# OpenSTREAM Manual

The OpenSTREAM Team

May 06, 2025

# USAGE

1 Welcome 1

2 Usage 3
- 2.1 Getting started 3
  - 2.1.1 Prerequisites and installation 3
- 2.2 Run sample script 4

3 Packages 7
- 3.1 InputEnums 7
- 3.2 Inputs 9
- 3.3 Session 18
- 3.4 Solvers 19
  - 3.4.1 FourField Solver 19
  - 3.4.2 Mixture Solver 25
  - 3.4.3 ThreeField Solver 29

4 ExternalPackages 37
- 4.1 CoolPropWrapper 37

Bibliography 39

MATLAB Module Index 41

Index 43

CHAPTER

ONE

# WELCOME

OpenSTREAM is a MATLAB program for simulating two-phase flow behaviors, inspired by Le Corre [2022].

Add more descriptive text.

The purpose of this documentation is to fully describe the usage and thought-process behind the design of the program.

The PDF version of this manual can be found here: OpenSTREAM.pdf

CHAPTER

# TWO

# USAGE

## 2.1 Getting started

### 2.1.1 Prerequisites and installation

OpenSTREAM is a MATLAB program for simulating two-phase flow behaviors. The Python version of CoolProp is used for calculating the fluid properties. CoolPropWrapper is a MATLAB interface that interacts with CoolProp through Python. Both OpenSTREAM and CoolProp are hosted here on GitHub. Making a copy (cloning) of these programs will be our first step.

#### Install using Git and Github

- There are two main ways of interacting with GitHub to use and contribute to the OpenSTREAM project: Git Command Line Interface (CLI), and the GitHub Desktop program. Use the links to set up one of the methods.
- After installing and setting up your Git mechanism of choice, if you chose to use the:
  - **Command line interface**
    - Clone (make a copy of) the **repo**sitory using: `git clone --recursive git@github.com:OpenSTREAM-solvers/openstream.git` (SSH authentication)

      or

      `git clone --recursive https://github.com/OpenSTREAM-solvers/openstream.git` (HTTPS authentication).

      (The `--recursive` flag is needed to clone submodules, such as the mfval/CoolPropWrapper, in one command.)
    - Then, use `cd openstream` to enter the newly created folder. You should see the newly downloaded files for the project.
  - **GitHub Desktop program**
    - Open the program, login, and **Clone repository**. Everything should be cloned automatically. This is part of the setup guide linked above.

#### MATLAB-Python compatibility

Before using the OpenSTREAM, we need to make sure Python with the appropriate version is installed, and MATLAB knows the location of the Python executable. The following instructions are from the CoolPropWrapper repo.

1. Determine which versions of Python are compatible with your version of MATLAB: compatibility list. The second column, *MATLAB Interface MATLAB Engine*, shows the compatible Python versions.
2. Start MATLAB and verify a compatible python version is installed:

- Run `pyenv()`
  - If the results are empty or a incompatible Python version is shown, go to step 3.
  - If the results are satisfactory, continue to step 5.

3. Install compatible Python3
   - **Windows**: Go to python.org to download the specific version of Python you need. Follow the setup procedure.
   - **Linux**: Chances are, you already have a version of Python installed. Verify the version using `python3 --version`. If Python is not installed, or a incompatible version is installed, install appropriate version using the OS package manager. For Debian-based Linux distros (such as Ubuntu), use `sudo apt-get install python3.x` to install the *x* version of Python3.
   - **MacOS**: Someone with a Mac, please write how you do this…
4. Specify Python installation location in MATLAB:
   - Run `pyenv('Version', pathtopython')`, where `pathtopython` is the path to where Python is installed. Here are some typical locations depending on your OS:
     - **Windows**: `C:\Users\Username\AppData\Local\Programs\Python\Python310\python.exe`, for version 3.10.
     - **Linux**: `/usr/bin/python3.10`, for version 3.10. Use `whereis python3` to find possible locations.
     - **MacOS**: Someone with a Mac, please write how you do this…
5. Try creating an instance of CoolPropWrapper() in MATLAB:
   - Run `cp=CoolPropWrapper()`.
     - If you are using the CoolPropWrapper as MATLAB Package in OpenSTREAM, use `cp=CoolPropWrapper.CoolPropWrapper()`.
   - If the CoolProp module is not installed, you will be prompted to do so automatically. If you prefer to do this manually, in your terminal, run `python3 -m pip install --user -U CoolProp`, replacing `python3` with the specific python path as necessary.

### Next steps

Congrats! Hopefully, at this point, you have successfully made a copy of this repo and installed the required programs. Next, let's run the Tutorial script to make sure everything is working properly.

## 2.2 Run sample script

At this point, you are ready to run the sample script. You will see some of the basic usages of OpenSTREAM.

```
% Clear existing variables from workspace
clearvars

% Import some of the package classes
import Inputs.*
import Solvers.*
import Solvers.Mixture.*

% InputSet is used to create a collection of input settings
inputSet = InputSet( ...
```

(continues on next page)

(continued from previous page)

```
            modelFilePath         = './inputs/models.inp',  modelID    = 'BARCS', ...
            optionsFilePath       = './inputs/options.inp', optionsID  = 'STEADY', ...
            geometryFilePath      = './inputs/geom.inp',    geometryID = 'BARC', ...
            bcFilePath            = './inputs/bc_barc.inp', ...
            sessionParentDir      = fullfile(pwd,'outputs'), ...
            overwriteSessionFiles = true, ...
            LOGMODE               = 'BOTH');

% Create a mixture solver
mixSolver = MixtureSolver(inputSet);

% mixSolver is initialized at construction. Here, it is explicitly initialized for␣
↪clarity.
mixSolver.initializeSolver();

% Solve (does not accept any argument)
mixSolver.solve();

% Axial and temporal plots
mixSolver.plotz(1);                                    % Axial plot at 1st time step
mixSolver.plotz(mixSolver.NTIME);                     % Axial plot at final time step
mixSolver.plott(mixSolver.NZ)                         % Temporal plot at last spatial␣
↪node
mixSolver.plott(mixSolver.NZ,'solveMode','STEADY')     % Temporal plot at last spatial␣
↪node of steady state solution
mixSolver.plotzt(mixSolver.NZ,"solveMode","TRANSIENT"); % Spatial-temporal map of␣
↪transient solution at channel exit

% Save results
mixSolver.saveResults(saveFormat="MAT");
```

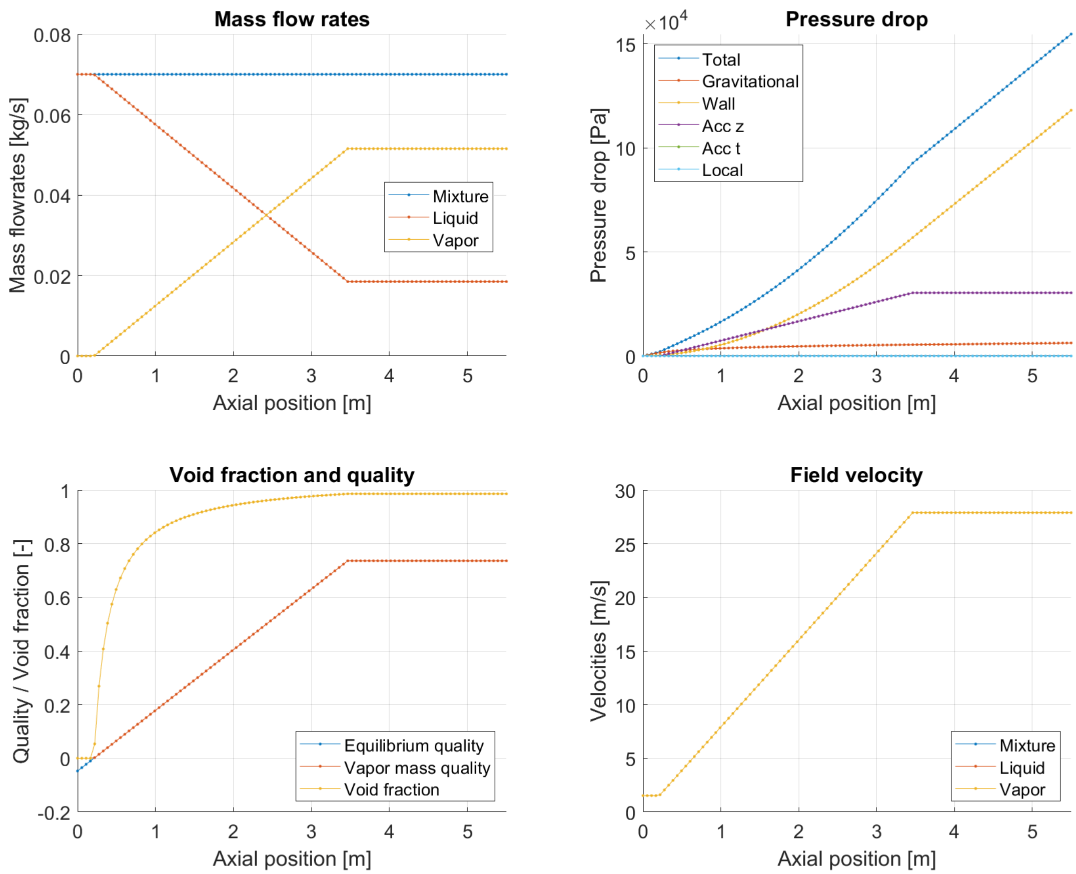


Fig. 1: Sample figure of axial distributions of mixture parameters at 0 [s].

CHAPTER

THREE

# PACKAGES

## 3.1 InputEnums

`class InputEnums.INTLENGTH`

INTLENGTH Interfacial length scale Defines the interfacial length scale model

`class InputEnums.VAPORFRIC`

VAPORFRIC Vapor to film friction model Detailed explanation goes here

`class InputEnums.BASEEQTHICK`

BASEEQTHICK Base film equilibrium thickness model Defines which base film equilibrium model to use

`class InputEnums.INTNU`

INTNU Interfacial Nusselt number Defines the interfacial Nusselt number model

`class InputEnums.MOMENTBASE`

MOMENTBASE Base film momentum conservation model Defines the type of momentum conservation used on the liquid film. The conservation equations are used to solve for liquid-film velocity

`class InputEnums.DROPDRAG`

DROPDRAG drop drag model Detailed explanation goes here

`class InputEnums.OAFFILMSPLIT`

OAFFILMSPLIT Film mass flow ratio at onset of annular flow Detailed explanation goes here

`class InputEnums.MOMENTWAVE`

MOMENTWAVE Film momentum conservation model Defines the type of momentum conservation used on the liquid film. The conservation equations are used to solve for liquid-film velocity

`class InputEnums.VOID`

VOID Void fraction model This defines the type of void fraction model used. The homogenous model assumes the velocity of the fields is the same. The slip model assigns a slip ratio. The Bestion model is a correlation.

`class InputEnums.DEPOSITION`

DEPOSITION Deposition models Defines which droplet deposition mass flux model is used. NOTE: the entrainment and deposition correlations are coupled, i.e. the same model should be used for both entrainment and deposotion mass flux

`class InputEnums.ENTRAINMENT`

ENTRAINMENT Entrainment models Defines which droplet entrainment mass flux model is used. NOTE: the entrainment and deposition correlations are coupled, i.e. the same model should be used for both entrainment and deposotion mass flux `class InputEnums.MOMENTDROP`

MOMENTDROP Drop momentum conservation model This defines the type of conservation model used on the droplets in the Drop class.

`class InputEnums.FLUIDPROPERTIES`

FLUIDPROPERTIES Fluid property calculation assumptions Defines the pressure to be used for calculating fluid properties

`class InputEnums.LOCRELVEL`

LOCRELVEL Local relative velocity between liquid and vapor phase Defines the model for the local relative velocity

`class InputEnums.TPFM`

TPFM Two-phase friction multiplier Detailed explanation goes here

`class InputEnums.INTTRANSH`

INTTRANSH Interfacial enthalpy transfer Defines the interfacial enthalpy exchange model from interfacial mass transfer The mass can be transfered either at phase bulk enthalpy, or at saturated enthaply

`class InputEnums.THINFILMFRIC`

THINFILMFRIC Thin film friction model Detailed explanation goes here

`class InputEnums.MOMENTLIQUID`

MOMENTLIQUID Liquid momentum conservation model This defines the type of conservation model used on the liquid phase in the liquid class.

`class InputEnums.OAFENTRAINED`

OAFENTRAINED Entrained model at onset of annular flow Detailed explanation goes here

`class InputEnums.INTAREA`

INTAREA Interfacial area Defines the interfacial area model

`class InputEnums.MOMENTFILM`

MOMENTFILM Film momentum conservation model Defines the type of momentum conservation used on the liquid film. The conservation equations are used to solve for liquid-film velocity

`class InputEnums.OAF`

OAF Onset of annular flow model Defines the model to predict the position at which the onset of annualr flow occurs based on local flow conditions. Wallis correlation is Equation 11.1019 in One-dimensional two-phase flow. Wallis (1969). The correlation was built using air-water data.

`class InputEnums.SCBOIL`

SCBOIL Subcooled boiling model Detailed explanation goes here

`class InputEnums.BOILCOEF`

BOILCOEF Wall boiling coefficient interpolation for subcooled boiling Detailed explanation goes here

`class InputEnums.MOMENTGAS`

MOMENTGAS Gas momentum conservation model This defines the type of conservation model used on the gas phase in the gas class.

`class InputEnums.WAVEFREQUENCY`

WAVEFREQUENCY Wave number conservation model This defines the type of conservation model used on the wave number in the Wave class.

`class InputEnums.EQSTROUHAL`

EQSTROUHAL Equilibrium Strouhal number Defines which base film equilibrium model to use `class InputEnums.TPKM`

TPKM Two-phase local loss multiplier Detailed explanation goes here

## 3.2 Inputs

*Inputs* is pretty cool.

`class Inputs.InputSet(opts)`

INPUTSET Creates set of input objects Detailed explanation goes here

**Constructor Summary**

`InputSet(opts)`

INPUTSET Construct an instance of this class Detailed explanation goes here

`class Inputs.FluidProperties(P, modelObj)`

FLUIDPROPERTIES Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

`FluidProperties(P, modelObj)`

FLUIDPROPERTIES Construct an instance of this class Detailed explanation goes here

**Property Summary**

`FLUID = 'WATER'`

Fluid ID

`PROPERTIES = 'SATURATED'`

Fluid property assumptions

`PRESSURE = 1`

[Pa] System pressure

`TSAT = 1`

[K] Saturated fluid temperature

`RHOF = 1`

[kg/m^3] Saturated liquid mass density

`RHOG = 1`

[kg/m^3] Saturated vapor mass density

`MUF = 1`

[Pa.s] Saturated liquid viscosity

`MUG = 1`

[Pa.s] Saturated vapor viscosity

`HF = 1`

[J/kg] Saturated liquid enthalpy

`HG = 1`

[J/kg] Saturated vapor enthalpy

`HFG = 0`

[J/kg] Latent heat of evaporation

`SIGMA = 1`
: [N/m] Surface tension

`KF = 1`
: [W/m/K] Saturated liquid conductivity

`KG = 1`
: [W/m/K] Saturated vapor conductivity

`PRANDTLF = 1`
: [-] Saturated liquid Prandtl number

`PRANDTLG = 1`
: [-] Saturated vapor Prandtl number

**Method Summary**

`T(H)`
: `T()` Fluid temperature [K] at obj.PRESSURE and H

`H(T)`
: `H()` Fluid enthalpy [J/kg] at obj.PRESSURE and T

`RHOL(H)`
: `RHOL()` Liquid mass density (subcooled to saturated) [kg/m^3]

`RHOV(H)`
: `RHOV()` Vapor mass density (saturated to superheated) [kg/m^3]

`MUL(H)`
: `MUL()` Liquid dynamic viscosity [Pa-s]

`MUV(H)`
: `MUV()` Vapor dynamic viscosity [Pa-s]

`KL(H)`
: `KL()` Liquid conductivity (subcooled to saturated) [W/m/K]

`KV(H)`
: `KV()` Vapor conductivity (saturated to superheated) [W/m/K]

`PRANDTLL(H)`
: `PRANDTLL()` Liquid Prandtl number (subcooled to saturated) [-]

`PRANDTLV(H)`
: `PRANDTLV()` Vapor Prandtl number (saturated to superheated) [-]

`plot(H)`
: PLOT Plot properties for given enthalpy vector

`class Inputs.BoundaryConditions(filePath, geometryObjInput)`

Bases: `Inputs.Input`

BOUNDARYCONDITIONS Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**BoundaryConditions**(*filePath*, *geometryObjInput*)
BOUNDARYCONDITIONS Construct an instance of this class Detailed explanation goes here

**Property Summary**

**TIME**
Time [s]

**PRESSURE = []**
System pressure [Pa]

**HIN = []**
Inlet enthalpy [J/kg]

**MFLOW = []**
Mass flow rate [kg/s]

**POWER = 0**
Total power [W]

**WMESH = 1**
Relative power node size distribution [m]

**WPOWER = 1**
Relative power distribution(s) [-]

**Method Summary**

**TSAT**(*fluidObj*)
*TSAT()* [K] Saturation temperature given fluidObj

**HF**(*fluidObj*)
*HF()* [J/kg] Liquid saturation given fluidObj

**HG**(*fluidObj*)
HF [J/kg] Vapor saturation given fluidObj

**TIN**(*fluidObj*)
*TIN()* [K] Inlet temperature

**DTIN**(*fluidObj*)
*DTIN()* [K] Inlet subcooling temperature difference given fluidObj

**DHIN**(*fluidObj*)
*DHIN()* [J/kg] Inlet subcooling enthalpy difference given fluidObj

**XIN**(*fluidObj*)
*XIN()* [-] Inlet equilibrium quality

**plot**(*fluidObj*)
PLOT Plot boundary conditions usage: bc.plot(fluidObj)

**class** Inputs.**Geometry**(*filePath*, *geometryID*)
Bases: *Inputs.Input*

GEOMETRY Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Geometry**(*filePath*, *geometryID*)

MODEL Construct an instance of this class Detailed explanation goes here

**Property Summary**

**ID**

Channel *ID*

**LENGTH = 1**

Axial length [m]

**AREA = 1**

Coolant area [m^2]

**PERIM = 1**

Perimeters [m]

**ANGLE = 0**

Angle [rad]

**Method Summary**

**HDIAM**()

*HDIAM()* Hydraulic diameter

**NWALL**()

*NWALL()* Number of walls

**class** Inputs.**Input**(*inputFilePath*, *key*, *val*)

Bases: `dynamicprops`

INPUT Su Detailed explanation goes here

**Constructor Summary**

**Input**(*inputFilePath*, *key*, *val*)

INPUT Parse inputFile to construct this class Detailed explanation goes here

**Method Summary**

**validateInputEntry**(*objPropname*, *opts*)

VALIDATEINPUTENTRY Description

**listInputProperties**(*opts*)

LISTINPUTPROPERTIES List of protected obj property names

**defaultValueUsedReport**(*propNames*, *propValues*)

DEFAULTVALUEUSEDREPORT Report of properties in which the default values were used.

**static convert2JSON**(*inputFilePath*)

Read inputFilePath

**class** Inputs.**Options**(*filePath*, *optionsID*)

Bases: *Inputs.Input*

OPTIONS Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

`Options`(*filePath*, *optionsID*)

MODEL Construct an instance of this class Detailed explanation goes here

**Property Summary**

`ID`

Option *ID*

`TSTEP = 0.1`

Time step [s]

`MAXITER = 100`

Max number of inner (point) iterations

`ERRORW = 1E-3`

Mass flow rate error target in inner iterations [kg/s]

`ERRORU = 1E-4`

Velocity error target in inner iterations [m/s]

`ERRORP = 1E+0`

Pressure error target in inner ierations [Pa]

`ERRORH = 1E+0`

Enthalpy error target in inner ierations [J/kg]

`SSTSTEP = 1.0`

Time step for steady-state iterations [s]

`SSMAXITER = 20`

Max number of steady-state iterations

`SSCONVW = 1E-3`

Mass flow rate steady-state convergence criterion [kg/s]

`SSCONVU = 1E-3`

Velocity steady-state convergence criterion [m/s]

`SSCONVP = 1E+0`

Pressure steady-state convergence criterion [Pa]

`SSCONVH = 1E+0`

Enthalpy steady-state convergence criterion [J/kg]

`AXIALINTERP = 'next'`

Axial power interpolation method

`TIMEINTERP = 'linear'`

Time-dependant boundary conditions interpolation method

`RELAXWM = 1`

Relaxation factor for the mixture mass conservation equation

`RELAXPM = 1`

Relaxation factor for the mixture momentum conservation equation

`RELAXHM = 1`

Relaxation factor for the mixture energy conservation equation

`RELAXWL = 1`
: Relaxation factor for the liquid mass conservation equation

`RELAXWV = 1`
: Relaxation factor for the vapor mass conservation equation

`RELAXUL = 1`
: Relaxation factor for the liquid momentum conservation equation

`RELAXUV = 1`
: Relaxation factor for the vapor momentum conservation equation

`RELAXHL = 1`
: Relaxation factor for the liquid mass conservation equation

`RELAXHV = 1`
: Relaxation factor for the vapor energy conservation equation

`ERRORWF = 1E-4`
: Film mass flow rate error target in inner iterations [kg/s/m]

`ERRORUF = 1E-2`
: Film velocity error target in inner iterations [m/s]

`ERRORUD = 1E-2`
: Drop velocity error target in inner iterations [m/s]

`SSCONVWF = 1E-4`
: Film mass flow rate steady-state convergence criterion [kg/s/m]

`SSCONVUF = 1E-2`
: Film velocity steady-state convergence criterion [m/s]

`SSCONVUD = 1E-2`
: Drop velocity steady-state convergence criterion [m/s]

`RELAXWF = 0.5`
: Relaxation factor for the film energy conservation equation

`RELAXUF = 0.2`
: Relaxation factor for the film momentum conservation equation

`RELAXUD = 0.2`
: Relaxation factor for the drop momentum conservation equation

`RELAXWB = 0.5`
: Relaxation factor for the base mass conservation equation

`RELAXUB = 0.2`
: Relaxation factor for the base momentum conservation equation

`RELAXWW = 0.5`
: Relaxation factor for the wave mass conservation equation

`RELAXUW = 0.2`
: Relaxation factor for the wave momentum conservation equation

`RELAXFW = 0.5`

Relaxation factor for the wave number conservation equation

**class** `Inputs.Model`(*filePath*, *modelID*)

Bases: `Inputs.Input`

MODEL Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

`Model`(*filePath*, *modelID*)

MODEL Construct an instance of this class Detailed explanation goes here

**Property Summary**

`ID`

Model *ID*

`NNODES = []`

Number of axial nodes

`FLUID = "WATER"`

Fluid ID

`PROPERTIES = 'SATURATED'`

Fluid property assumptions

`ANGLE = 0`

Flow axis angle from vertical [deg]

`FRICTION = [0.2-0.20]`

Wall friction coefficients

`TPFM = 'HOMOGENEOUS'`

Two-phase friction multiplier [-]

`KLOC = [00]`

Elevation of local perturbations [m]

`KLOSS = [00]`

corresponding pressure loss coefficients [-]

`TPKM = 'HOMOGENEOUS'`

Two-phase local loss multiplier [-]

`SCBOIL = 'NONE'`

Subcooled boiling mode

`VOID = 'HOMOGENEOUS'`

Void fraction model

`SLIP = 1`

Phase velocity ratio [-]

`MOMENTLIQUID = 'SLIP'`

Liquid momentum conservation model [-]

`MOMENTGAS = 'SLIP'`
: Gas momentum conservation model [-]

`BOILCOEF = 'QUADRATIC'`
: Subcooled boiling interpolation [-]

`INTLENGTH = 'CONSTANT'`
: Interfacial length scale model model [-]

`INTAREA = 'DISPGAS2DISPLIQ'`
: Interfacial area model [-]

`INTLENGTHCST = 2E-3`
: Constant interfacial length scale [m]

`INTTRANSH = 'BULK'`
: Interfacial enthalpy transfer [-]

`INTNU = 'CONSTANT'`
: Interfacial heat transfer model [-]

`INTNUVCST = 2.0`
: Dispersed gas constant interfacial Nusselt number [-]

`INTNULCST = 2.0`
: Dispersed liquid constant interfacial Nusselt number [-]

`RANZMARSHALLVCST = [20.61/21/3]`
: Dispersed gas Ranz-Marshall coefficients [-]

`RANZMARSHALLLCST = [20.61/21/3]`
: Dispersed liquid Ranz-Marshall coefficients [-]

`LOCRELVEL = 'SCALED'`
: Local relative velocity model [-]

`RELVELCST = 1E-1`
: Multiplication factor to determine the localrelative velocity [-]

`RELAXTCOND = 0.2`
: Condensation relaxation time [s]

`RELAXTEVAP = 0.2`
: Evaporation relaxation time [s]

`OAF = 'WALLIS'`
: Onset of annular flow model [-]

`OAFFILMSPLIT = 'RATIO'`
: Film mass flow rate at onset of annular flow [-]

`OAFBASERATIO = 0.5`
: Base/Film mass ratio at onset of annular flow [-]

`OAFENTRAINED = 'RATIO'`
: Entrained model at onset of annular flow [-]

`OAFDROPRATIO = 0.7`
: Drop/Liquid mass ratio at onset of annular flow [-]

`OAFTRANSITION = [0.100.0]`
: Annular flow transition function parameters (sigmoid width/location wrt OAF) [m]

`DEPOSITION = 'GOVAN'`
: Drop deposition model [-]

`ENTRAINMENT = 'GOVAN'`
: Film entrainment model [-]

`MOMENTFILM = 'ALGEBRAIC'`
: Film momentum conservation model [-]

`MOMENTDROP = 'SLIP'`
: Drop momentum conservation model [-]

`DROPSLIP = 1.0`
: Drop velocity ratio [-]

`THINFILMFRIC = 'TURBULENT'`
: Thin film wall friction model [-]

`THINFILMTHICK = 1E-4`
: Thin film thickness [m]

`VAPORFRIC = 'WALLIS'`
: Vapor friction model [-]

`VAPORFRICCST = 0.005`
: Vapor friction constant [-]

`POSFILM = true`
: Keep positive film flowrate/thickness

`DROPDIAM = 1E-3`
: Drop diameter [mm]

`DROPDRAG = 'CONSTANT'`
: Drop drag model

`DROPDRAGCOEF = 0.45`
: Drop drag coefficient

`BASEEQTHICKCOEF = [5.37E-5-0.641.21]`
: Base equilibrium thickness coefficient

`RELAXTB = 0.2`
: Base film relaxation time

`MOMENTBASE = 'ALGEBRAIC'`
: Base Film momentum conservation model [-]

`SHAPEFACTORCOEF = [1.325E52]`
: Wave shape factor coefficients

**EQSTROUHALCOEF = [1.1236E-40.5]**
Wave equilibrium Strouhal coefficients

**WAVEDRAGCOEF = [0.021.350E50.437]**
Wave drag coefficient

**WAVEFREQUENCY = 'RELAXATION'**
Wave number conservation model

**RELAXTW = 0.2**
Wave relaxation time

**MOMENTWAVE = 'FULL'**
Film momentum conservation model [-]

**WAVEMIXCOEF = 0.0**
Wave mixing coefficient

**G = 9.81**
[m/s^2] Gravitational acceleration

## 3.3 Session

`Session.`**`isLegalPath`**(*pathname*)
ISLEGALPATH Determine if a path is legal in current OS

**class** `Session.`**`Session`**(*opt*)
Bases: `handle`

SESSION Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**`Session`**(*opt*)
SESSION Construct an instance of this class Detailed explanation goes here

**Property Summary**

**showWarnings = true**
Warnings

**Method Summary**

**`setupLog`**(*session*, *LOGMODE*, *opts*)
SETUPLOG Setup log

**`makeSessionDirectory`**()
MAKESESSIONDIRECTORY Make the session directory

Check if session directory is legal and/or exists

**class** `Session.`**`Log`**(*LOGMODE*, *opts*)
Bases: `handle`

LOG Console log manager Detailed explanation goes here

**Constructor Summary**

**Log**(*LOGMODE*, *opts*)

LOG Construct an instance of this class Detailed explanation goes here

**Property Summary**

**LOGFID = -1**

log file

**showWarnings = true**

Warnings

**Method Summary**

**log**(*varargin*)

LOG Log events

**warning**(*varargin*)

WARNING Log warnings

**diaryOff**()

DIARYOFF Turn diary off

**diaryOn**()

DIARYON Turn diary on

**delete**()

DELETE Deconstructor of Log

**openLog**(*opts*)

SETUPLOG Open/Setup the logging file

**closeLog**()

CLOSELOG Close log file reference

**class** `Session.`**LogMode**

Bases: `uint16`

LOGMODE Enumeration class of possible log modes

## 3.4 Solvers

### 3.4.1 FourField Solver

`FourField Solver` is cooler.

**class** `Solvers.FourField.`**Wave**(*film*)

Bases: *`Solvers.AbstractFilm`*

WAVE Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Wave**(*film*)

WAVE Creates a *`Wave()`*, wave Detailed explanation goes here

**Property Summary**

**NZ = 0**
: % Solver properties

**NTIME = 0**
: [-] Number of time steps

**TIME = 0**
: [s] Time series

**DT = 0**
: [s] Time step size

**TIDX = 1**
: [-] Time step index

**Z = 1.**
: [m] Elevation

**HFLUX = 1.**
: [W/m^2] Film heat flux

**W = 1.**
: Flow properties

**U = 1.**
: [m/s] Velocity

**H = 1E6**
: [J/kg] Enthalpy

**FREQUENCY = 1**
: [Hz] Wave frequency

**ITR**
: % Iteration properties

**Method Summary**

**WL**(*wave*, *zIdx*)
: *WL()* wave mass flow rate per unit perimeter

**THICK**(*wave*, *zIdx*)
: *THICK()* wave thickness

**BETA**(*wave*, *zIdx*)
: *BETA()* Wave film interfacial fraction

**EPSILON**(*wave*, *zIdx*)
: BETA Wave mass flow fraction

**BETAP**(*wave*, *zIdx*)
: *BETAP()* Wave film heat flux fraction

**ETA**(*wave*, *zIdx*)
: *ETA()* Wave film deposition fraction

**MENT**(*wave*, *zIdx*)
: *MENT()* Wave entrainment mass flux

**MEVAP**(*wave*, *zIdx*)
: *MEVAP()* Wave evaporation mass flux

**MTURB**(*wave*, *zIdx*)
: *MTURB()* Turbulent mass exchange

**MBASE**(*wave*, *drop*, *zIdx*)
: MWAVE Mass flux interaction with base

**MTOT**(*wave*, *drop*, *zIdx*)
: *MTOT()* Total mass flux

**FWALL**(*wave*, *zIdx*)
: *FWALL()* Film wall shear stress

**FBASE**(*wave*, *zIdx*)
: *FBASE()* Base interfacial force

**FBASEMASS**(*wave*, *drop*, *zIdx*)
: *FBASEMASS()* Wave mass exchange force

**FVAPOR**(*wave*, *zIdx*)
: *FVAPOR()* Film vapor shear stress

**FDRAG**(*wave*, *zIdx*)
: *FDRAG()* Film vapor shear stress due to drag

**FSHEAR**(*wave*, *zIdx*)
: *FSHEAR()* Film vapor shear stress

**FTOT**(*wave*, *drop*, *zIdx*)
: *FTOT()* Total

**SHAPEFACTOR**(*wave*, *zIdx*)

**EQSTROUHAL**(*wave*, *zIdx*)
: STROUHAL Correlation of equilibrium Strouhal number

**EQFREQUENCY**(*wave*, *zIdx*)

**EQPERIOD**(*wave*, *zIdx*)
: *EQPERIOD()* Inverse of EQFREQ

**SPACING**(*wave*, *zIdx*)
: *SPACING()* Wave spacing

**PERIOD**(*wave*, *zIdx*)
: *PERIOD()* Wave time period Inverse of wave frequency

**N**(*wave*, *zIdx*)
: *N()* Wave number density Inverse of wave spacing

**WIDTH**(*wave*, *zIdx*)
: *WIDTH()* Wave width Function of amplitude, Shape factor

**AMP**(*wave*, *zIdx*)
: *AMP()* Wave amplitude Function of WL, rho, Shape factor, frequency

**DELTAFREQ**(*wave*, *zIdx*)

*DELTAFREQ()* Wave frequency source/sink

**DRAGCOEF**(*wave*, *zIdx*)

*DRAGCOEF()* Wave drag coefficient

**class** Solvers.FourField.**Drop**

Bases: *Solvers.ThreeField.Drop*

DROP Summary of this class goes here Detailed explanation goes here

**class** Solvers.FourField.**Base**(*film*)

Bases: *Solvers.AbstractFilm*

BASE Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Base**(*film*)

BASE Creates a *Base()*, base Detailed explanation goes here

**Property Summary**

**NZ = 0**

% Solver properties

**NTIME = 0**

[-] Number of time steps

**TIME = 0**

[s] Time series

**DT = 0**

[s] Time step size

**TIDX = 1**

[-] Time step index

**Z = 1.**

[m] Elevation

**HFLUX = 1.**

[W/m^2] Film heat flux

**W = 1.**

Flow properties

**U = 1.**

[m/s] Velocity

**H = 1E6**

[J/kg] Enthalpy

**ITR**

% Iteration properties

**Method Summary**

**EPSILON**(*base*, *zIdx*)
BETA Base film mass flow fraction

**BETA**(*base*, *zIdx*)
*BETA()* Base film interfacial fraction

**BETAP**(*base*, *zIdx*)
*BETAP()* Base film heat flux fraction

**ETA**(*base*, *zIdx*)
*ETA()* Base film deposition fraction

**MENT**(*base*, *zIdx*)
*MENT()* Base entrainment mass flux TODO: use coefficient later

**MEVAP**(*base*, *zIdx*)
*MEVAP()* Base evaporation mass flux

**MTURB**(*base*, *zIdx*)
*MTURB()* Turbulent mass exchange

**MWAVE**(*base*, *drop*, *zIdx*)
*MWAVE()* Net Mass flux interaction with wave Defined as a source term, Mwave is the

**MTOT**(*base*, *drop*, *zIdx*)
*MTOT()* Total mass flux of the base film

**FWAVE**(*base*, *zIdx*)
*FWAVE()* Wave interfacial force

**FWAVEMASS**(*base*, *drop*, *zIdx*)
*FWAVEMASS()* Base mass exchange force

**FBASEVAPOR**(*base*, *zIdx*)
*FBASEVAPOR()* Film base vapor shear stress TODO: consider different way of doing this…

**FDEP**(*base*, *drop*, *zIdx*)
FWAVE Droplet

**FTOT**(*base*, *drop*, *zIdx*)
*FTOT()* Total

**EQTHICK**(*base*, *zIdx*)
*EQTHICK()* Base equilibrium thickness

**TBASE**(*base*, *zIdx*)
*TBASE()* Base film exposed time The time in which the base film is exposed to the vapor (eq.24)

**TDRY**(*base*, *drop*, *zIdx*)
*TDRY()* Time duration in which the base is expected to dry out. Not included in the reference paper.

**FDRY**(*base*, *drop*, *zIdx*)
*FDRY()* Dry fraction

**WMIN**(*base*, *drop*, *zIdx*)
*WMIN()* Minimum base film mass flow rate (eq. 27) Base film mass flow rate at the end of the intermittent exposed time (TBASE).

**WMINL**(*base*, *drop*, *zIdx*)

*WMINL()* Minimum base film mass flow rate per perimeter

**THICKMIN**(*base*, *drop*, *zIdx*)

*THICKMIN()* Minimum base film thickness (eq. 28)

**copyFlowProperties**(*srcObj*, *targetObj*, *opts*)

COPYFLOWPROPERTIES

**class** Solvers.FourField.**Film**(*inputSet*, *fluid*)

Bases: *Solvers.AbstractFilm*

FILM Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Film**(*inputSet*, *fluid*)

FILM Creates a *Film()*, film Detailed explanation goes here

**Property Summary**

**NZ = 0**

Solver properties

**NTIME = 0**

[-] Number of time steps

**TIME = 0**

[s] Time series

**DT = 0**

[s] Time step size

**TIDX = 1**

[-] Time step index

**Z = 1.**

[m] Elevation

**HFLUX = 1.**

[W/m^2] Film heat flux

**MEVAP = -1.**

[kg/s/m^2] Evaporation mass flux

**ITR**

Iteration properties

**W**

Flow properties

**U**

) double {mustBeNumeric} = 1. % [m/s] Velocity

**Type**

(

**Type**

,

**H**

) double {mustBeNumeric} = 1E6 % [J/kg] Enthalpy

**Type**

(

**Type**

,

**Method Summary**

**distributeOAFW**(*film*, *W*, *zIdx*)

DISTRIBUTEOAFW Distributes film mass flow rate at onset of annular flow to base and wave, based on option: OAFFILMSPLIT

**EQUIL**(*film*, *Wf*, *zIdx*)

*EQUIL()* calculate base/wave split ratio in equilibrium

**initializeBaseAndWave**(*film*, *WIN*, *ITR*)

INITIALIZEBASEANDWAVE Initialize base and wave arrays given WTOT

**copyFlowProperties**(*srcObj*, *targetObj*, *opts*)

COPYFLOWPROPERTIES

### 3.4.2 Mixture Solver

`Mixture Solver` is pretty cool.

**class** Solvers.Mixture.**Liquid**(*mix*)

Bases: *Solvers.AbstractPhase*

LIQUID Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Liquid**(*mix*)

LIQUID Construct an instance of this class Detailed explanation goes here

**Method Summary**

**TIME**(*liquid*)

*TIME()* Time series [s] Detailed explanation goes here

**Z**(*liquid*, *zIdx*)

*Z()* Axial nodes [m] Detailed explanation goes here

**X**(*liquid*, *zIdx*)

*X()* Mass fraction [-]

**VF**(*liquid*, *zIdx*)

*VF()* Void fraction [-]

**W**(*liquid*, *zIdx*)

*W()* Mass flow rate [kg/s]

**U**(*liquid*, *zIdx*)

*U()* Velocity [m/s] NOTE: need to be verified

**H**(*liquid*, *zIdx*)

*H()* Enthalpy [J/kg]

**MFLUX**(*liquid*, *zIdx*)
: *MFLUX()* Mass flux [kg/m^2-s]

**RE**(*liquid*, *zIdx*)
: *RE()* Reynolds number [-]

`class Solvers.Mixture.`**`Vapor`**(*mix*)

Bases: `Solvers.AbstractPhase`

VAPOR Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Vapor**(*mix*)
: VAPOR Construct an instance of this class Detailed explanation goes here

**Method Summary**

**TIME**(*vapor*)
: *TIME()* Time series [s] Detailed explanation goes here

**Z**(*vapor*, *zIdx*)
: *Z()* Axial nodes [m] Detailed explanation goes here

**X**(*vapor*, *zIdx*)
: *X()* Mass fraction [-]

**VF**(*vapor*, *zIdx*)
: *VF()* Void fraction [-]

**W**(*vapor*, *zIdx*)
: *W()* Mass flow rate [kg/s]

**U**(*vapor*, *zIdx*)
: *U()* Velocity [m/s] NOTE: need to be verified

**H**(*vapor*, *zIdx*)
: *H()* Enthalpy [J/kg]

**MFLUX**(*vapor*, *zIdx*)
: *MFLUX()* Mass flux [kg/m^2-s]

**RE**(*vapor*, *zIdx*)
: *RE()* Reynolds number [-]

`class Solvers.Mixture.`**`Mixture`**(*inputSet*, *fluid*)

Bases: `Solvers.AbstractField`

MIXTURE Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Mixture**(*inputSet*, *fluid*)
: MIXTURE Creates a *Mixture()*, mix Detailed explanation goes here

**Property Summary**

**NZ = 0**
: Solver properties

**NTIME = 0**
: [-] Number of time steps

**TIME = 0**
: [s] Time series

**DT = 0**
: [s] Time step size

**TIDX = 1**
: [-] Time step index

**Z = 1.**
: [m] Elevation

**HFLUX = 1.**
: [W/m^2] Wall heat flux

**W = 1.**
: Flow properties

**P = 7E6**
: [Pa] Pressure

**H = 1E6**
: [J/kg] Enthalpy

**DP**
: [-] Detailed pressure drops

**ITR**
: Iteration properties

**liquid**
: Phases

**DZ = 0**
: [m] Axial step size

**Method Summary**

**MFLUX**(*mix*, *zIdx*)
: *MFLUX()* Mass flux [kg/m^2-s]

**XEQ**(*mix*, *zIdx*)
: *XEQ()* Equilibrium quality [-] This function only retrieves the mix.xeq values pre-calculated when mix.H is set. This is to eliminate the redundant calculation as a result of requent function calls. The values are calculated via mix.XEQ_CALC()

**X**(*mix*, *zIdx*)
: *X()* Vapor quality [-] This function only retrieves the mix.x values pre-calculated when mix.H is set. This is to eliminate the redundant calculation as a result of requent function calls. The values are calculated via mix.X_CALC()

**VF**(*mix*, *zIdx*)
*VF()* Void fraction [-]

**RHO**(*mix*, *zIdx*)
*RHO()* Density [kg/m^3]

**MU**(*mix*, *zIdx*)
*MU()* Dynamic viscosity [Pa-s]

**U**(*mix*, *zIdx*)
*U()* Velocity [m/s]

**JL**(*mix*, *zIdx*)
*JL()* Superfacial liquid velocity [m/s]

**JG**(*mix*, *zIdx*)
*JG()* Superfacial vapor velocity [m/s]

**RE**(*mix*, *zIdx*)
*RE()* Reynolds number [-]

**REL**(*mix*, *zIdx*)
*REL()* Liquid-equivalent Reynolds number [-]

**FW**(*mix*, *zIdx*)
*FW()* Wall friction factor [-]

**TAUW**(*mix*, *zIdx*)
*TAUW()* wall shear stress [Pa]

**KLOSS**(*mix*, *zIdx*)
*KLOSS()* Local pressure loss coefficient [-] TODO: NEED TO BE VERIFIED

**DPGRAV**(*mix*, *zIdx*)
DPK Gravitational pressure loss [Pa]

**DPWALL**(*mix*, *zIdx*)
*DPWALL()* Wall friction pressure drop [Pa]

**DPACCZ**(*mix*, *zIdx*)
*DPACCZ()* Spatial acceleration pressure drop [Pa]

**DPACCT**(*mix*, *Uold*, *zIdx*)
*DPACCT()* Temporal acceleration pressure drop [Pa]

**DPK**(*mix*, *zIdx*)
*DPK()* Local pressure loss [Pa]

**DPTOT**(*mix*, *Uold*, *zIdx*)
*DPTOT()* Total pressure loss [Pa]

**DPPARTS**(*mix*, *Uold*, *zIdx*)
*DPPARTS()* Pressure loss components[Pa]

**T**(*mix*, *zIdx*)
*T()* Temperature [K]

**OAFIDX**(*mix*)
: *OAFIDX()* Onset of annular flow node

**OAFZ**(*mix*)
: *OAFZ()* Onset of annular flow elevation

**OAFWL**(*mix*)
: *OAFWL()* Liquid mass flow rate at onset of annular flow

**AFFNC**(*mix*, *zIdx*)
: *AFFNC()* Annular flow function

**AFDISTR**(*mix*, *param1*, *param2*, *zIdx*)
: *AFDISTR()* Annular flow distribution function

### 3.4.3 ThreeField Solver

`ThreeField Solver` is pretty cool.

**class** `Solvers.ThreeField.`**Drop**(*inputSet*, *fluid*)

Bases: *Solvers.AbstractField*

DROP Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

**Drop**(*inputSet*, *fluid*)
: DROP Creates a *Drop()* ?solver? drop Detailed explanation goes here

**Property Summary**

**NZ = 0**
: Solver properties

**NTIME = 0**
: [-] Number of time steps

**TIME = 0**
: [s] Time series

**DT = 0**
: [s] Time step size

**TIDX = 1**
: [-] Time step index

**Z = 1.**
: [m] Elevation

**W = 1.**
: Flow properties

**U = 1.**
: [m/s] Velocity

**H = 1E6**
: [J/kg] Enthalpy

**ITR**
Iteration properties

**mix = NaN**
Mixture

**Method Summary**

**CONC**(*drop*, *zIdx*)
Drop concentration

**MDEP**(*drop*, *zIdx*)
Drop deposition mass flux

**RE**(*drop*, *zIdx*)
*RE()* Reynolds number [-]

**DIAM**(*drop*, *zIdx*)
*DIAM()* drop diameter

**AREA**(*drop*, *zIdx*)
*AREA()* drop interfacial area

**VOLUME**(*drop*, *zIdx*)
*VOLUME()* drop volume

**DENSITY**(*drop*, *zIdx*)
*DENSITY()* drop number density

**AI**(*drop*, *zIdx*)
*AI()* drop volumetric interfacial area

**DRAG**(*drop*, *zIdx*)
*DRAG()* drop drag coefficient

**FBUOY**(*drop*, *zIdx*)
*FBUOY()* Drop buoyancy

**FGRAV**(*drop*, *zIdx*)
*FGRAV()* Drop gravity

**FDRAG**(*drop*, *zIdx*)
FVAPOR drop vapor drag

**FENT**(*drop*, *film*, *zIdx*)
*FENT()* Film entrainment shear

**FTOT**(*drop*, *film*, *zIdx*, *opt*)
*FTOT()* Total

**USLIP**(*drop*, *zIdx*)
Slip drop velocity model

**UALGEBR**(*drop*, *film*, *zIdx*)
Algebraic drop velocity model (consistent with mixture model)

**UEQUIL**(*drop*, *film*, *zIdx*, *opt*)
*UEQUIL()* Drop velocity based on equilibrium model (Ftot = 0)

**class** `Solvers.ThreeField.`**`Film`**

Bases: *`Solvers.AbstractFilm`*

FILM Summary of this class goes here Detailed explanation goes here

**Property Summary**

**`NZ = 0`**

Solver properties

**`NTIME = 0`**

[-] Number of time steps

**`TIME = 0`**

[s] Time series

**`DT = 0`**

[s] Time step size

**`TIDX = 1`**

[-] Time step index

**`Z = 1.`**

[m] Elevation

**`HFLUX = 1.`**

[W/m^2] Film heat flux

**`MEVAP = -1.`**

[kg/s/m^2] Evaporation mass flux

**`W = 1.`**

Flow properties

**`U = 1.`**

[m/s] Velocity

**`H = 1E6`**

[J/kg] Enthalpy

**`ITR`**

Iteration properties

*`Solvers`* is pretty cool.

**class** `Solvers.`**`AbstractField`**

Bases: `matlab.mixin.Copyable`

ABSTRACTFIELD Summary of this class goes here

Detailed explanation goes here

**Property Summary**

**`NZ`**

[-] Number of axial steps

**`NTIME`**

[-] Number of time steps

**TIME**
[s] Time series

**DT**
[s] Time step size

**TIDX**
[-] Time step index

**Z**
[m] Elevation

**ITR**
Iteration properties

**flowProperties = {'W','U','H'}**
Flow properties used for copying

**Method Summary**

**memoizeFunction**(*methodStr*, *methodHandle*, *varargin*)
MEMOIZEDMETHOD Implement a mechanism for registering memoizeable functions
Adapted from https://stackoverflow.com/a/75037451
For the first call with a particular method, create and memoize a function handle view of the method

**struct**()
STRUCT Converter to *struct()*

**copyFlowProperties**(*srcObj*, *targetObj*, *opts*)
COPYFLOWPROPERTIES

class Solvers.**SolverPlotter**(*Titles*, *WallIdxs*)
Bases: `handle`

SOLVERPLOTTER Framework for generating solver plots Detailed explanation goes here

**Constructor Summary**

**SolverPlotter**(*Titles*, *WallIdxs*)
SOLVERPLOTTER Construct an instance of this class Detailed explanation goes here

**Property Summary**

**FontSize = 10**
Font size of text in plots

**Title = ""**
Figure title

**WallIdx = 1**
Wall index

**Grid = 'on'**
*Grid* option

**Method Summary**

**plotz**(*plotters*, *YData*, *fieldName*, *opts*)
PLOTZ Summary of this method goes here Detailed explanation goes here

`gca`(*plotters*)
Loop through plotters

`ylim`(*plotters*, *newLim*)
Loop through plotters

`setZs`(*plotters*, *Zs*)
Loop through plotters

`legend`(*plotters*, *varargin*)
Loop through plotters

`static fieldName2plotStyle`(*fieldName*)
Default colors

`class Solvers.SolverState`
Bases: `uint16`

SOLVERSTATE Enumeration class of possible solver states

`class Solvers.AbstractPhase`
Bases: `handle`

ABSTRACTPHASE Summary of this class goes here Detailed explanation goes here

`class Solvers.AbstractSolver`(*inputSet*)
Bases: `handle`

ABSTRACTSOLVER Summary of this class goes here Detailed explanation goes here

**Constructor Summary**

`AbstractSolver`(*inputSet*)
ABSTRACTSOLVER Constructor

**Property Summary**

`STATE`
Solver state defined by SolverState enum

**Method Summary**

`log`(*solver*, *varargin*)
LOG Log events

`static CreateITR`(*NZ*, *ITRFields*)
Create inner iteration value struct

`class Solvers.AbstractFilm`(*inputSet*, *fluid*)
Bases: `Solvers.AbstractField`

ABSTRACTFILM Summary of this class goes here

Detailed explanation goes here

**Constructor Summary**

**AbstractFilm**(*inputSet*, *fluid*)

ABSFILM Creates an abstract film, absfilm

Detailed explanation goes here

**Property Summary**

**DZ = 0**

[m] Axial step size

**Method Summary**

**WL**(*absfilm*, *zIdx*)

*WL()* Film mass flow rate per unit perimeter

**THICK**(*absfilm*, *zIdx*)

*THICK()* Film thickness

**RE**(*absfilm*, *zIdx*)

*RE()* Film Reynolds number [-]

**MENT**(*absfilm*, *zIdx*)

*MENT()* Film entrainment mass flux

**MTOT**(*absfilm*, *drop*, *zIdx*)

*MTOT()* Total

**CW**(*absfilm*, *zIdx*)

*CW()* Wall friction factor

**FWALL**(*absfilm*, *zIdx*)

*FWALL()* Film wall shear stress

**UWALL**(*absfilm*, *zIdx*)

*UWALL()* [m/s] Film wall velocity

**YPLUS2THICK**(*absfilm*, *yplus*, *zIdx*)

*YPLUS2THICK()* Calculate thickness from wall unit value

**CV**(*absfilm*, *zIdx*)

*CV()* Film/vapor interfacial friction factor

**FVAPOR**(*absfilm*, *zIdx*)

*FVAPOR()* Film vapor shear stress

**FBUOY**(*absfilm*, *zIdx*)

*FBUOY()* Film buoyancy

**FGRAV**(*absfilm*, *zIdx*)

*FGRAV()* Film gravity

**FDEP**(*absfilm*, *drop*, *zIdx*)

*FDEP()* Drop deposition shear

**FTOT**(*absfilm*, *drop*, *zIdx*)

*FTOT()* Total

**UALGEBR**(*absfilm*, *zIdx*)

*UALGEBR()* Film velocity based on simple algebraic model

**UEQUILS**(*absfilm*, *zIdx*)

*UEQUILS()* Film velocity based on simple equilibrium model (Fwall + Fvapor = 0)

**UEQUIL**(*absfilm*, *drop*, *zIdx*)

*UEQUIL()* Film velocity based on complete equilibrium model (Ftot = 0)

CHAPTER

FOUR

# EXTERNALPACKAGES

## 4.1 CoolPropWrapper

# BIBLIOGRAPHY

[LeCorre22] Jean-Marie Le Corre. Phenomenological model of disturbance waves in annular two-phase flow. *International Journal of Multiphase Flow*, 151:104057, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0301932222000696, doi:https://doi.org/10.1016/j.ijmultiphaseflow.2022.104057.

# MATLAB MODULE INDEX

### c

CoolPropWrapper, 37

### i

InputEnums, 7
Inputs, 9

### s

Session, 18
Solvers, 31
Solvers.FourField, 19
Solvers.Mixture, 25
Solvers.ThreeField, 29

# INDEX


## A

AbstractField (*class in Solvers*), 31
AbstractFilm (*class in Solvers*), 33
AbstractFilm() (*Solvers.AbstractFilm method*), 33
AbstractPhase (*class in Solvers*), 33
AbstractSolver (*class in Solvers*), 33
AbstractSolver() (*Solvers.AbstractSolver method*), 33
AFDISTR() (*Solvers.Mixture.Mixture method*), 29
AFFNC() (*Solvers.Mixture.Mixture method*), 29
AI() (*Solvers.ThreeField.Drop method*), 30
AMP() (*Solvers.FourField.Wave method*), 21
ANGLE (*Inputs.Geometry attribute*), 12
ANGLE (*Inputs.Model attribute*), 15
AREA (*Inputs.Geometry attribute*), 12
AREA() (*Solvers.ThreeField.Drop method*), 30
AXIALINTERP (*Inputs.Options attribute*), 13

## B

Base (*class in Solvers.FourField*), 22
Base() (*Solvers.FourField.Base method*), 22
BASEEQTHICK (*class in InputEnums*), 7
BASEEQTHICKCOEF (*Inputs.Model attribute*), 17
BETA() (*Solvers.FourField.Base method*), 23
BETA() (*Solvers.FourField.Wave method*), 20
BETAP() (*Solvers.FourField.Base method*), 23
BETAP() (*Solvers.FourField.Wave method*), 20
BOILCOEF (*class in InputEnums*), 8
BOILCOEF (*Inputs.Model attribute*), 16
BoundaryConditions (*class in Inputs*), 10
BoundaryConditions() (*Inputs.BoundaryConditions method*), 10

## C

closeLog() (*Session.Log method*), 19
CONC() (*Solvers.ThreeField.Drop method*), 30
convert2JSON() (*Inputs.Input static method*), 12
CoolPropWrapper (*module*), 37
copyFlowProperties() (*Solvers.AbstractField method*), 32
copyFlowProperties() (*Solvers.FourField.Base method*), 24
copyFlowProperties() (*Solvers.FourField.Film method*), 25
CreateITR() (*Solvers.AbstractSolver static method*), 33
CV() (*Solvers.AbstractFilm method*), 34
CW() (*Solvers.AbstractFilm method*), 34

## D

defaultValueUsedReport() (*Inputs.Input method*), 12
delete() (*Session.Log method*), 19
DELTAFREQ() (*Solvers.FourField.Wave method*), 21
DENSITY() (*Solvers.ThreeField.Drop method*), 30
DEPOSITION (*class in InputEnums*), 7
DEPOSITION (*Inputs.Model attribute*), 17
DHIN() (*Inputs.BoundaryConditions method*), 11
DIAM() (*Solvers.ThreeField.Drop method*), 30
diaryOff() (*Session.Log method*), 19
diaryOn() (*Session.Log method*), 19
distributeOAFW() (*Solvers.FourField.Film method*), 25
DP (*Solvers.Mixture.Mixture attribute*), 27
DPACCT() (*Solvers.Mixture.Mixture method*), 28
DPACCZ() (*Solvers.Mixture.Mixture method*), 28
DPGRAV() (*Solvers.Mixture.Mixture method*), 28
DPK() (*Solvers.Mixture.Mixture method*), 28
DPPARTS() (*Solvers.Mixture.Mixture method*), 28
DPTOT() (*Solvers.Mixture.Mixture method*), 28
DPWALL() (*Solvers.Mixture.Mixture method*), 28
DRAG() (*Solvers.ThreeField.Drop method*), 30
DRAGCOEF() (*Solvers.FourField.Wave method*), 22
Drop (*class in Solvers.FourField*), 22
Drop (*class in Solvers.ThreeField*), 29
Drop() (*Solvers.ThreeField.Drop method*), 29
DROPDIAM (*Inputs.Model attribute*), 17
DROPDRAG (*class in InputEnums*), 7
DROPDRAG (*Inputs.Model attribute*), 17
DROPDRAGCOEF (*Inputs.Model attribute*), 17
DROPSLIP (*Inputs.Model attribute*), 17
DT (*Solvers.AbstractField attribute*), 32
DT (*Solvers.FourField.Base attribute*), 22
DT (*Solvers.FourField.Film attribute*), 24
DT (*Solvers.FourField.Wave attribute*), 20
DT (*Solvers.Mixture.Mixture attribute*), 27

DT (*Solvers.ThreeField.Drop attribute*), 29
DT (*Solvers.ThreeField.Film attribute*), 31
DTIN() (*Inputs.BoundaryConditions method*), 11
DZ (*Solvers.AbstractFilm attribute*), 34
DZ (*Solvers.Mixture.Mixture attribute*), 27

## E

ENTRAINMENT (*class in InputEnums*), 7
ENTRAINMENT (*Inputs.Model attribute*), 17
EPSILON() (*Solvers.FourField.Base method*), 22
EPSILON() (*Solvers.FourField.Wave method*), 20
EQFREQUENCY() (*Solvers.FourField.Wave method*), 21
EQPERIOD() (*Solvers.FourField.Wave method*), 21
EQSTROUHAL (*class in InputEnums*), 8
EQSTROUHAL() (*Solvers.FourField.Wave method*), 21
EQSTROUHALCOEF (*Inputs.Model attribute*), 17
EQTHICK() (*Solvers.FourField.Base method*), 23
EQUIL() (*Solvers.FourField.Film method*), 25
ERRORH (*Inputs.Options attribute*), 13
ERRORP (*Inputs.Options attribute*), 13
ERRORU (*Inputs.Options attribute*), 13
ERRORUD (*Inputs.Options attribute*), 14
ERRORUF (*Inputs.Options attribute*), 14
ERRORW (*Inputs.Options attribute*), 13
ERRORWF (*Inputs.Options attribute*), 14
ETA() (*Solvers.FourField.Base method*), 23
ETA() (*Solvers.FourField.Wave method*), 20

## F

FBASE() (*Solvers.FourField.Wave method*), 21
FBASEMASS() (*Solvers.FourField.Wave method*), 21
FBASEVAPOR() (*Solvers.FourField.Base method*), 23
FBUOY() (*Solvers.AbstractFilm method*), 34
FBUOY() (*Solvers.ThreeField.Drop method*), 30
FDEP() (*Solvers.AbstractFilm method*), 34
FDEP() (*Solvers.FourField.Base method*), 23
FDRAG() (*Solvers.FourField.Wave method*), 21
FDRAG() (*Solvers.ThreeField.Drop method*), 30
FDRY() (*Solvers.FourField.Base method*), 23
FENT() (*Solvers.ThreeField.Drop method*), 30
FGRAV() (*Solvers.AbstractFilm method*), 34
FGRAV() (*Solvers.ThreeField.Drop method*), 30
fieldName2plotStyle() (*Solvers.SolverPlotter static method*), 33
Film (*class in Solvers.FourField*), 24
Film (*class in Solvers.ThreeField*), 30
Film() (*Solvers.FourField.Film method*), 24
flowProperties (*Solvers.AbstractField attribute*), 32
FLUID (*Inputs.FluidProperties attribute*), 9
FLUID (*Inputs.Model attribute*), 15
FLUIDPROPERTIES (*class in InputEnums*), 8
FluidProperties (*class in Inputs*), 9
FluidProperties() (*Inputs.FluidProperties method*), 9
FontSize (*Solvers.SolverPlotter attribute*), 32
FREQUENCY (*Solvers.FourField.Wave attribute*), 20
FRICTION (*Inputs.Model attribute*), 15
FSHEAR() (*Solvers.FourField.Wave method*), 21
FTOT() (*Solvers.AbstractFilm method*), 34
FTOT() (*Solvers.FourField.Base method*), 23
FTOT() (*Solvers.FourField.Wave method*), 21
FTOT() (*Solvers.ThreeField.Drop method*), 30
FVAPOR() (*Solvers.AbstractFilm method*), 34
FVAPOR() (*Solvers.FourField.Wave method*), 21
FW() (*Solvers.Mixture.Mixture method*), 28
FWALL() (*Solvers.AbstractFilm method*), 34
FWALL() (*Solvers.FourField.Wave method*), 21
FWAVE() (*Solvers.FourField.Base method*), 23
FWAVEMASS() (*Solvers.FourField.Base method*), 23

## G

G (*Inputs.Model attribute*), 18
gca() (*Solvers.SolverPlotter method*), 32
Geometry (*class in Inputs*), 11
Geometry() (*Inputs.Geometry method*), 11
Grid (*Solvers.SolverPlotter attribute*), 32

## H

H (*Solvers.FourField.Base attribute*), 22
H (*Solvers.FourField.Film attribute*), 24
H (*Solvers.FourField.Wave attribute*), 20
H (*Solvers.Mixture.Mixture attribute*), 27
H (*Solvers.ThreeField.Drop attribute*), 29
H (*Solvers.ThreeField.Film attribute*), 31
H() (*Inputs.FluidProperties method*), 10
H() (*Solvers.Mixture.Liquid method*), 25
H() (*Solvers.Mixture.Vapor method*), 26
HDIAM() (*Inputs.Geometry method*), 12
HF (*Inputs.FluidProperties attribute*), 9
HF() (*Inputs.BoundaryConditions method*), 11
HFG (*Inputs.FluidProperties attribute*), 9
HFLUX (*Solvers.FourField.Base attribute*), 22
HFLUX (*Solvers.FourField.Film attribute*), 24
HFLUX (*Solvers.FourField.Wave attribute*), 20
HFLUX (*Solvers.Mixture.Mixture attribute*), 27
HFLUX (*Solvers.ThreeField.Film attribute*), 31
HG (*Inputs.FluidProperties attribute*), 9
HG() (*Inputs.BoundaryConditions method*), 11
HIN (*Inputs.BoundaryConditions attribute*), 11

## I

ID (*Inputs.Geometry attribute*), 12
ID (*Inputs.Model attribute*), 15
ID (*Inputs.Options attribute*), 13
initializeBaseAndWave() (*Solvers.FourField.Film method*), 25
Input (*class in Inputs*), 12
Input() (*Inputs.Input method*), 12
InputEnums (*module*), 7

Inputs (*module*), 9
InputSet (*class in Inputs*), 9
InputSet() (*Inputs.InputSet method*), 9
INTAREA (*class in InputEnums*), 8
INTAREA (*Inputs.Model attribute*), 16
INTLENGTH (*class in InputEnums*), 7
INTLENGTH (*Inputs.Model attribute*), 16
INTLENGTHCST (*Inputs.Model attribute*), 16
INTNU (*class in InputEnums*), 7
INTNU (*Inputs.Model attribute*), 16
INTNULCST (*Inputs.Model attribute*), 16
INTNUVCST (*Inputs.Model attribute*), 16
INTTRANSH (*class in InputEnums*), 8
INTTRANSH (*Inputs.Model attribute*), 16
isLegalPath() (*in module Session*), 18
ITR (*Solvers.AbstractField attribute*), 32
ITR (*Solvers.FourField.Base attribute*), 22
ITR (*Solvers.FourField.Film attribute*), 24
ITR (*Solvers.FourField.Wave attribute*), 20
ITR (*Solvers.Mixture.Mixture attribute*), 27
ITR (*Solvers.ThreeField.Drop attribute*), 29
ITR (*Solvers.ThreeField.Film attribute*), 31

## J

JG() (*Solvers.Mixture.Mixture method*), 28
JL() (*Solvers.Mixture.Mixture method*), 28

## K

KF (*Inputs.FluidProperties attribute*), 10
KG (*Inputs.FluidProperties attribute*), 10
KL() (*Inputs.FluidProperties method*), 10
KLOC (*Inputs.Model attribute*), 15
KLOSS (*Inputs.Model attribute*), 15
KLOSS() (*Solvers.Mixture.Mixture method*), 28
KV() (*Inputs.FluidProperties method*), 10

## L

legend() (*Solvers.SolverPlotter method*), 33
LENGTH (*Inputs.Geometry attribute*), 12
Liquid (*class in Solvers.Mixture*), 25
liquid (*Solvers.Mixture.Mixture attribute*), 27
Liquid() (*Solvers.Mixture.Liquid method*), 25
listInputProperties() (*Inputs.Input method*), 12
LOCRELVEL (*class in InputEnums*), 8
LOCRELVEL (*Inputs.Model attribute*), 16
Log (*class in Session*), 18
Log() (*Session.Log method*), 18
log() (*Session.Log method*), 19
log() (*Solvers.AbstractSolver method*), 33
LOGFID (*Session.Log attribute*), 19
LogMode (*class in Session*), 19

## M

makeSessionDirectory() (*Session.Session method*), 18
MAXITER (*Inputs.Options attribute*), 13
MBASE() (*Solvers.FourField.Wave method*), 21
MDEP() (*Solvers.ThreeField.Drop method*), 30
memoizeFunction() (*Solvers.AbstractField method*), 32
MENT() (*Solvers.AbstractFilm method*), 34
MENT() (*Solvers.FourField.Base method*), 23
MENT() (*Solvers.FourField.Wave method*), 20
MEVAP (*Solvers.FourField.Film attribute*), 24
MEVAP (*Solvers.ThreeField.Film attribute*), 31
MEVAP() (*Solvers.FourField.Base method*), 23
MEVAP() (*Solvers.FourField.Wave method*), 20
MFLOW (*Inputs.BoundaryConditions attribute*), 11
MFLUX() (*Solvers.Mixture.Liquid method*), 25
MFLUX() (*Solvers.Mixture.Mixture method*), 27
MFLUX() (*Solvers.Mixture.Vapor method*), 26
mix (*Solvers.ThreeField.Drop attribute*), 30
Mixture (*class in Solvers.Mixture*), 26
Mixture() (*Solvers.Mixture.Mixture method*), 26
Model (*class in Inputs*), 15
Model() (*Inputs.Model method*), 15
MOMENTBASE (*class in InputEnums*), 7
MOMENTBASE (*Inputs.Model attribute*), 17
MOMENTDROP (*class in InputEnums*), 7
MOMENTDROP (*Inputs.Model attribute*), 17
MOMENTFILM (*class in InputEnums*), 8
MOMENTFILM (*Inputs.Model attribute*), 17
MOMENTGAS (*class in InputEnums*), 8
MOMENTGAS (*Inputs.Model attribute*), 15
MOMENTLIQUID (*class in InputEnums*), 8
MOMENTLIQUID (*Inputs.Model attribute*), 15
MOMENTWAVE (*class in InputEnums*), 7
MOMENTWAVE (*Inputs.Model attribute*), 18
MTOT() (*Solvers.AbstractFilm method*), 34
MTOT() (*Solvers.FourField.Base method*), 23
MTOT() (*Solvers.FourField.Wave method*), 21
MTURB() (*Solvers.FourField.Base method*), 23
MTURB() (*Solvers.FourField.Wave method*), 21
MU() (*Solvers.Mixture.Mixture method*), 28
MUF (*Inputs.FluidProperties attribute*), 9
MUG (*Inputs.FluidProperties attribute*), 9
MUL() (*Inputs.FluidProperties method*), 10
MUV() (*Inputs.FluidProperties method*), 10
MWAVE() (*Solvers.FourField.Base method*), 23

## N

N() (*Solvers.FourField.Wave method*), 21
NNODES (*Inputs.Model attribute*), 15
NTIME (*Solvers.AbstractField attribute*), 31
NTIME (*Solvers.FourField.Base attribute*), 22
NTIME (*Solvers.FourField.Film attribute*), 24
NTIME (*Solvers.FourField.Wave attribute*), 20

NTIME (*Solvers.Mixture.Mixture attribute*), 27
NTIME (*Solvers.ThreeField.Drop attribute*), 29
NTIME (*Solvers.ThreeField.Film attribute*), 31
NWALL() (*Inputs.Geometry method*), 12
NZ (*Solvers.AbstractField attribute*), 31
NZ (*Solvers.FourField.Base attribute*), 22
NZ (*Solvers.FourField.Film attribute*), 24
NZ (*Solvers.FourField.Wave attribute*), 19
NZ (*Solvers.Mixture.Mixture attribute*), 26
NZ (*Solvers.ThreeField.Drop attribute*), 29
NZ (*Solvers.ThreeField.Film attribute*), 31

## O

OAF (*class in InputEnums*), 8
OAF (*Inputs.Model attribute*), 16
OAFBASERATIO (*Inputs.Model attribute*), 16
OAFDROPRATIO (*Inputs.Model attribute*), 16
OAFENTRAINED (*class in InputEnums*), 8
OAFENTRAINED (*Inputs.Model attribute*), 16
OAFFILMSPLIT (*class in InputEnums*), 7
OAFFILMSPLIT (*Inputs.Model attribute*), 16
OAFIDX() (*Solvers.Mixture.Mixture method*), 28
OAFTRANSITION (*Inputs.Model attribute*), 17
OAFWL() (*Solvers.Mixture.Mixture method*), 29
OAFZ() (*Solvers.Mixture.Mixture method*), 29
openLog() (*Session.Log method*), 19
Options (*class in Inputs*), 12
Options() (*Inputs.Options method*), 12

## P

P (*Solvers.Mixture.Mixture attribute*), 27
PERIM (*Inputs.Geometry attribute*), 12
PERIOD() (*Solvers.FourField.Wave method*), 21
plot() (*Inputs.BoundaryConditions method*), 11
plot() (*Inputs.FluidProperties method*), 10
plotz() (*Solvers.SolverPlotter method*), 32
POSFILM (*Inputs.Model attribute*), 17
POWER (*Inputs.BoundaryConditions attribute*), 11
PRANDTLF (*Inputs.FluidProperties attribute*), 10
PRANDTLG (*Inputs.FluidProperties attribute*), 10
PRANDTLL() (*Inputs.FluidProperties method*), 10
PRANDTLV() (*Inputs.FluidProperties method*), 10
PRESSURE (*Inputs.BoundaryConditions attribute*), 11
PRESSURE (*Inputs.FluidProperties attribute*), 9
PROPERTIES (*Inputs.FluidProperties attribute*), 9
PROPERTIES (*Inputs.Model attribute*), 15

## R

RANZMARSHALLLCST (*Inputs.Model attribute*), 16
RANZMARSHALLVCST (*Inputs.Model attribute*), 16
RE() (*Solvers.AbstractFilm method*), 34
RE() (*Solvers.Mixture.Liquid method*), 26
RE() (*Solvers.Mixture.Mixture method*), 28
RE() (*Solvers.Mixture.Vapor method*), 26
RE() (*Solvers.ThreeField.Drop method*), 30
REL() (*Solvers.Mixture.Mixture method*), 28
RELAXFW (*Inputs.Options attribute*), 14
RELAXHL (*Inputs.Options attribute*), 14
RELAXHM (*Inputs.Options attribute*), 13
RELAXHV (*Inputs.Options attribute*), 14
RELAXPM (*Inputs.Options attribute*), 13
RELAXTB (*Inputs.Model attribute*), 17
RELAXTCOND (*Inputs.Model attribute*), 16
RELAXTEVAP (*Inputs.Model attribute*), 16
RELAXTW (*Inputs.Model attribute*), 18
RELAXUB (*Inputs.Options attribute*), 14
RELAXUD (*Inputs.Options attribute*), 14
RELAXUF (*Inputs.Options attribute*), 14
RELAXUL (*Inputs.Options attribute*), 14
RELAXUV (*Inputs.Options attribute*), 14
RELAXUW (*Inputs.Options attribute*), 14
RELAXWB (*Inputs.Options attribute*), 14
RELAXWF (*Inputs.Options attribute*), 14
RELAXWL (*Inputs.Options attribute*), 13
RELAXWM (*Inputs.Options attribute*), 13
RELAXWV (*Inputs.Options attribute*), 14
RELAXWW (*Inputs.Options attribute*), 14
RELVELCST (*Inputs.Model attribute*), 16
RHO() (*Solvers.Mixture.Mixture method*), 28
RHOF (*Inputs.FluidProperties attribute*), 9
RHOG (*Inputs.FluidProperties attribute*), 9
RHOL() (*Inputs.FluidProperties method*), 10
RHOV() (*Inputs.FluidProperties method*), 10

## S

SCBOIL (*class in InputEnums*), 8
SCBOIL (*Inputs.Model attribute*), 15
Session (*class in Session*), 18
Session (*module*), 18
Session() (*Session.Session method*), 18
setupLog() (*Session.Session method*), 18
setZs() (*Solvers.SolverPlotter method*), 33
SHAPEFACTOR() (*Solvers.FourField.Wave method*), 21
SHAPEFACTORCOEF (*Inputs.Model attribute*), 17
showWarnings (*Session.Log attribute*), 19
showWarnings (*Session.Session attribute*), 18
SIGMA (*Inputs.FluidProperties attribute*), 9
SLIP (*Inputs.Model attribute*), 15
SolverPlotter (*class in Solvers*), 32
SolverPlotter() (*Solvers.SolverPlotter method*), 32
Solvers (*module*), 31
Solvers.FourField (*module*), 19
Solvers.Mixture (*module*), 25
Solvers.ThreeField (*module*), 29
SolverState (*class in Solvers*), 33
SPACING() (*Solvers.FourField.Wave method*), 21
SSCONVH (*Inputs.Options attribute*), 13
SSCONVP (*Inputs.Options attribute*), 13

SSCONVU (*Inputs.Options attribute*), 13
SSCONVUD (*Inputs.Options attribute*), 14
SSCONVUF (*Inputs.Options attribute*), 14
SSCONVW (*Inputs.Options attribute*), 13
SSCONVWF (*Inputs.Options attribute*), 14
SSMAXITER (*Inputs.Options attribute*), 13
SSTSTEP (*Inputs.Options attribute*), 13
STATE (*Solvers.AbstractSolver attribute*), 33
struct() (*Solvers.AbstractField method*), 32

## T

T() (*Inputs.FluidProperties method*), 10
T() (*Solvers.Mixture.Mixture method*), 28
TAUW() (*Solvers.Mixture.Mixture method*), 28
TBASE() (*Solvers.FourField.Base method*), 23
TDRY() (*Solvers.FourField.Base method*), 23
THICK() (*Solvers.AbstractFilm method*), 34
THICK() (*Solvers.FourField.Wave method*), 20
THICKMIN() (*Solvers.FourField.Base method*), 24
THINFILMFRIC (*class in InputEnums*), 8
THINFILMFRIC (*Inputs.Model attribute*), 17
THINFILMTHICK (*Inputs.Model attribute*), 17
TIDX (*Solvers.AbstractField attribute*), 32
TIDX (*Solvers.FourField.Base attribute*), 22
TIDX (*Solvers.FourField.Film attribute*), 24
TIDX (*Solvers.FourField.Wave attribute*), 20
TIDX (*Solvers.Mixture.Mixture attribute*), 27
TIDX (*Solvers.ThreeField.Drop attribute*), 29
TIDX (*Solvers.ThreeField.Film attribute*), 31
TIME (*Inputs.BoundaryConditions attribute*), 11
TIME (*Solvers.AbstractField attribute*), 31
TIME (*Solvers.FourField.Base attribute*), 22
TIME (*Solvers.FourField.Film attribute*), 24
TIME (*Solvers.FourField.Wave attribute*), 20
TIME (*Solvers.Mixture.Mixture attribute*), 27
TIME (*Solvers.ThreeField.Drop attribute*), 29
TIME (*Solvers.ThreeField.Film attribute*), 31
TIME() (*Solvers.Mixture.Liquid method*), 25
TIME() (*Solvers.Mixture.Vapor method*), 26
TIMEINTERP (*Inputs.Options attribute*), 13
TIN() (*Inputs.BoundaryConditions method*), 11
Title (*Solvers.SolverPlotter attribute*), 32
TPFM (*class in InputEnums*), 8
TPFM (*Inputs.Model attribute*), 15
TPKM (*class in InputEnums*), 8
TPKM (*Inputs.Model attribute*), 15
TSAT (*Inputs.FluidProperties attribute*), 9
TSAT() (*Inputs.BoundaryConditions method*), 11
TSTEP (*Inputs.Options attribute*), 13

## U

U (*Solvers.FourField.Base attribute*), 22
U (*Solvers.FourField.Film attribute*), 24
U (*Solvers.FourField.Wave attribute*), 20
U (*Solvers.ThreeField.Drop attribute*), 29
U (*Solvers.ThreeField.Film attribute*), 31
U() (*Solvers.Mixture.Liquid method*), 25
U() (*Solvers.Mixture.Mixture method*), 28
U() (*Solvers.Mixture.Vapor method*), 26
UALGEBR() (*Solvers.AbstractFilm method*), 34
UALGEBR() (*Solvers.ThreeField.Drop method*), 30
UEQUIL() (*Solvers.AbstractFilm method*), 35
UEQUIL() (*Solvers.ThreeField.Drop method*), 30
UEQUILS() (*Solvers.AbstractFilm method*), 34
USLIP() (*Solvers.ThreeField.Drop method*), 30
UWALL() (*Solvers.AbstractFilm method*), 34

## V

validateInputEntry() (*Inputs.Input method*), 12
Vapor (*class in Solvers.Mixture*), 26
Vapor() (*Solvers.Mixture.Vapor method*), 26
VAPORFRIC (*class in InputEnums*), 7
VAPORFRIC (*Inputs.Model attribute*), 17
VAPORFRICCST (*Inputs.Model attribute*), 17
VF() (*Solvers.Mixture.Liquid method*), 25
VF() (*Solvers.Mixture.Mixture method*), 27
VF() (*Solvers.Mixture.Vapor method*), 26
VOID (*class in InputEnums*), 7
VOID (*Inputs.Model attribute*), 15
VOLUME() (*Solvers.ThreeField.Drop method*), 30

## W

W (*Solvers.FourField.Base attribute*), 22
W (*Solvers.FourField.Film attribute*), 24
W (*Solvers.FourField.Wave attribute*), 20
W (*Solvers.Mixture.Mixture attribute*), 27
W (*Solvers.ThreeField.Drop attribute*), 29
W (*Solvers.ThreeField.Film attribute*), 31
W() (*Solvers.Mixture.Liquid method*), 25
W() (*Solvers.Mixture.Vapor method*), 26
WallIdx (*Solvers.SolverPlotter attribute*), 32
warning() (*Session.Log method*), 19
Wave (*class in Solvers.FourField*), 19
Wave() (*Solvers.FourField.Wave method*), 19
WAVEDRAGCOEF (*Inputs.Model attribute*), 18
WAVEFREQUENCY (*class in InputEnums*), 8
WAVEFREQUENCY (*Inputs.Model attribute*), 18
WAVEMIXCOEF (*Inputs.Model attribute*), 18
WIDTH() (*Solvers.FourField.Wave method*), 21
WL() (*Solvers.AbstractFilm method*), 34
WL() (*Solvers.FourField.Wave method*), 20
WMESH (*Inputs.BoundaryConditions attribute*), 11
WMIN() (*Solvers.FourField.Base method*), 23
WMINL() (*Solvers.FourField.Base method*), 23
WPOWER (*Inputs.BoundaryConditions attribute*), 11

## X

X() (*Solvers.Mixture.Liquid method*), 25

X() (*Solvers.Mixture.Mixture method*), 27
X() (*Solvers.Mixture.Vapor method*), 26
XEQ() (*Solvers.Mixture.Mixture method*), 27
XIN() (*Inputs.BoundaryConditions method*), 11

# Y

ylim() (*Solvers.SolverPlotter method*), 33
YPLUS2THICK() (*Solvers.AbstractFilm method*), 34

# Z

Z (*Solvers.AbstractField attribute*), 32
Z (*Solvers.FourField.Base attribute*), 22
Z (*Solvers.FourField.Film attribute*), 24
Z (*Solvers.FourField.Wave attribute*), 20
Z (*Solvers.Mixture.Mixture attribute*), 27
Z (*Solvers.ThreeField.Drop attribute*), 29
Z (*Solvers.ThreeField.Film attribute*), 31
Z() (*Solvers.Mixture.Liquid method*), 25
Z() (*Solvers.Mixture.Vapor method*), 26