModel_oneWayVTK
dataExchangeModel_twoWayFiles	dataExchangeModel_twoWayMPI
forceModel	forceModel_Archimedes
forceModel_ArchimedesIB	forceModel_DiFeliceDrag
forceModel_GidaspowDrag	forceModel_KochHillDrag
forceModel_LaEuScalarTemp	forceModel_MeiLift
forceModel_SchillerNaumannDrag	forceModel_SchirgaonkarIB
forceModel_gradPForce	forceModel_noDrag
forceModel_virtualMassForce	forceModel_viscForce
liggghtsCommandModel	liggghtsCommandModel_execute
liggghtsCommandModel_readLiggghtsData	liggghtsCommandModel_runLiggghts
liggghtsCommandModel_writeLiggghts	locateModel
locateModel_engineSearch	locateModel_engineSearchIB
locateModel_standardSearch	meshMotionModel
meshMotionModel_noMeshMotion	momCoupleModel
momCoupleModel_explicitCouple	momCoupleModel_implicitCouple
momCoupleModel_noCouple	regionModel
regionModel_allRegion	voidfractionModel

voidfractionModel_GaussVoidFraction	voidfractionModel_IBVoidFraction
voidfractionModel_bigParticleVoidFraction	voidfractionModel_centreVoidFraction
voidfractionModel_dividedVoidFraction	

averagingModel_dense command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel dense;
```

Examples:

```
averagingModel dense;
```

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities). In the "cfdemParticle cloud" this averaging model is used to calculate the average particle velocity inside a CFD cell. The "dense" model is supposed to be applied to cases where the granular regime is rather dense. The particle velocity inside a CFD cell is evaluated as an ensemble average of the particle velocities.

Restrictions:

None.

Related commands:

averagingModel, dilute

Default: none

averagingModel_dilute command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel dilute;
```

Examples:

```
averagingModel dilute;
```

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities). In the "cfdemParticle cloud" this averaging model is used to calculate the average particle velocity inside a CFD cell. The "dilute" model is supposed to be applied to cases where the granular regime is rather dilute. The particle velocity inside a CFD cell is evaluated from a single particle in a cell (no averaging).

Restrictions:

This model is computationally efficient, but should only be used when only one particle is inside one CFD cell.

Related commands:

averagingModel, dense

averagingModel command

Syntax:

Defined in couplingProperties dictionary.

```
averagingModel model;
```

- model = name of averaging model to be applied

Examples:

```
test
averagingModel dense;
averagingModel dilute;
```

Note: This examples list might not be complete - please look for other averagin models (averagingModel_XY) in this documentation.

Description:

The averaging model performs the Lagrangian->Eulerian mapping of data (e.g. particle velocities).

Restrictions:

None.

Related commands:

dense, dilute

Default: none

CFDEMcoupling Documentation

1. Contents

The CFDEMcoupling documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so we can improve the CFDEMcoupling documentation.

1.1 About CFDEMcoupling

CFDEM coupling provides an open source parallel coupled CFD-DEM framework combining the strengths of LIGGGHTS DEM code and the Open Source CFD package OpenFOAM(R)(*). The CFDEMcoupling toolbox allows to expand standard CFD solvers of OpenFOAM(R)(*) to include a coupling to the DEM code LIGGGHTS. In this toolbox the particle representation within the CFD solver is organized by "cloud" classes. Key functionalities are organised in sub-models (e.g. force models, data exchange models, etc.) which can easily be selected and combined by dictionary settings.

The coupled solvers run fully parallel on distributed-memory clusters. Features are:

- its modular approach allows users to easily implement new models
- its MPI parallelization enables to use it for large scale problems
- the forum on CFD-DEM gives the possibility to exchange with other users / developers
- the use of GIT allows to easily update to the latest version
- basic documentation is provided

The file structure:

- *src* directory including the source files of the coupling toolbox and models
- *applications* directory including the solver files for coupled CFD-DEM simulations
- *doc* directory including the documentation of CFDEMcoupling
- *tutorials* directory including basic tutorial cases showing the functionality

Details on installation are given on the CFDEMproject WWW Site . The functionality of this CFD-DEM framwork is described via tutorial cases showing how to use different solvers and models.

CFDEMcoupling stands for Computational Fluid Dynamics (CFD) -Discrete Element Method (DEM) coupling.

CFDEMcoupling is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

Core development of CFDEMcoupling is done by Christoph Goniva and Christoph Kloss, both at DCS Computing GmbH, 2012

This documentation was written by Christoph Goniva, DCS Computing GmbH, 2012

(*) OpenFOAM(R) is a registered trade mark of Silicon Graphics International Corp. This offering is not affiliated, approved or endorsed by Silicon Graphics International Corp., the producer of the OpenFOAM(R) software and owner of the OpenFOAM(R) trademark.

1.2 Installation

Please follow the installation routine provided at www.cfdem.com. In order to get the latest code version, please use the git repository at http://github.com (githubAccess).

1.3 Tutorials

General:

Each solver of the CFDEMcoupling is comes with at least one tutorial example, showing its functionality and correct useage. Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell scripts. These scripts perform all necessary steps (preprocessing, run, postprocessing, visualization).

Location:

The tutorials can be found in the directory $CFDEM_PROJECT_DIR/tutorials, which can be reached by typing "cfdemTut"

Structure:

Each case is structured in a directory called "CFD" covering the CFD relevant settings and data, and a dirctory called "DEM" covering the DEM relevant settings and data. This allows to easily expand a pure CFD or DEM simulation case to a coupled case.

Usage:

Provided that the installation is correct, the tutorials can be run via "Allrun.sh" shell script, executed by typing "./Allrun.sh". The successful run of the script might need some third party software (e.g. octave, evince, etc.).

Settings:

The main settings of a simulation are done via dictionaries:

The DEM setup of each case is defined by a LIGGGHTS input file located in $caseDir/DEM (e.g. in.liggghts_init). For details on the LIGGGHTS setup, please have a look in the LIGGGHTS manual.

Standard CFD settings are defined in $caseDir/CFD/constant (e.g. transportProperties, RASproperties, etc.) and $caseDir/CFD/system (e.g. fvSchemes, controlDict). You can find more information on that in OpenFOAM(R)(*) documentations (www.openFoam.com)(*).

Settings of the coupling routines are defined in $caseDir/CFD/constant/couplingProperies (e.g. force models, data exchange model, etc.) and $caseDir/CFD/constant/liggghtsCommands (allows to execute a LIGGGHTS command during a coupled simulation).

1.4 "couplingProperties" dictionary

General:

In the "couplingProperties" dictionary the setup of the coupling routines of the CFD-DEM simulation are defined.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, "sub-models & settings" and "sub-model properties".

In "sub-models & settings" the following routines must be specified:

- modelType
- couplingInterval
- voidFractionModel
- locateModel
- meshMotionModel
- regionModel
- IOModel
- dataExchangeModel
- averagingModel

- forceModels
- momCoupleModels
- turbulenceModelType

In "sub-model properties" sub-dictionaries might be defined to specify model specific parameters.

Settings:

Reasonable example settings for the "couplingProperties" dictionary are given in the tutorial cases.

1.5 "liggghtsCommands" dictionary

General:

In the "liggghtsCommands" dictionary liggghts commands being executed during a coupled CFD-DEM simulation are specified.

Location: $caseDir/CFD/constant

Structure:

The dictionary is divided into two parts, first a list of "liggghtsCommandModels" is defined, then the settings for each model must be specified.

Settings:

Reasonable example settings for the "liggghtsCommands" dictionary are given in the tutorial cases.

1.6 Models/Solvers

This section lists all CFDEMcoupling sub-models and solvers alphabetically, with a separate listing below of styles within certain commands.

IOModel	IOModel_basicIO
IOModel_noIO	averagingModel
averagingModel_dilute	cfdemSolverIB
cfdemSolverInterDyM	cfdemSolverInterDyMAirRelax
cfdemSolverPiso	cfdemSolverPisoScalar
clockModel	clockModel_noClock
clockModel_standardClock	dataExchangeModel
dataExchangeModel_noDataExchange	dataExchangeModel_oneWayVTK
dataExchangeModel_twoWayFiles	dataExchangeModel_twoWayMPI
forceModel	forceModelMS
forceModelMS_DiFeliceDragMS	forceModel_Archimedes
forceModel_ArchimedesIB	forceModel_DiFeliceDrag
forceModel_GidaspowDrag	forceModel_KochHillDrag
forceModel_KochHillDragNLift	forceModel_LaEuScalarDust

forceModel_LaEuScalarTemp	forceModel_MeiLift
forceModel_SchillerNaumannDrag	forceModel_SchirgaonkarIB
forceModel_fieldTimeAverage	forceModel_gradPForce
forceModel_gradULiftForce	forceModel_interface
forceModel_noDrag	forceModel_totalMomentumExchange
forceModel_virtualMassForce	forceModel_viscForce
forceModel_volWeightedAverage	liggghtsCommandModel
liggghtsCommandModel_execute	liggghtsCommandModel_readLiggghtsData
liggghtsCommandModel_runLiggghts	liggghtsCommandModel_writeLiggghts
locateModel	locateModel_engineSearch
locateModel_engineSearchIB	locateModel_standardSearch
locateModel_turboEngineSearch	meshMotionModel
meshMotionModel_DEMdrivenMeshMotion	meshMotionModel_noMeshMotion
momCoupleModel	momCoupleModel_explicitCouple
momCoupleModel_implicitCouple	momCoupleModel_noCouple
regionModel	regionModel_allRegion
regionModel_differentialRegion	voidfractionModel
voidfractionModel_GaussVoidFraction	voidfractionModel_IBVoidFraction
voidfractionModel_bigParticleVoidFraction	voidfractionModel_centreVoidFraction
voidfractionModel_dividedMSVoidFractionMS	voidfractionModel_dividedVoidFraction

cfdemSolverIB command

Description:

"cfdemSolverIB" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework, for calculating the dynamics between immersed bodies and the surrounding fluid. Being an implementation of an immersed boundary method it allows tackling problems where the body diameter exceeds the maximal size of a fluid cell. Usung the toolbox of OpenFOAM(R)(*) the governing equations of the fluid are computed and the corrections of velocity and pressure field with respect to the body-movement information, gained from LIGGGHTS, are incorporated.

Code of this solver contributions by Alice Hager, JKU.

see:

GONIVA, C., KLOSS, C., HAGER,A., WIERINK, G. and PIRKER, S. (2011): "A MULTI-PURPOSE OPEN SOURCE CFD-DEM APPROACH", Proc. of the 8th Int. Conf. on CFD in Oil and Gas, Metallurgical and Process Industries, Trondheim, Norway

and

HAGER, A., KLOSS, C. and GONIVA, C. (2011): "TOWARDS AN EFFICIENT IMMERSED BOUNDARY METHOD WITHIN AN OPEN SOURCE FRAMEWORK", Proc. of the 8th Int. Conf. on CFD in Oil and Gas, Metallurgical and Process Industries, Trondheim, Norway

---

---

cfdemSolverInterDyMAirRelax command

Description:

"cfdemSolverInterDyMAirRelax" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on InterDyMFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations for two phases (VOF) applying PISO algorithm, "cfdemSolverInterDyMAirRelax" has additional functionality for a coupling to the DEM code "LIGGGHTS". The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles whose trajectories are calculated in the DEM code LIGGGHTS.

INPUT FROM KLEMENS PLEASE!

see:

GONIVA, C., GRUBER, K., KLOSS, C. (2012): "Sediment Erosion a Numerical and Experimental Study", Proc. of the River Flow Conference 2012, Costa Rica

cfdemSolverInterDyM command

Description:

"cfdemSolverInterDyM" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on InterDyMFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations for two phases (VOF) applying PISO algorithm, "cfdemSolverInterDyM" has additional functionality for a coupling to the DEM code "LIGGGHTS". The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles whose trajectories are calculated in the DEM code LIGGGHTS.

see:

GONIVA, C., GRUBER, K., KLOSS, C. (2012): "Sediment Erosion a Numerical and Experimental Study", Proc. of the River Flow Conference 2012, Costa Rica

cfdemSolverPiso command

Description:

"cfdemSolverPiso" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on pisoFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations applying PISO algorithm, "cfdemSolverPiso" has additional functionality for a coupling to the DEM code "LIGGGHTS". The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles whose trajectories are calculated in the DEM code LIGGGHTS.

see:

GONIVA, C., KLOSS, C., HAGER,A. and PIRKER, S. (2010): "An Open Source CFD-DEM Perspective", Proc. of OpenFOAM Workshop, GÃ¶teborg, June 22.-24.

cfdemSolverPisoScalar command

Description:

"cfdemSolverPisoScalar" is a coupled CFD-DEM solver using CFDEMcoupling, an open source parallel coupled CFD-DEM framework. Based on pisoFoam(R)(*), a finite volume based solver for turbulent Navier-Stokes equations applying PISO algorithm, "cfdemSolverPisoScalar" has additional functionality for a coupling to the DEM code "LIGGGHTS" as well as a scalar transport equation. The volume averaged Navier-Stokes Equations are solved accounting for momentum exchange and volume displacement of discrete particles whose trajectories are calculated in the DEM code LIGGGHTS. The scalar transport equation is coupled to scalar properties of the particle phase, thus convective heat transfer in a fluid granular system can be modeled with "cfdemSolverPisoScalar".

see:

GONIVA, C., KLOSS, C., HAGER,A. and PIRKER, S. (2010): "An Open Source CFD-DEM Perspective", Proc. of OpenFOAM Workshop, GÃ¶teborg, June 22.-24.

clockModel command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel model;
```

- model = name of the clockModel to be applied

Examples:

```
clockModel standardClock;
```

Note: This examples list might not be complete - please look for other models (clockModel_XY) in this documentation.

Description:

The clockModel is the base class for models to examine the code/algorithm with respect to run time.

Main parts of this model are written by Josef Kerbl, JKU.

Restrictions: none.

Default: none.

clockModel_noClock command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel off;
```

Examples:

```
clockModel off;
```

Description:

The "noClock" model is a dummy clockModel model which does not measure/evaluate the run time.

Restrictions: none.

Related commands:

clockModel

clockModel_standardClock command

Syntax:

Defined in couplingProperties dictionary.

```
clockModel standardClock;
```

Examples:

```
clockModel standardClock;
```

Description:

The "standardClock" model is a basic clockModel model which measures the run time between every ".start(name)" and ".stop()" statement placed in the code. If a ".start(name)" is called more than once (e.g. in a loop) the accumulated times are calculated. After the simulation has finished, the data is stored in $caseDir/CFD/clockData/$startTime/*.txt .

Restrictions: none.

Related commands:

clockModel

dataExchangeModel command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel model;
```

- model = name of data exchange model to be applied

Examples:

```
dataExchangeModel twoWayFiles;
dataExchangeModel twoWayMPI;
```

Note: This examples list might not be complete - please look for other models (dataExchangeModel_XY) in this documentation.

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code.

Restrictions:

None.

Related commands:

noDataExchange, oneWayVTK, twoWayFiles, twoWayMPI

Default: none

dataExchangeModel_noDataExchange command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel noDataExchange;
```

Examples:

```
dataExchangeModel noDataExchange;
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The noDataExchange model is a dummy model where no data is exchanged.

Restrictions:

None.

Related commands:

dataExchangeModel

dataExchangeModel_oneWayVTK command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel oneWayVTK;
oneWayVTKProps
{
    DEMts timeStep;
    relativePath "path";
    couplingFilename "filename";
    maxNumberOfParticles number;
};
```

- *timeStep* = time step size of stored DEM data
- *path* = path to the VTK data files relative do simulation directory
- *filename* = filename of the VTK file series
- *number* = maximum nuber of particles in DEM simulation

Examples:

```
dataExchangeModel oneWayVTK;
oneWayVTKProps
{
    DEMts 0.0001;
    relativePath "../DEM/post";
    couplingFilename "vtk_out%4.4d.vtk";
    maxNumberOfParticles 30000;
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The oneWayVTK model is a model that can exchange particle properties from DEM to CFD based on previously stored VTK data.

Restrictions:

None.

Related commands:

dataExchangeModel

dataExchangeModel_twoWayFiles command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel twoWayFiles;
twoWayFilesProps
{
    couplingFilename "filename";
    maxNumberOfParticles number;
};
```

- *filename* = filename of the VTK file series
- *number* = maximum nuber of particles in DEM simulation

Examples:

```
dataExchangeModel twoWayFiles;
twoWayFilesProps
{
    couplingFilename "vtk_out%4.4d.vtk";
    maxNumberOfParticles 30000;
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The twoWayFiles model is a model that can exchange particle properties from DEM to CFD and from CFD to DEM. Data is exchanged via files that are sequentially written/read by the codes.

Restrictions:

Developed only for two processors, one for DEM and on for CFD run.

Related commands:

dataExchangeModel

dataExchangeModel_twoWayMPI command

Syntax:

Defined in couplingProperties dictionary.

```
dataExchangeModel twoWayMPI;
twoWayMPIProps
{
    liggghtsPath "path";
};
```

- *path* = path to the DEM simulation input file

Examples:

```
dataExchangeModel twoWayMPI;
twoWayMPIProps
{
    liggghtsPath "../DEM/in.liggghts_init";
}
```

Description:

The data exchange model performs the data exchange between the DEM code and the CFD code. The twoWayMPI model is a model that can exchange particle properties from DEM to CFD and from CFD to DEM. Data is exchanged via MPI technique. The DEM run is executed by the coupling model, via a liggghtsCommandModel object.

Restrictions:

none.

Related commands:

dataExchangeModel

forceModel_Archimedes command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    Archimedes
);
ArchimedesProps
{
    densityFieldName "density";
    gravityFieldName "gravity";
};
```

- *density* = name of the finite volume density field
- *gravity* = name of the finite volume gravity field

Examples:

```
forceModels
(
    Archimedes
);
ArchimedesProps
{
    densityFieldName "rho";
    gravityFieldName "g";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The Archimedes model is a model that calculates the Archimedes' volumetric lift force stemming from density difference of fluid and particle.

Restrictions:

none.

Related commands:

forceModel

forceModel_ArchimedesIB command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    ArchimedesIB
);
ArchimedesIBProps
{
    densityFieldName "density";
    gravityFieldName "gravity";
};
```

- *density* = name of the finite volume density field
- *gravity* = name of the finite volume gravity field

Examples:

```
forceModels
(
    ArchimedesIB
);
ArchimedesIBProps
{
    densityFieldName "rho";
    gravityFieldName "g";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The ArchimedesIB model is a model that calculates the ArchimedesIB' volumetric lift force stemming from density difference of fluid and particle. This model is especially suited for resolved CFD-DEM simulations where the particle is represented by immersed boundrary method.

Restrictions:

Only for immersed boundary solvers.

Related commands:

forceModel

forceModel_DiFeliceDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    densityFieldName "density";
    interpolation;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *interpolation* = flag to use interolate interpolated voidfraction and velocity values (normally off)

Examples:

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    densityFieldName "rho";
    interpolation;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The DiFeliceDrag model is a model that calculates the particle based drag force following the correlation of Di Felice (see Zhou et al. (2010), JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel_DiFeliceDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    densityFieldName "density";
    interpolation;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *interpolation* = flag to use interolate interpolated voidfraction and velocity values (normally off)

Examples:

```
forceModels
(
    DiFeliceDrag
);
DiFeliceDragProps
{
    velFieldName "U";
    densityFieldName "rho";
    interpolation;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The DiFeliceDrag model is a model that calculates the particle based drag force following the correlation of Di Felice (see Zhou et al. (2010), JFM).

INPUT FROM KLEMENS PLEASE!!!

Restrictions:

none.

Related commands:

forceModel

forceModel_fieldTimeAverage command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    fieldTimeAverage
);
fieldTimeAverageProps
{
    startTime time;
    scalarFieldNames
    (
        "scalarField"
    );
    vectorFieldNames
    (
        "vectorField"
    );
};
```

- *time* = time to start temporal averaging
- *scalarField* = names of the finite volume scalar fields to be temporally averaged
- *vectorField* = names of the finite volume vector fields to be temporally averaged

Examples:

```
forceModels
(
    fieldTimeAverage
);
fieldTimeAverageProps
{
    startTime 1.0;
    scalarFieldNames
    (
        "voidfraction"
    );
    vectorFieldNames
    (
        "Us"
    );
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! Starting at start time the specified fields are temporally averaged and written at "writeTime". They can be can the be probed using standard function object probes. The output name is timeAverage_scalarField, where scalarField is the name of the original field.

Restrictions:

none.

Related commands:

forceModel

forceModel_GidaspowDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    GidaspowDrag
);
GidaspowDragProps
{
    velFieldName "U";
    densityFieldName "density";
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field

Examples:

```
forceModels
(
    GidaspowDrag
);
GidaspowDragProps
{
    velFieldName "U";
    densityFieldName "rho";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The GidaspowDrag model is a model that calculates the particle based drag force following the correlation of Gidaspow which is a combination of Egrun (1952) and Wen & Yu (1966) (see Zhu et al. (2007): "Discrete particle simulation of particulate systems: Theoretical developments" ,ChemEngScience).

Restrictions:

none.

Related commands:

forceModel

forceModel_gradPForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    gradPForce;
);
gradPForceProps
{
    pFieldName "pressure";
    densityFieldName "density";
    velocityFieldName "U";
    interpolation;
};
```

- *pressure* = name of the finite volume fluid pressure field
- *density* = name of the finite volume gravity field
- *U* = name of the finite volume fluid velocity field
- *interpolation* = flag to use interolate interpolated pressure values (normally off)

Examples:

```
forceModels
(
    gradPForce;
);
gradPForceProps
{
    pFieldName "p";
    densityFieldName "rho";
    velocityFieldName "U";
    interpolation;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The gradPForce model is a model that calculates the particle based pressure gradient force -(grad(p)) * Vparticle (see Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model formulations and their applicability" ,JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel_gradULiftForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    gradULiftForce;
);
gradULiftForceProps
{
    UFieldName "U";
    densityFieldName "density";
    voidfractionFieldName "voidfraction";
    interpolation;
    verbose;
    treatDEM;
    lowerLiftVoidfractionLevel value1;
    upperLiftVoidfractionLevel value2;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *U* = name of the finite volume fluid velocity field
- *interpolation* = flag to use interolate interpolated pressure values (normally off)
- *verbose* = flag to write info to terminal
- *treatDEM* = flag to apply force only on DEM side
- *value1* = ???
- *value2* = ???

Examples:

```
forceModels
(
    gradULiftForce;
);
gradULiftForceProps
{
    UFieldName "U";
    densityFieldName "rho";
    voidfractionFieldName "voidfraction";
    interpolation;
    verbose;
    treatDEM;
    lowerLiftVoidfractionLevel 0.5;
    upperLiftVoidfractionLevel 0.8;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The gradULiftForce model is a model that calculates the particle based pressure gradient force -(grad(p)) * Vparticle (see Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model

formulations and their applicability" ,JFM). pressure gradient force grad(p)*Vp=mp*rhof/rhop(U*grad(U)) based on cfx Manual p153 Eq. 5.27 including interpolation of the velocity to the exact position INPUT FROM KLEMENS PLEASE!!!

Restrictions:

none.

Related commands:

forceModel

forceModel command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    model_x
    model_y
);
```

- model = name of force model to be applied

Examples:

```
forceModels
(
    Archimedes
    DiFeliceDrag
);
```

Note: This examples list might not be complete - please look for other models (forceModel_XY) in this documentation.

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. All force models selected are executed sequentially and the forces on the particles are superposed.

Restrictions:

None.

Related commands:

Archimedes, DiFeliceDrag, gradPForce, viscForce

Note: This examples list may be incomplete - please look for other models (forceModel_XY) in this documentation.

Default: none.

forceModel_interface command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    interface
);
interfaceProps
{
    VOFvoidfractionFieldName "alpha";
    alphaThreshold value;
    sigma   value2;
    theta   value3;
};
```

- *alpha* = name of the finite volume fluid volume fraction field in a VOF simulatuion
- *value* = parameter used to define the distance of influence of the model
- *value2* = fluid gas surface tension
- *value3* = Three-phase contact angle for interface force

Examples:

```
forceModels
(
    interface
);
interfaceProps
{
    VOFvoidfractionFieldName "alpha1";
    alphaThreshold 0.55;
    sigma   0.072;
    theta   1.15;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The interface model is a model that calculates the particle-fluid-gas interfacial force. (see Wierink et al. (2011): "Mechanistic modelling of particle interface interaction in three phase flows" ,proc of the 8th int. conf. on CFD in oil and gas..., ISBN 9788214052626).

Restrictions:

Only for VOF simulations.

Related commands:

forceModel

forceModel_KochHillDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    KochHillDrag
);
KochHillDragProps
{
    velFieldName "U";
    densityFieldName "density";
    voidfractionFieldName "voidfraction";
    interpolation;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *voidfraction* = name of the finite volume voidfraction field
- *interpolation* = flag to use interolate interpolated voidfraction and fluid velocity values (normally off)

Examples:

```
forceModels
(
    KochHillDrag
);
KochHillDragProps
{
    velFieldName "U";
    densityFieldName "rho";
    voidfractionFieldName "voidfraction";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The KochHillDrag model is a model that calculates the particle based drag force following the correlation of Koch & Hill (2001) (see van Buijtenen et al. (2011): "Numerical and experimental study on multiple-spout fluidized beds" ,ChemEngScience).

Restrictions:

none.

Related commands:

forceModel

forceModel_KochHillDragNLift command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    KochHillDragNLift
);
KochHillDragNLiftProps
{
    velFieldName "U";
    densityFieldName "density";
    volfFieldName "voidfraction";
    interpolation;
    dragForce;
    modSurfaceDrag;
    modSurfaceDragVolumefraction value1;
    rho_1000;
    beta value2;
    betaLaw;
    verbose;
    liftForce;
    liftDragRatio value3;
    lowerLiftVoidfractionLevel value4;
    upperLiftVoidfractionLevel value5;
    kFieldName "k";
    turbEffect;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *voidfraction* = name of the finite volume voidfraction field
- *interpolation* = flag to use interolate interpolated voidfraction and fluid velocity values (normally off)
- *dragForce* = flag to include drag force
- *modSurfaceDrag* = flag to ???
- *value1* = ???
- *rho_1000* = flag to ???
- *value2* = flag to ???
- *betaLaw* = flag to ???
- *verbose* = flag to write info to terminal
- *liftForce* = flag to include lift force
- *value3* = ???
- *value4* = ???
- *value5* = ???
- *k* = turbulent kinetic energy field
- *turbEffect* = flag to ???

Examples:

```
forceModels
(
```

```
    KochHillDragNLift
);
KochHillDragNLiftProps
{
    velFieldName "U";
    densityFieldName "rho";
    voidfractionFieldName "voidfraction";
    interpolation;
    dragForce;
    modSurfaceDrag;
    modSurfaceDragVolumefraction 0.5;
    rho_1000;
    beta 1;
    betaLaw;
    verbose;
    liftForce;
    liftDragRatio 1.5;
    lowerLiftVoidfractionLevel 0.5;
    upperLiftVoidfractionLevel 0.8;
    kFieldName "k";
    turbEffect;
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The KochHillDragNLift model is a model that calculates the particle based drag force following the correlation of Koch & Hill (2001) (see van Buijtenen et al. (2011): "Numerical and experimental study on multiple-spout fluidized beds" ,ChemEngScience).

INPUT FROM KLEMENS PLEASE!!! Chepil (1958) "The use of evenly spaced hemisphere to evaluate aerodynamic forces on a soil surface."

Restrictions:

none.

Related commands:

forceModel

forceModel_LaEuScalarDust command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    LaEuScalarDust
);
LaEuScalarDustProps
{
    velFieldName "U";
    dustFieldName "T";
    dustSourceFieldName "Tsource";
    partAccelerationName "acceleration";
    partDustFluxName "fluxName";
    velModelParam value;
    accModelParam value1;
};
```

- *U* = name of the finite volume fluid velocity field
- *T* = name of the finite volume scalar field field representing dust:l *Tsource* = name of the finite volume scalar field field representing dust sources:l *acceleration* = name of the DEM data representing the particles's accelerations
- *fluxName* = name of the DEM data representing the particles-fluid scalar exchange flux
- *value* = model parameter
- *value1* = model parameter

Examples:

```
forceModels
(
    LaEuScalarDust
);
LaEuScalarDustProps
{
    velFieldName "U";
    dustFieldName "T";
    dustSourceFieldName "Tsource";
    partAccelerationName "acceleration";
    partDustFluxName "convectiveHeatFlux";
    velModelParam -0.001;
    accModelParam 0;
}
```

Description:

This "forceModel" does not influence the particles or the fluid flow! Using the particles' acceleration a scalar field representing "produced" dust is calculated. The solver then uses this source field in the scalar transport equation. The model for dust production is based on Hilton and Cleary (2011): "Dust dispersal modelling in a conveyor chute using a coupled discrete element and CFD method", proc of the 8th int. conf. on CFD in oil and gas..., ISBN 9788214052626)

Restrictions:

Goes only with cfdemSolverScalar.

Related commands:

forceModel

forceModel_LaEuScalarTemp command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    LaEuScalarTemp
);
LaEuScalarTempProps
{
    velFieldName "U";
    tempFieldName "T";
    tempSourceFieldName "Tsource";
    voidfractionFieldName "voidfraction";
    partTempName "Temp";
    partHeatFluxName "convectiveHeatFlux";
    lambda value;
    Cp value1;
    densityFieldName "density";
};
```

- *U* = name of the finite volume fluid velocity field
- *T* = name of the finite volume scalar temperature field
- *Tsource* = name of the finite volume scalar temperature source field
- *voidfraction* = name of the finite volume voidfraction field
- *Temp* = name of the DEM data representing the particles temperature
- *convectiveHeatFlux* = name of the DEM data representing the particle-fluid convective heat flux
- *value* = fluid thermal conductivity [W/(m*K)]
- *value1* = fluid specific heat capacity [W*s/(kg*K)]
- *density* = name of the finite volume fluid density field

Examples:

```
forceModels
(
    LaEuScalarTemp
);
LaEuScalarTempProps
{
    velFieldName "U";
    tempFieldName "T";
    tempSourceFieldName "Tsource";
    voidfractionFieldName "voidfraction";
    partTempName "Temp";
    partHeatFluxName "convectiveHeatFlux";
    lambda 0.0256;
    Cp 1007;
    densityFieldName "rho";
}
```

Description:

This "forceModel" does not influence the particles or the fluid flow! Using the particles' temperature a scalar field representing "particle-fluid heatflux" is calculated. The solver then uses this source field in the scalar transport equation for the temperature. The model for convective heat transfer is based on Li and Mason (2000), A computational investigation of transient heat transfer in pneumatic transport of granular particles, Pow.Tech 112

Restrictions:

Goes only with cfdemSolverScalar.

Related commands:

forceModel

forceModel_MeiLift command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    MeiLift
);
MeiLiftProps
{
    velFieldName "U";
    densityFieldName "density";
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume fluid density field

Examples:

```
forceModels
(
    MeiLift
);
MeiLiftProps
{
    velFieldName "U";
    densityFieldName "rho";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The MeiLift model calculates the lift force for each particle based on Loth and Dorgan (2009)

Restrictions:

None.

Related commands:

forceModel

forceModelMS_DiFeliceDragMS command

Syntax:

Defined in couplingProperties dictionary.

```
forceModelsMS
(
    DiFeliceDrag
);
DiFeliceDragMSProps
{
    velFieldName "U";
    densityFieldName "density";
    voidfractionFieldName "voidfraction";
    hydraulicDiameter 0.005;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *voidfraction* = name of the finite volume voidfraction field
- *hydraulicDiameter* = hydraulic diameter of the clump used for the drag calculation

Examples:

```
forceModelsMS
(
    DiFeliceDragMS
);
DiFeliceDragMSProps
{
    velFieldName "U";
    densityFieldName "rho";
    voidfractionFieldName "voidfraction";
    hydraulicDiameter 0.005;
}
```

Description:

The force model MS (Multi-Sphere) performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM clump (multisphere consisting of several particles). The DiFeliceDragMS model calculates the particle based drag force following the correlation of Di Felice (see Zhou et al. (2010), JFM).

Restrictions:

Only for multisphere solvers.

Related commands:

forceModelMS

forceModelMS command

Syntax:

Defined in couplingProperties dictionary.

```
forceModelsMS
(
    model_x
    model_y
);
```

- model = name of force model to be applied

Examples:

```
forceModels
(
    off
);
forceModelsMS
(
    DiFeliceDragMS
);
```

Note: This examples list might not be complete - please look for other models (forceModelMS_XY) in this documentation.

Description:

The force model MS (Multi-Sphere) performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM clump (multisphere consisting of several particles). All force models selected are executed sequentially and the forces on the particles are superposed.

Restrictions:

Only for multisphere solvers.

Related commands:

DiFeliceDragMS

Note: This examples list may be incomplete - please look for other models (forceModelMS_XY) in this documentation.

Default: none.

forceModel_noDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    off
);
```

Examples:

```
forceModels
(
    off
);
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The noDrag model sets the forces acting on the particle to zero. If several force models are selected and noDrag is the last model being executed, the fluid particle force will be set to zero.

Restrictions:

None.

Related commands:

forceModel

forceModel_SchillerNaumannDrag command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    SchillerNaumannDrag
);
SchillerNaumannDragProps
{
    velFieldName "U";
    densityFieldName "density";
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field

Examples:

```
forceModels
(
    SchillerNaumannDrag
);
SchillerNaumannDragProps
{
    velFieldName "U";
    densityFieldName "rho";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The SchillerNaumannDrag model is a model that calculates the particle based drag force following the correlation of Schiller and Naumann.

Restrictions:

none.

Related commands:

forceModel

forceModel_SchirgaonkarIB command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    SchirgaonkarIB
);
SchirgaonkarIBProps
{
    velFieldName "U";
    densityFieldName "density";
    pressureFieldName "pressure";
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume density field
- *pressure* = name of the finite volume pressure field

Examples:

```
forceModels
(
    SchirgaonkarIB
);
SchirgaonkarIBProps
{
    velFieldName "U";
    densityFieldName "rho";
    pressureFieldName "p";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The SchirgaonkarIB model calculates the drag force (viscous and pressure force) acting on each particle in a resolved manner (see Shirgaonkar et al. (2009): "A new mathematical formulation and fast algorithm for fully resolved simulation of self-propulsion", Journal of Comp. Physics). This model is only suited for resolved CFD-DEM simulations where the particle is represented by immersed boundrary method.

Restrictions:

Only for immersed boundary solvers.

Related commands:

forceModel

forceModel_totalMomentumExchange command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    totalMomentumExchange
);
totalMomentumExchangeProps
{
    implicitMomExFieldName "Ksl";
    explicitMomExFieldName "f";
    fluidVelFieldName "U";
    granVelFieldName "Us";
    densityFieldName "rho";
};
```

- *Ksl* = name of the finite volume momentum exchange field of implicit forces \(otherwise "none"\)
- *f* = name of the finite volume momentum exchange field of explicit forces \(otherwise "none"\)
- *U* = name of the finite volume fluid velocity field
- *Us* = name of the finite volume granular velocity field
- *rho* = name of the finite volume fluid density field

Examples:

```
forceModels
(
    totalMomentumExchange
);
totalMomentumExchangeProps
{
    implicitMomExFieldName "Ksl";
    explicitMomExFieldName "none";
    fluidVelFieldName "U";
    granVelFieldName "Us";
    densityFieldName "rho";
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! Every coupling step the total fluid-particle interaction force is evaluated by integration of Ksl * (Us-U) / rho * Vcell + f over the whole CFD domain.

Restrictions:

none.

Related commands:

forceModel

forceModel_virtualMassForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    virtualMassForce
);
virtualMassForceProps
{
    velFieldName "U";
    densityFieldName "density";
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume fluid density field

Examples:

```
forceModels
(
    virtualMassForce
);
virtualMassForceProps
{
    velFieldName "U";
    densityFieldName "rho";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The virtualMassForce model calculates the virtual mass force for each particle.

Restrictions:

Model not validated!

Related commands:

forceModel

forceModel_viscForce command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    viscForce;
);
viscForceProps
{
    velocityFieldName "U";
    densityFieldName "density";
    interpolation;
};
```

- *U* = name of the finite volume fluid velocity field
- *density* = name of the finite volume gravity field
- *interpolation* = flag to use interolate interpolated stress values (normally off)

Examples:

```
forceModels
(
    viscForce;
);
viscForceProps
{
    velocityFieldName "U";
    densityFieldName "density";
}
```

Description:

The force model performs the calculation of forces (e.g. fluid-particle interaction forces) acting on each DEM particle. The viscForce model calculates the particle based viscous force, -(grad(tau)) * Vparticle (see Zhou et al. (2010): "Discrete particle simulation of particle-fluid flow: model formulations and their applicability" ,JFM).

Restrictions:

none.

Related commands:

forceModel

forceModel_volWeightedAverage command

Syntax:

Defined in couplingProperties dictionary.

```
forceModels
(
    volWeightedAverage
);
volWeightedAverageProps
{
    startTime time;
    scalarFieldNames
    (
        scalarField
    );
    vectorFieldNames
    (
        vectorField
    );
    upperThreshold value;
    lowerThreshold value2;
};
```

- *time* = time to start the averaging (default 0)
- *scalarField* = names of the finite volume scalar fields to be temporally averaged
- *vectorField* = names of the finite volume vector fields to be temporally averaged
- *value* = only cells with a field value (magnitude) lower than this upper threshold are considered
- *value2* = only cells with a field value (magnitude) greater than this lower threshold are considered

Examples:

```
forceModels
(
    volWeightedAverage
);
volWeightedAverageProps
{
    startTime 0.1;
    scalarFieldNames
    (
        voidfraction
    );
    vectorFieldNames
    (
    );
    upperThreshold 0.999;
    lowerThreshold 0;
}
```

Description:

This "forceModel" does not influence the particles or the simulation - it is a postprocessing tool! Starting at start time the volume weighted averages of those cells of the fields within the threshold are calculated. At

"writeTime" a field named volAverage_field , where scalarField is the name of the original field, is written. This can be can the be probed using standard function object probes.

Restrictions:

Currently all fields have the same threshold value!

Related commands:

forceModel

githubAccess_non-public

Description:

This routine describes how to setup a github account and pull repositories of the CFDEMproject.

Procedure:

- Basically the following steps have to be performed: create an account at http://github.com
- create your RSA key
- add your RSA key to your github account
- *git clone* the desired repository
- update your repositories by *git pull*
- get access to non-public repositories

Create an account:

Please create a free account at https://github.com.

```
example:
user (username)
user@mail.com
pwd (pwd)
```

Please use your own username and mail adress here and for the following steps!

Create your RSA key:

Please find the complete setup description here, or use the short description below.

Open a terminal and execute:

```
cd ~/.ssh
ssh-keygen -t rsa -C "user@mail.com"
gedit id_rsa.pub&
```

Add your RSA key to your github account:

Login at https://github.com with you user, then

- click *Account Settings*
- click *SSH Keys*
- click *Add SSH key*
- paste your key into the *Key* field
- Hit *Add Key.*

To check your settings, open a terminal and execute:

```
ssh -T git@github.com
```

***git clone* the desired repository:**

To clone the public LIGGGHTS repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/LIGGGHTS-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://user@github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
```

To clone the public CFDEMcoupling repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://user@github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type:

```
git pull
```

Get acces to non-public repositories:

If you have a support contract / non-public repository access by DCS Computing GmbH, please follow above steps to set up your user and RSA key. After that please send your username and company affiliation to DCS Computing GmbH to get your account activated. Afterwards you can clone also the non-public repositories.

githubAccess_public

Description:

This routine describes how to setup a github account and pull repositories of the CFDEMproject. After setting some environment variables LIGGGHTS and CFDEMcoupling can be compiled

Procedure:

Basically the following steps have to be performed:

- *git clone* the desired repository
- update your repositories by *git pull*
- set environment variables
- compile LIGGGHTS and CFDEMcoupling
- run your own cases

***git clone* the desired repository:**

If not already done, open a terminal and create a directory for LIGGGHTS in $HOME:

```
cd

mkdir LIGGGHTS

cd LIGGGHTS
```

To clone the public LIGGGHTS repository, open a terminal and execute:

git clone git://cfdem.git.sourceforge.net/gitroot/cfdem/liggghtsdev LIGGGHTS-PUBLIC

If not already done, open a terminal and create a directory for CFDEMcoupling in $HOME:

```
cd

mkdir CFDEM

cd CFDEM
```

Make sure that OpenFOAM(R)-2.1.x is already set up correctly!

To clone the public CFDEMcoupling repository, open a terminal and execute:

git clone git://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type: *Warning: git stash will remove your changes in $HOME/CFDEM/CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION !*

```
cd $HOME/CFDEM/CFDEMcoupling-PUBLIC-$WM_PROJECT_VERSION
git stash
git pull
```

set environment variables:

Now you need to set some environment variables in ~/.bashrc (if you use c-shell, manipulate ~/.cshrc accordingly). Open ~/.bashrc

```
gedit ~/.bashrc &
```

add the lines:

```
#=============================================#
#- source cfdem env vars
export CFDEM_VERSION=PUBLIC
export CFDEM_PROJECT_DIR=$HOME/CFDEM/CFDEMcoupling-$CFDEM_VERSION-$WM_PROJECT_VERSION
export CFDEM_SRC_DIR=$CFDEM_PROJECT_DIR/src/lagrangian/cfdemParticle
export CFDEM_SOLVER_DIR=$CFDEM_PROJECT_DIR/applications/solvers
export CFDEM_DOC_DIR=$CFDEM_PROJECT_DIR/doc
export CFDEM_UT_DIR=$CFDEM_PROJECT_DIR/applications/utilities
export CFDEM_TUT_DIR=$CFDEM_PROJECT_DIR/tutorials
export CFDEM_PROJECT_USER_DIR=$HOME/CFDEM/$LOGNAME-$CFDEM_VERSION-$WM_PROJECT_VERSION
export CFDEM_bashrc=$CFDEM_SRC_DIR/etc/bashrc
export CFDEM_LIGGGHTS_SRC_DIR=$HOME/LIGGGHTS/LIGGGHTS-PUBLIC/src
export CFDEM_LIGGGHTS_MAKEFILE_NAME=fedora_fpic
. $CFDEM_bashrc
#=============================================#
```

Save the ~/.bashrc, open a new terminal and test the settings. The commands:

```
$CFDEM_PROJECT_DIR
$CFDEM_SRC_DIR
$CFDEM_LIGGGHTS_SRC_DIR
```

should give "...: is a directory" otherwise something went wrong and the environment variables in ~/bashrc are not set correctly.

To specify the paths of pizza, please check the settings in $CFDEM_SRC_DIR/etc/bashrc.

If $CFDEM_SRC_DIR is set correctly, you can type

```
cfdemSysTest
```

to get some information if the paths are set correctly.

compile LIGGGHTS and CFDEMcoupling:

If above settings were done correctly, you can compile LIGGGHTS by typing:

```
git clone git://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git CFDEMcoupling-PUBLIC-$WM_PRO
```

```
cfdemCompLIG
```

and you can then compile CFDEMcoupling by typing:

```
cfdemCompCFDEM
```

You can run the tutorial cases by executing .../etc/testTutorial.sh through the alias *cfdemTestTUT*. Alternatively you can run each tutorial using the *Allrun.sh* scripts in the tutorial directories.

In case questions concerning the installation arise, please feel free to contact our forum at www.cfdem.com.

run your own cases:

If you want to run your own cases, please do so in $CFDEM_PROJECT_USER_DIR/run which is automatically being generated. E.g. copy one of the tutorial cases there, adapt it to your needs. Changes in $CFDEM_TUT_DIR will be lost after every *git stash*!

IOModel_basicIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "basicIO";
```

Examples:

```
IOModel "basicIO";
```

Description:

The basic IO-model writes particle positions velocities and radii to files. The output directory ($casePath/CFD/particles) is created automatically. Data is written every write time of the CFD simulation.

Restrictions: None.

Related commands:

IOModel

IOModel command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "model";
```

- model = name of IO-model to be applied

Examples:

IOModel "off";

Note: This examples list might not be complete - please look for other models (IOModel_XY) in this documentation.

Description:

The IO-model is the base class to write data (e.g. particle properties) to files.

Restrictions:

none.

Related commands:

Note: This examples list may be incomplete - please look for other models (IOModel_XY) in this documentation.

Default: none.

IOModel_noIO command

Syntax:

Defined in couplingProperties dictionary.

```
IOModel "off";
```

Examples:

```
IOModel "off";
```

Description:

The noIO-model is a dummy IO model.

Restrictions: None.

Related commands:

IOModel

liggghtsCommandModel_execute command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   execute
);
executeProps0
{
    command
    (
        run
        $couplingInterval
    );
    runFirst switch1;
    runLast switch2;
    runEveryCouplingStep switch3;
    runEveryWriteStep switch4;
}
```

- *command* = LIGGGHTS command to be executed. Each word in a new line, numbers and symbols need special treatment (e.g. $couplingInterval will be replaced by correct coupling interval in the simulation)
- *switch1* = switch (choose on/off) if the command is executed only at first time step
- *switch2* = switch (choose on/off) if the command is executed only at last time step
- *switch3* = switch (choose on/off) if the command is executed at every coupling step
- *switch4* = switch (choose on/off) if the command is executed at every writing step

Examples:

```
liggghtsCommandModels
(
   execute
   execute
);
executeProps0
{
    command
    (
        run
        $couplingInterval
    );
    runFirst off;
    runLast off;
    runEveryCouplingStep on;
}
executeProps1
{
    command
    (
        write_restart
        noBlanks
        dotdot
```

```
        slash
        DEM
        slash
        liggghts.restart_
        timeStamp
    );
    runFirst off;
    runLast off;
    runEveryCouplingStep off;
    runEveryWriteStep on;
}
```

Description:

The execute liggghtsCommand Model can be used to execute a LIGGGHTS command during a CFD run. In above example execute_0 for instance executes "run $couplingInterval" every coupling step. $couplingInterval is automatically replaced by the correct number of DEM steps. Additionally execute_1 executes "write_restart ../DEM/liggghts.restart_$timeStamp" every writing step, where $timeStamp is automatically set.

These rather complex execute commands can be replaced by the "readLiggghts" and "writeLiggghts" commands!

Restrictions: None.

Related commands:

liggghtsCommandModel

liggghtsCommandModel command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
    model_x
    model_y
);
```

- model = name of the liggghtsCommandModel to be applied

Examples:

```
liggghtsCommandModels
(
   runLiggghts
   writeLiggghts
);
```

Note: This examples list might not be complete - please look for other models (liggghtsCommandModel_XY) in this documentation.

Description:

The liggghtsCommandModel is the base class to execute DEM commands within a CFD run.

Restrictions:

Works only with MPI coupling.

Default: none.

liggghtsCommandModel_readLiggghtsData command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   readLiggghtsData
);
readLiggghtsDataProps0
{
    ???
}
```

Examples:

```
liggghtsCommandModels
(
   readLiggghtsData
   readLiggghtsData
);
readLiggghtsDataProps0
{
    ???
}
```

Description:

The readLiggghtsData liggghtsCommand Model can be used to ???

Restrictions:

Note: Model is not up to date.

Related commands:

liggghtsCommandModel

liggghtsCommandModel_runLiggghts command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   runLiggghts
);
```

Examples:

```
liggghtsCommandModels
(
   runLiggghts
);
```

Description:

The liggghtsCommand models can be used to execute a LIGGGHTS command during a CFD run. The "runLiggghts" command executes the command "run $nrDEMsteps", where $nrDEMsteps is automaically set according to the coupling intervals, every coupling step.

Restrictions: None.

Related commands:

liggghtsCommandModel

liggghtsCommandModel_writeLiggghts command

Syntax:

Defined in liggghtsCommmands dictionary.

```
liggghtsCommandModels
(
   writeLiggghts
);
writeLiggghtsProps
{
    writeName "name";
    overwrite switch;
}
```

- *name* = name of the restart file to be written in /$caseDir/DEM/
- *switch* = switch (choose on/off) to select if only one restart file $name or many files $name_$timeStamp are written

Examples:

```
liggghtsCommandModels
(
   runLiggghts
   writeLiggghts
);
writeLiggghtsProps
{
    writeName "liggghts_restart";
    overwrite off;
}
```

Description:

The liggghtsCommand models can be used to execute a LIGGGHTS command during a CFD write. The "writeLiggghts" command executes the command "write_restart $name", where $name is the name of the restart file, every write step.

Restrictions: None.

Related commands:

liggghtsCommandModel

locateModel_engineSearch command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel engine;
engineProps
{
    faceDecomp switch1;
    treeSearch switch2;
}
```

- *switch1* = time to start the averaging (default 0)
- *switch2* = names of the finite volume scalar fields to be temporally averaged

Examples:

```
locateModel engine;
engineProps
{
    faceDecomp false;
    treeSearch false;
}
```

Description:

The locateModel "engine" locates the CFD cell and cellID corresponding to a given position. The engineSearch locate Model can be used with different settings to use different algorithms:

- faceDecomp false; treeSearch false; will execute some geometric (linear) search using the last known cellID (recommended)
- faceDecomp false; treeSearch true; will use a recursive tree structure to find the cell.

Restrictions: none.

Related commands:

locateModel

locateModel_engineSearchIB command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel engineIB;
engineIBProps
{
    engineProps
    {
        faceDecomp false;
        treeSearch false;
    }
    zSplit value1;
    xySplit value2;
}
```

- *switch1* = time to start the averaging (default 0)
- *switch2* = names of the finite volume scalar fields to be temporally averaged
- *value1* = number of z-normal layers for satellite points
- *value2* = number of satellite points in each layer

Examples:

```
locateModel engineIB;
engineIBProps
{
    engineProps
    {
        faceDecomp false;
        treeSearch false;
    }
    zSplit 8;
    xySplit 16;
}
```

Description:

The locateModel "engine" locates the CFD cell and cellID corresponding to a given position. This locate model is especially designed for parallel immersed boundary method. Each particle is represented by "satellite points" if it is distributed over several processors.

The engineSearchIB locate Model can be used with different settings to use different algorithms:

- faceDecomp false; treeSearch false; will execute some geometric (linear) search using the last known cellID (recommended)
- faceDecomp false; treeSearch true; will use a recursive tree structure to find the cell.

Restrictions:

Only for immersed boundary solvers!

Related commands:

locateModel

locateModel command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel model;
```

- model = name of the locateModel to be applied

Examples:

```
locateModel engine;
```

Note: This examples list might not be complete - please look for other models (locateModel_XY) in this documentation.

Description:

The locateModel is the base class for models which search for the CFD cell and cellID corresponding to a position. In general it is used to find the cell a particle is located in.

Restrictions: none.

Default: none.

locateModel_standardSearch command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel standard;
```

Examples:

```
locateModel standard;
```

Description:

The locateModel "standard" locates the CFD cell and cellID corresponding to a given position. A very straight-forward (robust!) locate algorithm is used.

Restrictions: none.

Related commands:

locateModel

locateModel_turboEngineSearch command

Syntax:

Defined in couplingProperties dictionary.

```
locateModel turboEngine;
turboEngineProps
{
    faceDecomp switch1;
    treeSearch switch2;
}
```

- *switch1* = time to start the averaging (default 0)
- *switch2* = names of the finite volume scalar fields to be temporally averaged

Examples:

```
locateModel turboEngine;
turboEngineProps
{
    faceDecomp false;
    treeSearch false;
}
```

Description:

The locateModel "turboEngine" locates the CFD cell and cellID corresponding to a given position. The algorithm is improved compared to engine search to show better parallel performance.

The turboEngineSearch locate Model can be used with different settings to use different algorithms:

- faceDecomp false; treeSearch false; will execute some geometric (linear) search using the last known cellID (recommended)
- faceDecomp false; treeSearch true; will use a recursive tree structure to find the cell.

Restrictions: none.

Related commands:

locateModel

meshMotionModel_DEMdrivenMeshMotion command

Syntax:

Defined in couplingProperties dictionary.

```
meshMotionModel DEMdrivenMeshMotion;
DEMdrivenMeshMotionProps
{
    movingPatchNames
    (
        "name"
    );
}
```

- *name* = name of patch that are to be moved (list)

Examples:

```
meshMotionModel DEMdrivenMeshMotion;
DEMdrivenMeshMotionProps
{
    movingPatchNames
    (
        "inlet"
    );
}
```

Description:

The noMeshMotion-model is a meshMotion model reading deformation from DEM simulation and move patches of the CFD domain.

Restrictions: None.

Related commands:

meshMotionModel

meshMotionModel command

Syntax:

Defined in couplingProperties dictionary.

```
meshMotionModel model;
```

- model = name of the meshMotionModel to be applied

Examples:

```
meshMotionModel noMeshMotion;
```

Note: This examples list might not be complete - please look for other models (meshMotionModel_XY) in this documentation.

Description:

The meshMotionModel is the base class for models which manipulate the CFD mesh according to the DEM mesh motion.

Restrictions: none.

Default: none.

meshMotionModel_noMeshMotion command

Syntax:

Defined in couplingProperties dictionary.

```
meshMotionModel noMeshMotion;
```

Examples:

```
meshMotionModel noMeshMotion;
```

Description:

The noMeshMotion-model is a dummy meshMotion model.

Restrictions: None.

Related commands:

meshMotionModel

momCoupleModel_explicitCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    explicitCouple
);
explicitCoupleProps
{
    fLimit vector;
}
```

- *vector* = limiter vector for explicit force term (default (1e10,1e10,1e10))

Examples:

```
momCoupleModels
(
    explicitCouple
);
explicitCoupleProps
{
    fLimit (1e3 1e2 1e4);
}
```

Description:

The explicitCouple-model is a momCoupleModel model providing an explicit momentum source term for the CFD solver.

Restrictions:

Only for solvers that include explicit momentum exchange.

Related commands:

momCoupleModel

momCoupleModel command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    model
);
```

- model = name of the momCoupleModel to be applied

Examples:

```
momCoupleModels
(
    implicitCouple
);
```

Note: This examples list might not be complete - please look for other models (momCoupleModel_XY) in this documentation.

Description:

The momCoupleModel is the base class for momentum exchange between DEM and CFD simulation.

Restrictions: none.

Default: none.

momCoupleModel_implicitCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    implicitCouple
);
implicitCoupleProps
{
    velFieldName "U";
    granVelFieldName "Us";
    voidfractionFieldName "voidfraction";
}
```

- *U* = name of the finite volume fluid velocity field
- *Us* = name of the finite volume granular velocity field
- *voidfraction* = name of the finite volume voidfraction field

Examples:

```
momCoupleModels
(
    implicitCouple
);
implicitCoupleProps
{
    velFieldName "U";
    granVelFieldName "Us";
    voidfractionFieldName "voidfraction";
}
```

Description:

The implicitCouple-model is a momCoupleModel model providing an implicit momentum source term for the CFD solver.

Restrictions:

Only for solvers that include implicit momentum exchange.

Related commands:

momCoupleModel

momCoupleModel_noCouple command

Syntax:

Defined in couplingProperties dictionary.

```
momCoupleModels
(
    off
);
```

Examples:

```
momCoupleModels
(
    off
);
```

Description:

The noCouple-model is a dummy momCoupleModel model providing an no momentum source term for the CFD solver.

Restrictions:

Only for solvers that include no momentum exchange, e.g. immersed boundary.

Related commands:

momCoupleModel

regionModel_allRegion command

Syntax:

Defined in couplingProperties dictionary.

```
regionModel allRegion;
```

Examples:

```
regionModel allRegion;
```

Description:

The allRegion-model is a region model including the whole CFD region for the coupling.

Restrictions: None.

Related commands:

regionModel

regionModel_differentialRegion command

Syntax:

Defined in couplingProperties dictionary.

```
regionModel differentialRegion;
```

Examples:

```
regionModel differentialRegion;
```

Description:

The differentialRegion-model is a region model including only a part of the CFD region for the coupling.

Restrictions:

This code is not up to date and cannot be used!

Related commands:

regionModel

regionModel command

Syntax:

Defined in couplingProperties dictionary.

```
regionModel model;
```

- model = name of the regionModel to be applied

Examples:

```
regionModel allRegion;
```

Note: This examples list might not be complete - please look for other models (regionModel_XY) in this documentation.

Description:

The regionModel is the base class for region models to select a certain region for coupled simulation.

Restrictions: none.

Default: none.

voidfractionModel_bigParticleVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel bigParticle;
bigParticleProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    scaleUpVol number3;
}
```

- *number1* = max number of cells covered by a partilce (search will fail when more than *number1* cells are covered by the particle)
- *number2* = mininmum limit for voidfraction
- *number3* = diameter of the particle's representation is artificially increased according to *number3* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel bigParticle;
bigParticleProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    scaleUpVol 5.0;
}
```

Description:

The bigParticle voidFraction model is supposed to be used when a particle (or it's representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Code of this sub-model contributed by Alice Hager, JKU.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_centreVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel centre;
centreProps
{
    alphaMin value;
}
```

- *value* = mininmum limit for voidfraction

Examples:

```
voidfractionModel centre;
centreProps
{
    alphaMin 0.1;
}
```

Description:

The centre voidFraction model calculates the voidfraction in a CFD cell accounting for the volume of the particles whose centres are inside the cell.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_dividedVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel divided;
dividedProps
{
    alphaMin number1;
    scaleUpVol number2;
}
```

- *number1* = mininmum limit for voidfraction
- *number2* = diameter of the particle's representation is artificially increased according to *number2* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel divided;
dividedProps
{
    alphaMin 0.2;
    scaleUpVol 1.0;
}
```

Description:

The divided voidFraction model is supposed to be used when a particle (or it's representation) is in the size range of a CFD cell. Satellite points are used to divide the particle's volume to the touched cells.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Restrictions: none.

Related commands:

voidfractionModel

voidfractionModel_dividedMSVoidFractionMS command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel dividedMS;
dividedMSProps
{
    alphaMin number1;
    scaleUpVol number2;
    clumpVol number3;
    nrigid number4;
}
```

- *number1* = mininmum limit for voidfraction
- *number2* = diameter of the particle's representation is artificially increased according to *number2* * Vparticle, volume remains unaltered!
- *number3* = diameter of the clump for the calculation of its volume
- *number4* = number of particle's a clump is composed of

Examples:

```
voidfractionModel dividedMS;
dividedMSProps
{
    alphaMin 0.05;
    scaleUpVol 1.0;
    clumpVol 1.8433e-9;
    nrigid 10;
}
```

Description:

The dividedMS voidFraction model is supposed to be used when a particle (or it's representation) is in the size range of a CFD cell. Satellite points are used to divide the particle's volume to the touched cells.

The volume of the clump is set to "clumpVol", not from the volume of the particles.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Restrictions:

For multisphere solvers.

Related commands:

voidfractionModel

voidfractionModel_GaussVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel Gauss;
GaussProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    scaleUpVol number3;
}
```

- *number1* = max number of cells covered by a partilce (search will fail when more than *number1* cells are covered by the particle)
- *number2* = mininmum limit for voidfraction
- *number3* = diameter of the particle's representation is artificially increased according to *number3* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel Gauss;
GaussProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    scaleUpVol 5.0;
}
```

Description:

The Gauss voidFraction model is supposed to be used when a particle (or it's representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle. The volume is here distributed according to a Gaussian dirstibution.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Restrictions: none.

Related commands:

voidfractionModel , bigParticle

voidfractionModel command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel model;
```

- model = name of the voidfractionModel to be applied

Examples:

```
voidfractionModel centre;
```

Note: This examples list might not be complete - please look for other models (voidfractionModel_XY) in this documentation.

Description:

The voidfractionModel is the base class for models to represent the DEM particle's volume in the CFD domain via a voidfraction field.

Restrictions: none.

Default: none.

voidfractionModel_IBVoidFraction command

Syntax:

Defined in couplingProperties dictionary.

```
voidfractionModel IB;
IBProps
{
    maxCellsPerParticle number1;
    alphaMin number2;
    scaleUpVol number3;
}
```

- *number1* = max number of cells covered by a partilce (search will fail when more than *number1* cells are covered by the particle)
- *number2* = mininmum limit for voidfraction
- *number3* = diameter of the particle's representation is artificially increased according to *number3* * Vparticle, volume remains unaltered!

Examples:

```
voidfractionModel IB;
IBProps
{
    maxCellsPerParticle 1000;
    alphaMin 0.10;
    scaleUpVol 5.0;
}
```

Description:

The IB voidFraction model is supposed to be used when a particle (or it's representation) is bigger than a CFD cell. The voidfraction field is set in those cell whose centres are inside the particle. The model is specially designed for cfdemSolverIB and creates a smooth transition of the voidfraction at the particle surface.

The region of influence of a particle can be increased artificially by "scaleUpVol", which blows up the particles, but keeps their volume (for voidfraction calculation) constant.

Code of this sub-model contributed by Alice Hager, JKU.

Restrictions: none.

Related commands:

voidfractionModel