Definitions:

- *Compound*:
  A chemical substance defined by particular set of physical properties, calculation methods and data. Examples: water, hydrogen, oxygen.

- *Phase*:
  Stable collection of compounds with a defined amount of substance and a homogeneous composition. It has an associated state of aggregation, e.g. liquid.

- *State of aggregation*:
  The physical state in which the compounds in that phase occur. Possible state of aggregation: vapor, liquid, solid.

- *Class*:
  A software component which consists of member variables (data fields) and associated functions (methods).

- *Objects:*
  Independent instances of specified class.

Interfaces of material stream and holdup:

Functions to work with basic stream properties
GetStreamName SetStreamName
Functions to work with time points
AddTimePoint RemoveTimePoint RemoveTimePoints RemoveTimePointsAfter GetAllTimePoints GetTimePointsForInterval GetLastTimePoint GetPreviousTimePoint
Functions to work with overall properties
GetMassFlow / GetMass SetMassFlow / SetMass GetTemperature SetTemperature GetPressure SetPressure GetOverallProperty SetOverallProperty CalcTemperatureFromProperty CalcPressureFromProperty
Functions to work with compounds
GetCompoundFraction GetCompoundPhaseFraction SetCompoundPhaseFraction GetCompoundMassFlow / GetCompoundMass GetCompoundConstant GetCompoundTPDProp GetCompoundInteractionProp
Functions to work with phases
GetPhaseMassFlow / GetPhaseMass SetPhaseMassFlow / SetPhaseMass GetSinglePhaseProp SetSinglePhaseProp GetPhaseTPDProp
Functions to work with solid distributed properties
GetFraction SetFraction GetDistribution SetDistribution ApplyTM NormalizeDistribution
Functions to work with particle size distributions
GetPSD SetPSD
Functions to work with lookup tables
GetLookupTable CalcTemperatureFromProperty CalcPressureFromProperty CalcPropertyFromTemperature

CalcPropertyFromPressure
Functions to work with other streams
CopyFromStream CopyFromHoldup AddStream CopyFromHoldup AddHoldup AddStream
Tables of properties

*Function – the same functions for CMaterialStream and CHoldup
*Function – specific functions for CMaterialStream
*Function – specific functions for CHoldup

Functions to work with basic stream properties

std::string GetStreamName *()*
Returns the name of the material stream/holdup.

void SetStreamName *(std::string Name)*
Sets the name of the material stream/holdup.

Functions to work with time points

void AddTimePoint *(double Time, double SourceTime = -1)*
Adds new time point *Time* to the material stream/holdup. Data for this time point is copied from *SourceTime*. By default (*SourceTime* = -1) data will be copied from the previous time point. If this is the first time point in the material stream/holdup, all data will be set to 0. If such time point already exists, nothing will be done.

void RemoveTimePoint *(double Time)*
Removes time point *Time* from the material stream/holdup, if such point exists.

void RemoveTimePoints *(double Start, double End)*
Removes all time points from the specified interval, including boundaries.

void RemoveTimePointsAfter *(double Start, bool IncludeStart = false)*
Removes all data after the specified time point including (if *IncludeStart* is set to *true*) or excluding (*IncludeStart* is set to *false*) point *Start*.

std::vector<double> GetAllTimePoints *()*
Returns all time points which are defined in the material stream/holdup.

std::vector<double> GetTimePointsForInterval *(double Start, double End, bool ForceInclBoudaries = false)*
Returns the list of time points for the specified time interval (incl. boundary points *Start* and *End*). If *ForceInclBoudaries* is set to true, resulting vector will contain boundary points even if they have not been defined in the material stream/holdup.

double GetLastTimePoint *()*
Returns last defined time point in the material stream/holdup. Returns -1 if no time points have been defined.

double GetPreviousTimePoint *(double Time)*
Returns the nearest time point before *Time*. Returns -1 if there is no time points before the specified value.

Functions to work with overall properties

double GetMassFlow / GetMass *(double Time, unsigned Basis = BASIS_MASS)*
Returns mass flow/mass of the material stream/holdup at the specified time point *Time*. If such time point has not been defined, interpolation of data will be done. *Basis* is a basis of results (*BASIS_MASS* [*kg/s* or *kg*] or *BASIS_MOLL* [*mol/s* or *mol*]).

BASIS_MASS	BASIS_MOLL
$\boldsymbol{m}$ [kg/s] or [kg]	$\sum_i \frac{m \cdot w_i}{M_i}$ [mol/s] or [mol]

m – total mass flow/mass of the material stream/holdup
w_i – mass fraction of the phase *i*
M_i – molar mass of the phase *i*

void SetMassFlow / SetMass *(double Time, double Value, unsigned Basis = BASIS_MASS)*
Sets mass flow/mass of the material stream/holdup at the time point *Time*. Negative values before setting will be converted to *0*. If the time point *Time* has not been defined in the material stream/holdup, then the value will not be set. *Basis* is a basis of results (*BASIS_MASS* [*kg/s* or *kg*] or *BASIS_MOLL* [*mol/s* or *mol*]).

BASIS_MASS	BASIS_MOLL
$\boldsymbol{m = Value}$ $\boldsymbol{Value}$ in [kg/s] or [kg]	$m = Value \cdot \sum_i M_i \cdot w_i$ $Value$ in [mol/s] or [mol]

m – total mass flow/mass of the material stream/holdup
w_i – mass fraction of the phase *i*
M_i – molar mass of the phase *i*

double GetTemperature *(double Time)*
Returns temperature of the material stream/holdup at the specified time point *Time* in [K]. If such time point has not been defined, interpolation of data will be done.

void SetTemperature *(double Time, double Value)*
Sets temperature of the material stream/holdup at the time point *Time* in [K]. Negative values before setting will be converted to *0*. If time the point *Time* has not been defined in the material stream/holdup, the value will not be set.

double GetPressure*(double Time)*
Returns pressure of the material stream/holdup at the specified time point *Time* in [Pa]. If such time point has not been defined, interpolation of data will be done.

void SetPressure *(double Time, double Value)*
Sets pressure of the material stream/holdup in the time point *Time* in [Pa]. Negative values before setting will be converted to *0*. If the time point *Time* has not been defined in the material stream/holdup, the value will not be set.

double GetOverallProperty *(double Time, unsigned Property, unsigned Basis = BASIS_MASS)*
Returns non-constant physical property value for the overall mixture at the specified time point *Time*. *Property* is an identifier of a physical property. If such time point has not been defined, interpolation of data will be done. *Basis* is a basis of results (*BASIS_MASS* or *BASIS_MOLL*).

Possible properties (Table 1):

- *FLOW*, *TOTAL_FLOW* / *MASS*, *TOTAL_MASS*
- *TEMPERATURE*
- *PRESSURE*
- *MOLAR_MASS*
- *ENTHALPY*

FLOW, TOTAL_FLOW / MASS, TOTAL_MASS

Refer to GetMassFlow / GetMass functions.

TEMPERATURE

Refer to GetTemperature function.

PRESSURE

Refer to GetPressure function.

MOLAR_MASS

$$M = \sum_i M_i w_i$$

M – molar mass of the total flow
M_i – molar mass of the phase i
w_i – mass fraction of the phase i

ENTHALPY

BASIS_MASS	BASIS_MOLL
$\boldsymbol{H = \sum_i H_i w_i}$ [J/kg/s] or [J/kg]	$H = \sum_i H_i x_i$ [J/mol/s] or [J/mol]

H – enthalpy of the material stream/holdup
H_i – enthalpy of the phase *i*
w_i – mass fraction of the phase *i*
x_i – mole fraction of the phase *i*

double SetOverallProperty *(double Time, unsigned Property, double Value, unsigned Basis = BASIS_MASS)*

Sets non-constant physical property value for the overall mixture at the specified time point *Time*. *Property* is an identifier of a physical property. *Basis* is a basis of the value (*BASIS_MASS* or *BASIS_MOLL*).

Possible properties (Table 1):

- *FLOW*, *TOTAL_FLOW* / *MASS*, *TOTAL_MASS*
- *TEMPERATURE*
- *PRESSURE*

FLOW, TOTAL_FLOW / MASS, TOTAL_MASS

Refer to SetMassFlow / SetMass functions.

TEMPERATURE

Refer to SetTemperature function.

PRESSURE

Refer to SetPressure function.

double CalcTemperatureFromProperty *(ECompoundTPProperties Property, double Time, double Value)*

Returns temperature of the material stream/holdup for a specific value *Value* of the property *Property* at the time point *Time*. Possible properties are those defined in material database. For more information, refer to “Thermodynamics.pdf”

double CalcPressureFromProperty *(ECompoundTPProperties Property, double Time, double Value)*

Returns pressure of the material stream/holdup for a specific value *Value* of the property *Property* at the time point *Time*. Possible properties are those defined in material database. Possible properties are those defined in material database. For more information, please refer to “Thermodynamics.pdf”

Functions to work with compounds

double GetCompoundFraction *(double Time , std::string CompoundKey, unsigned Basis = BASIS_MASS)*

Returns total fraction of the compound with key *CompoundKey* at the time point *Time*. If such time point has not been defined, interpolation of data will be done.

BASIS_MASS	BASIS_MOLL
$f_c = \sum_i w_i \cdot f_{i,c}$	$f_c^{mol} = \sum_i w_i \frac{f_{i,c}}{M_c \cdot \sum_j \frac{f_{i,j}}{M_j}}$

f_i – mass fraction of compound *i*

f_i^{mol} – mol fraction of compound *i*

w_i – mass fraction of phase *i*

$f_{i,j}$ – mass fraction of compound *j* in phase *i*

M_j – molar mass of compound *j*

double GetCompoundPhaseFraction *(double Time, std::string CompoundKey, unsigned Phase, unsigned Basis = BASIS_MASS)*

Returns fraction of the compound with the key *CompoundKey* in the phase *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the time point *Time*. If such time point has not been defined, interpolation of data will be done.

BASIS_MASS	BASIS_MOLL
$f_{i,j}$	$f_{i,c}^{mol} = \frac{f_{i,j}}{M_c \cdot \sum_j \frac{f_{i,j}}{M_j}}$

$f_{i,j}$ – mass fraction of compound *j* in phase *i*

$f_{i,j}^{mol}$ – molar fraction of compound *j* in phase *i*

M_j – molar mass of compound *j*

void SetCompoundPhaseFraction *(double Time, std::string CompoundKey, unsigned Phase, double Fraction, unsigned Basis = BASIS_MASS)*

Sets fraction of the compound with key *CompoundKey* in phase *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the time point *Time*. If such time point has not been defined, nothing will be done. Negative values before setting will be converted to zero.

BASIS_MASS	BASIS_MOLL
$f_{i,j} = Fraction$	$f_{i,c} = \frac{Fraction \cdot M_c}{\sum_j \frac{f_{i,j}}{M_j}}$

$f_{i,j}$ – mass fraction of compound *j* in phase *i*

M_j – molar mass of compound *j*

double GetCompoundMassFlow / GetCompoundMass *(double Time, std::string CompoundKey, unsigned Phase, unsigned Basis = BASIS_MASS)*

Returns mass flow/mass of the compound with key *CompoundKey* in phase *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the time point *Time*. If such time point has not been defined,

interpolation of data will be done. *Basis* is a basis of value (*BASIS_MASS* [*kg/s* or *kg*] or *BASIS_MOLL* [*mol/s* or *mol*])

BASIS_MASS	BASIS_MOLL
$\boldsymbol{m_{i,j} = w_i \cdot f_{i,j} \cdot m}$ [kg/s] or [kg]	$m_{i,j} = w_i \cdot f_{i,j} \cdot \sum_k \frac{m \cdot w_k}{M_k}$ [mol/s] or [mol]

$m_{i,j}$ – mass flow/mass of compound *j* in phase *i*

w_i – mass fraction of phase *i*

$f_{i,j}$ – mass fraction of compound *j* in phase *i*

m – total mass flow/mass of the material stream/holdup

M_k – molar mass of phase *k*

double GetCompoundConstant *(std::string CompoundKey, ECompoundConstProperties ConstProperty)*

Returns value of the constant physical property *ConstProperty* for the specified compound. These properties are stored in the database of materials. Possible constants (Table 3):

- *CRITICAL_PRESSURE*
- *CRITICAL_TEMPERATURE*
- *HEAT_OF_FUSION_AT_NORMAL_FREEZING_POINT*
- *HEAT_OF_VAPORIZATION_AT_NORMAL_BOILING_POINT*
- *MOLAR_MASS*
- *NORMAL_BOILING_POINT*
- *NORMAL_FREEZING_POINT*
- *REACTIVITY_TYPE*
- *SOA_AT_NORMAL_CONDITIONS*
- *STANDARD_FORMATION_ENTHALPY*
- *CONST_PROP_USER_DEFINED_XX*

double GetCompoundTPDProp *(std::string CompoundKey, unsigned Property, double Temperature, double Pressure)*

Returns value of the temperature/pressure-dependent physical *Property* (which are stored in the database of materials) for the compound with the specified *Temperature* [K] and *Pressure* [Pa]. Possible properties (Table 4):

- *HEAT_CAPACITY_CP*
- *ENTHALPY*
- *THERMAL_CONDUCTIVITY*
- *VAPOR_PRESSURE*
- *VISCOSITY*
- *DENSITY*
- *PERMITTIVITY*
- *TP_PROP_USER_DEFINED_XX*

double GetCompoundTPDProp *(double Time, std::string CompoundKey, unsigned Property)*

Returns value of the temperature/pressure-dependent physical *Property* (which are stored in the database of materials) for the compound with the current temperature and pressure. Possible properties (Table 4):

- *HEAT_CAPACITY_CP*
- *ENTHALPY*
- *THERMAL_CONDUCTIVITY*
- *VAPOR_PRESSURE*
- *VISCOSITY*
- *DENSITY*
- *PERMITTIVITY*

- *TP_PROP_USER_DEFINED_XX*

double GetCompoundInteractionProp *(std::string CompoundKey1, std::string CompoundKey2, unsigned Property, double Temperature, double Pressure)*
Returns the value of the interaction property *Property* for the selected compounds under the specified *Temperature* [K] and *Pressure* [Pa]. These properties are stored in the database of materials. Possible properties (Table 5):

- *INTERFACE_TENSION*
- *INT_PROP_USER_DEFINED_XX*

double GetCompoundInteractionProp *(double Time, std::string CompoundKey1, std::string CompoundKey2, unsigned Property)*
Returns the value of the interaction property *Property* for the selected compounds under the current temperature and pressure. These properties are stored in the database of materials. Possible properties (Table 5):

- *INTERFACE_TENSION*
- *INT_PROP_USER_DEFINED_XX*

Functions to work with phases

double GetPhaseMassFlow / GetPhaseMass *(double Time, unsigned Phase, unsigned Basis = BASIS_MASS)*
Returns mass flow/mass of the specified phase *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) in the material stream/holdup for the time point *Time*. If such time point has not been defined, the value will be interpolated. *Basis* is a basis of value (*BASIS_MASS* [*kg/s* or *kg*] or *BASIS_MOLL* [*mol/s* or *mol*])

BASIS_MASS	BASIS_MOLL
$\boldsymbol{m_i = m \cdot w_i}$ [kg/s]	$m_i = \frac{m \cdot w_i}{M_i}$ [mol/s]

m_i – mass flow/mass of phase *i*
w_i – mass fraction of phase *i*
m – total mass flow/mass of the material stream/holdup
M_i – molar mass of phase *i*

void SetPhaseMassFlow / SetPhaseMass *(double Time, unsigned Phase, double Value, unsigned Basis = BASIS_MASS)*
Sets mass flow/mass of the specified phase *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) in the material stream/holdup for the time point *Time*. Is performed by calculation and setting of a new total mass flow/mass of the material stream/holdup and new phase fractions (according to the new mass flow/mass of the specified phase). Negative values before setting will be converted to *0*. If there is no specified time point or phase in the material stream/holdup, the value will not be set. *Basis* is a basis of value (*BASIS_MASS* [*kg/s* or *kg*] or *BASIS_MOLL* [*mol/s* or *mol*])

BASIS_MASS	BASIS_MOLL
$\boldsymbol{m = m + (Value - m_i)}$ $\boldsymbol{m_i = Value}$ $\boldsymbol{w_i = \frac{m_i}{m}}$ – for each defined phase ***Value***in [kg/s] or [kg]	$m = m + (\boldsymbol{Value} \cdot \mathrm{M_i} - m_i)$ $m_i = \boldsymbol{Value} \cdot \mathrm{M_i}$ $w_i = \frac{m_i}{m}$ – for each defined phase ***Value***in [mol/s] or [mol]

m – total mass flow/mass of the material stream/holdup

m_i– mass flow/mass of the phase *i*
w_i – mass fraction of the phase *i*
M_i – molar mass of the phase *i*

double GetSinglePhaseProp *(double Time, unsigned Property, unsigned Phase, unsigned Basis = BASIS_MASS)*
Returns non-constant physical property value for the phase mixture *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the specified time point. If such time point has not been defined, interpolation of data will be done. *Property* is an identifier of a physical property. *Basis* is a basis of results (*BASIS_MASS* or *BASIS_MOLL*).

Possible properties (Table 2):

- *FLOW* / *MASS*
- *TEMPERATURE*
- *PRESSURE*
- *PHASE_FRACTION, FRACTION*
- *MOLAR_MASS*
- *DENSITY*
- *HEAT_CAPACITY_CP*
- *THERMAL_CONDUCTIVITY*
- *VISCOSITY*
- *VAPOR_PRESSURE*
- *PERMITTIVITY*
- *ENTHALPY*
- *TP_PROP_USER_DEFINED_XX*

FLOW / *MASS*

Refer to GetMassFlow / GetMass functions.

TEMPERATURE

Refer to GetTemperatute function.

PRESSURE

Refer to GetPressure function.

PHASE_FRACTION, FRACTION

BASIS_MASS	BASIS_MOLL
$\boldsymbol{Ret = w_i}$	$Ret = \frac{w_i}{M_i \sum_j \frac{w_j}{M_j}}$

Ret – value to return
w_i – mass fraction of the phase
M_i – molar mass of the phase

MOLAR_MASS

$$\frac{1}{M} = \sum_i \frac{w_i}{M_i}$$

M – molar mass of the phase [kg/mol]
M_i – molar mass of the compound *i*
w_i – mass fraction of compound *i* in phase

DENSITY, HEAT_CAPACITY_CP, THERMAL_CONDUCTIVITY, VISCOSITY, VAPOR_PRESSURE, PERMITTIVITY, TP_PROP_USER_DEFINED_XX

Refer to GetPhaseTPDProp function.

ENTHALPY

	BASIS_MASS	BASIS_MOLL
Solid and liquid phase	$h = h_0 + Cp \cdot \Delta T + \frac{M}{\rho}(P - P_0)$	$h = h_0 + Cp \cdot \Delta T + \frac{M}{\rho}(P - P_0)$

	$H = \sum_i \frac{h_i \cdot f_i}{M_i}$ [J/kg/s] or [J/kg]	$H = \sum_i h_i \cdot f_i$ [J/mol/s] or [J/mol]
Vapor phase	$h = h_0 + Cp \cdot \Delta T$ $H = \sum_i \frac{h_i \cdot f_i}{M_i}$ [J/kg/s] or [J/kg]	$h = h_0 + Cp \cdot \Delta T$ $H = \sum_i h_i \cdot f_i$ [J/mol/s] or [J/mol]

H – enthalpy of the phase [J/kg/s], [J/mol/s] / [J/kg], [J/mol]
h_i – enthalpy of the compound *i* [J/mol]
f_i – mass fraction of the compound *i* in phase
M_i – molar mass of the compound *i*
h_0 – formation enthalpy [J/mol]
Cp – heat capacity for constant pressure of the compound
ΔT – difference between the temperature at normal conditions (298.15 K) and current temperature
P – current pressure
P_0 – pressure at normal conditions (101325 Pa)

void SetSinglePhaseProp *(double Time, unsigned Property, unsigned Phase, double Value, unsigned Basis = BASIS_MASS)*

Sets non-constant physical property value for phase mixture *Phase* (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the specified time point. If there is no specified time point or phase in the material stream/holdup, the value will not be set. *Property* is an identifier of a physical property. *Basis* is a basis of value (*BASIS_MASS* or *BASIS_MOLL*).

Possible properties:

- *FLOW* / *MASS*
- *FRACTION*

FLOW / *MASS*

Refer to SetPhaseMassFlow / SetPhaseMass functions.

FRACTION

$$w_i = Value$$

w_i – mass fraction of the phase *i*

double GetPhaseTPDProp *(double Time, unsigned Property, unsigned Phase)*

Returns value of temperature/pressure-dependent physical property for specified phase (*SOA_SOLID, SOA_LIQUID, SOA_VAPOR*) for the time point *Time*. If such time point has not been defined, interpolation of data will be done.

Possible properties (Table 4):

- *HEAT_CAPACITY_CP*
- *THERMAL_CONDUCTIVITY*
- *VAPOR_PRESSURE*
- *VISCOSITY*
- *DENSITY*
- *PERMITTIVITY*
- *ENTHALPY*
- *TP_PROP_USER_DEFINED_XX*

HEAT_CAPACITY_CP

$$Cp = \sum_i w_i Cp_i$$

Cp – heat capasity of the phase [J/(kg·K)]
Cp_i – heat capasity of compound *i*
w_i – mass fraction of compound *i* in phase

THERMAL_CONDUCTIVITY

Solid phase	$\lambda = \sum_i w_i \lambda_i$
Liquid phase	$\lambda = \frac{1}{\sqrt{\sum_i x_i \lambda_i^{-2}}}$
Vapor phase	$\lambda = \sum_i \frac{x_i \lambda_i}{\sum_j x_j F_{i,j}}$; $F_{i,j} = \frac{\left(1 + \sqrt{\frac{\lambda_i}{\lambda_j}}\sqrt[4]{\frac{M_j}{M_i}}\right)^2}{\sqrt{8\left(1 + \frac{M_i}{M_j}\right)}}$

λ – thermal conductivity of the phase [W/(m·K)]
λ_i – thermal conductivity of compound *i*
w_i – mass fraction of compound *i* in phase
M_i – molar mass of compound *i*

VAPOR_PRESSURE

$$Pv = \min_i Pv_i$$

Pv – vapor pressure of the phase [Pa]
Pv_i – vapor pressure of compound *i*

VISCOSITY

Solid phase	$\eta = \sum_i w_i \eta_i$
Liquid phase	$\ln \eta = \frac{\sum_i w_i \ln \eta_i}{\sum_i w_i}$
Vapor phase	$\eta = \frac{\sum_i x_i \sqrt{M_i} \eta_i}{\sum_i x_i \sqrt{M_i}}$

η – viscosity of the phase [Pa·s]
η_i – viscosity of compound *i*
w_i – mass fraction of compound *i* in phase
x_i – mole fraction of compound *i* in phase
M_i – molar mass of compound *i*

DENSITY

Solid phase	$\rho = \sum_{i,j} \rho_i (1 - \varepsilon_j) f_{i,j}$
Liquid and vapor phase	$\frac{1}{\rho} = \sum_i \frac{w_i}{\rho_i}$

ρ – density of the phase [kg/m^3]
ρ_i – density of compound *i*
ε_j – porosity in interval *j*
$f_{i,j}$ – mass fraction of compound *i* with porosity *j*
w_i – mass fraction of compound *i* in phase

PERMITTIVITY

$$\varepsilon = \sum_i w_i \varepsilon_i$$

ε – permittivity of the phase [J/(kg·K)]
ε_i – permittivity of the compound *i*
w_i – mass fraction of the compound *i* in phase

ENTHALPY

$$H = \sum_i w_i H_i$$

H – enthalpy of the phase [J/kg]
H_i – enthalpy of the compound *i*
w_i – mass fraction of the compound *i* in phase

TP_PROP_USER_DEFINED_XX

$$Y = \sum_i w_i Y_i$$

Y – property value of the phase
Y_i – property value of the compound *i*
w_i – mass fraction of the compound *i* in phase

Functions to work with solid distributed properties

double GetFraction *(double Time, std::vector<unsigned> Coords)*
Returns solid mass fraction by specified coordinates according to all defined distributions. If such time point has not been defined, interpolation of data will be done.

void SetFraction *(double Time, std::vector<unsigned> Coords, double Value)*
Sets solid mass fraction by specified coordinates according to all defined distributions. If such time point has not been defined in the material stream/holdup, nothing will be done. Direct setting of fractions to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool GetDistribution *(double Time, EDistrTypes Dim, std::vector<double>& Result)*
Returns vector of distributed property for specified time point *Time* and dimension *Dim*. If such time point has not been defined in the material stream/holdup, then linear interpolation will be used to obtain data. Returns *false* on error.

bool GetDistribution *(double Time, EDistrTypes Dim1, EDistrTypes Dim2, CDense2DMatrix& Result)*
Returns matrix of two distributed dependent properties *Dim1* and *Dim2* for the specified time point *Time*. If such time point has not been defined in the material stream/holdup, then linear interpolation will be used to obtain data. Rows of resulting matrix will correspond to *Dim1*, columns – to *Dim2*. Returns *false* on error.

bool GetDistribution *(double Time, std::vector<EDistrTypes> Dims, CDenseMDMatrix& Result)*
Returns multidimensional matrix of distributed dependent properties for specified time point *Time* and dimensions *Dims*. If such time point has not been defined in the material stream/holdup, then linear interpolation will be used to obtain data. Returns *false* on error.

bool GetDistribution *(double Time, EDistrTypes Dim, std::string Compound, std::vector<double>& Result)*
Returns vector of distributed property for specified time point *Time*, dimension *Dim* and compound *Compound*. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*).

If specified compound has not been defined in the material stream/holdup, nothing will be done. If specified time point has not been defined, then linear interpolation will be used to obtain data. Returns *false* on error.

bool GetDistribution *(double Time, EDistrTypes Dim1, EDistrTypes Dim2, std::string Compound, CDense2DMatrix& 2DResult)*
Returns matrix of two distributed dependent properties *Dim1* and *Dim2* for specified compound *Compound* and time point *Time*. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*). If specified compound has not been defined in the material stream/holdup, nothing will be done. If specified time point has not been defined, then linear interpolation will be used to obtain data. Rows of resulting matrix will correspond to *Dim1*, columns – to *Dim2*. Returns *false* on error.

bool GetDistribution *(double Time, std::vector<EDistrTypes> Dims, std::string Compound, CDenseMDMatrix& MDResult)*
Returns multidimensional matrix of distributed dependent properties for specified time point *Time*, dimensions *Dims* and compound *Compound*. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*). If specified compound has not been defined in the material stream/holdup, nothing will be done. If specified time point has not been defined, then linear interpolation will be used to obtain data. Returns *false* on error.

bool SetDistribution *(double Time, EDistrTypes Dim, std::vector<double> Distr)*
Sets distributed property *Distr* of type *Dim* for specified time point *Time*. If such time point or dimension doesn't exist nothing will be done. Returns *false* on error. Direct setting of distribution to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool SetDistribution *(double Time, EDistrTypes Dim1, EDistrTypes Dim2, CDense2DMatrixDistr)*
Sets matrix *Distr* of two dependent distributed properties of types *Dim1* and *Dim2* for specified time point *Time*. If such time point or dimensions don't exist nothing will be done. Returns *false* on error. Direct setting of distribution to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool SetDistribution *(double Time, CDenseMDMatrix Distr)*
Sets multidimensional matrix *Distr* of dependent distributed properties for specified time point *Time*. If such time point or dimensions, which are specified in *Distr*, don't exist, nothing will be done. Returns *false* on error. Direct setting of distribution to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool SetDistribution *(double Time, EDistrTypes Dim, std::string Compound, std::vector<double> Distr)*
Sets distributed property *Distr* of type *Dim* for specified compound *Compound* and time point *Time*. If such time point, compound or dimension doesn't exist nothing will be done. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*). Returns *false* on error. Direct setting of distribution to the holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool SetDistribution *(double Time, EDistrTypes Dim1, EDistrTypes Dim2, std::string Compound, CDense2DMatrix 2DDistr)*
Sets matrix *2DDistr* of two dependent distributed properties of types *Dim1* and *Dim2* for specified compound *Compound* and time point *Time*. If such time point, compound or dimensions don't exist nothing will be done. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*). Returns *false* on error. Direct setting of distribution to the holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool SetDistribution *(double Time, std::string Compound, CDenseMDMatrix MDDistr)*
Sets multidimensional matrix *MDDistr* of dependent distributed properties for specified compound *Compound* and time point *Time*. If such time point, compound or dimensions, which are specified in *MDDistr*, don't exist, nothing will be done. Input dimensions should not include distribution by compounds (*DISTR_COMPOUNDS*). Returns *false* on error. Direct setting of distribution to the holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this.

bool ApplyTM *(double Time, CTransformMatrix Transformation)*
Transforms matrix of distributed parameters of solids for time point *Time* by applying a movement matrix *Transformation*. Returns true if the transformation was successful.

bool ApplyTM *(double Time, std::string Compound, CTransformMatrix Transformation)*
Transforms matrix of distributed parameters of solids for specified compound *Compound* and time point *Time* by applying a movement matrix *Transformation*. Dimensions of transformation matrix should not include distribution by compounds (*DISTR_COMPOUNDS*). Returns true if the transformation was successful.

void NormalizeDistribution *(double Time)*
Normalizes data in solid distribution matrix at the specified time point. If time point *Time* has not been defined, nothing will be done.

void NormalizeDistribution *(double Start, double End)*
Normalizes data in solid distribution matrix in each time point from interval.

void NormalizeDistribution *()*
Normalizes data in solid distribution matrix in all defined time points.

Functions to work with particle size distributions

std::vector<double> GetPSD *(double Time, EPSDType PSDType, EPSDGridType PSDGridType = EPSDGridType::DIAMETER)*
Returns particle size distribution of the total mixture of the solid phase at the time point *Time*. *PSDGridType* defines grid units, if needed: *EPSDGridType::DIAMETER* [m] / *EPSDGridType::VOLUME* [m^3]. *PSDType* is a type of distribution. Possible PSDTypes: *PSD_q0*, *PSD_Q0*, *PSD_q2*, *PSD_Q2*, *PSD_q3*, *PSD_Q3*, *PSD_MassFrac*, *PSD_Number*. PSD data is originally stored in a form of mass fractions and all transformations are performed by the following equations:

PSD_MassFrac

Returns the size distribution in the form of mass fractions with the total sum of 1.

PSD_q3

Mass related distribution of particles.

$$q3_i = \frac{w_i}{\Delta d_i}$$

PSD_Q3

$$Q3_0 = w_0$$

$$Q3_i = Q3_{i-1} + w_i$$

PSD_q0

Number related distribution of particles.

$$q0_i = \frac{N_i}{N_{tot} \cdot \Delta d_i}$$

PSD_Q0

$$Q0_i = Q0_{i-1} + q0_i \cdot \Delta d_i$$

PSD_q2

Surface area related distribution of particles.

$$q2_i = \frac{Q2_i - Q2_{i-1}}{\Delta d_i}$$

PSD_Q2

$$Q2_i = \frac{\sum_{j=0}^{i} N_j \pi d_j^2}{\sum_j N_j \pi d_j^2}$$

PSD_Number

Obtaining of number related distribution of particles depends on several conditions. Three cases of calculation can be distinguished:

1. If only one compound specified

$$N_i = \frac{m_i}{\rho \frac{\pi}{6} d_i^3}$$

2. For several compounds

$$N_i = \sum_j \frac{M_{tot} \cdot w_{i,j}}{\frac{\pi \cdot d_i^3}{6} \cdot \rho_j}$$

3. If distribution by particle porosity has been defined

$$N_i = \sum_j N_{i,j}$$

$$N_{i,j} = \sum_k \frac{M_{tot} \cdot w_{i,j,k}}{\frac{\pi \cdot d_i^3}{6} \cdot \rho_j \cdot (1 - \varepsilon_k)}$$

i – index of size classes
j – index of compounds
k – index of porosities
d_i – particle diameter of class i
Δd_i – size of the class i
m_i – mass of particles of class i
M_{tot} – total mass of particles
N_i – number of particles of class i
$N_{i,j}$ – number of particles of compound j with size class i
N_{tot} – total number of particles
w_i – mass fraction of particles of class i
$w_{i,j}$ – mass fraction of particles of compound j with size class i
$w_{i,j,k}$ – mass fraction of particles of compound j with size class i and porosity k
ρ_j – density of compound j
ε_k – porosity of class k
$q0$ – number related density distribution
$Q0$ – number related cumulative distribution

$q2$ – surface area related density distribution
$Q2$ – surface area related cumulative distribution
$q3$ – mass related density distribution
$Q3$ – mass related cumulative distribution

std::vector<double> GetPSD *(double Time, EPSDType PSDType, std::string Compound, EPSDGridType PSDGridType = EPSDGridType::DIAMETER)*
Returns particle size distribution of compound *Compound* of the solid phase of the material stream/holdup at the time point *Time*. *_PSDType* is a type of distribution. Refer to GetPSD(Time, PSDType, PSDGridType) function for used equations.

void SetPSD *(double Time, EPSDType PSDType, std::vector<double> PSD, EPSDGridType PSDGridType = EPSDGridType::DIAMETER)*
Sets particle size distribution *PSD* with type *PSDType* to the solid phase of the material stream/holdup for time point *Time*. Direct setting of PSD to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this. Possible PSDTypes: *PSD_q0*, *PSD_Q0*, *PSD_q2*, *PSD_Q2*, *PSD_q3*, *PSD_Q3*, *PSD_MassFrac, PSD_Number*. Note using *PSD_Number*: If the total mass of the particles given in the number distribution differs from the material stream/holdup's total particle mass, the number will be normalized to match the material stream/holdup's particle mass. As mass fractions are used to store data, *PSD* will be converted using functions Convertq0ToMassFractions(), ConvertQ0ToMassFractions(), Convertq3ToMassFractions(), ConvertQ3ToMassFractions(), ConvertNumbersToMassFractions() (refer to 'PSD functions.pdf'). *PSDGridType* defines grid units, if needed: *EPSDGridType::DIAMETER* [m] / *EPSDGridType::VOLUME* [m^3].

void SetPSD *(double Time, EPSDType PSDType, std::string Compound, std::vector<double> PSD, EPSDGridType PSDGridType = EPSDGridType::DIAMETER)*
Sets particle size distribution *PSD* with type *PSDType* for the specific compound *Compound* to the solid phase of the material stream/holdup for time point *Time*. Direct setting of PSD to the material stream/holdup leads to a change of all dependent distributions. Approach with transformation matrix should be used to avoid this. Possible PSDTypes: *PSD_q0*, *PSD_Q0*, *PSD_q2*, *PSD_Q2*, *PSD_q3*, *PSD_Q3*, *PSD_MassFrac, PSD_Number*. Note using *PSD_Number*: If the total mass of the particles given in the number distribution differs from the material stream/holdup's total particle mass, the number will be normalized to match the material stream/holdup's particle mass. As mass fractions are used to store data, *PSD* will be converted using functions Convertq0ToMassFractions(), ConvertQ0ToMassFractions(),Convertq2ToMassFractions(), ConvertQ2ToMassFractions(), Convertq3ToMassFractions(), ConvertQ3ToMassFractions(), ConvertNumbersToMassFractions() (refer to 'PSD functions.pdf'). *PSDGridType* defines grid units, if needed: *EPSDGridType::DIAMETER* [m] / *EPSDGridType::VOLUME* [m^3].

Functions to work with lookup tables

*CLookupTable** GetLookupTable *(ECompoundTPProperties Property, EDependencyTypes DependencyType, double Time)*
Creates (if not yet exists), fills with compounds fractions and returns a corresponding lookup table for the specified *Property* (Table 4), *DependencyType* (DEPENDENCE_TEMP or DEPENDENCE_PRES) and *Time*. For more information, refer to "Thermodynamics.pdf".

double CalcTemperatureFromProperty *(ECompoundTPProperties Property, double Time, double Value)*
Reads the temperature from the corresponding lookup table for a specific *Value* of the selected *Property* (Table 4) at the corresponding *Time* point. For more information, refer to "Thermodynamics.pdf".

double CalcPressureFromProperty *(ECompoundTPProperties Property, double Time, double Value)*
Reads the pressure from the corresponding lookup table for a specific *Value* of the selected *Property* (Table 4) at the corresponding *Time* point. For more information, refer to “Thermodynamics.pdf”.

double CalcPropertyFromTemperature *(ECompoundTPProperties Property, double Time, double T)*
Reads the value of the specified *Property* (Table 4) from the corresponding lookup table for the given temperature *T*. For more information, refer to “Thermodynamics.pdf”.

double CalcPropertyFromPressure *(ECompoundTPProperties Property, double Time, double P)*
Reads the value of the specified *Property* (Table 4) from the corresponding lookup table for the given pressure *P*. For more information, refer to “Thermodynamics.pdf”.

Functions to work with other streams

void CopyFromStream *(CMaterialStream *SrcStream, double Time, bool DeleteDataAfter = true)*
Copies all stream data from *SrcStream* for specified time point *Time* to the current material stream. Before copying all data after the time point *Time* in the destination stream can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromStream *(CMaterialStream *SrcStream, double Start, double End, bool DeleteDataAfter = true)*
Copies all stream data from *SrcStream* on the certain time interval to the current material stream. Boundary points *Start* and *End* are included into this interval. Before copying, all data after the time point *Start* in the destination stream can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromStream *(double TimeDst, CMaterialStream *SrcStream, double TimeSrc, bool DeleteDataAfter = true)*
Copies all stream data from time point *TimeSrc* of material stream *SrcStream* to the time point *TimeDst* of this material stream. Before copying, all data after the time point *TimeDst* in the destination material stream can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromHoldup *(CHoldup *SrcHoldup, double Time, double MassFlow, bool DeleteDataAfter = true)*
Copies all data from *SrcHoldup* for the specified time point to the current material stream and sets mass flow *MassFlow*. Before copying, all data after the time point *Time* in the destination material stream can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromHoldup *(double TimeDst, CHoldup *SrcHoldup, double TimeSrc, double MassFlow, bool DeleteDataAfter = true)*
Copies all stream data from time point *TimeSrc* of holdup *SrcHoldup* to the time point *TimeDst* of this material stream with setting of new mass flow *MassFlow*. Before copying, all data after the time point *TimeDst* in the destination stream can be removed if flag *DeleteDataAfter* is set to *true*.

void AddStream *(CMaterialStream *Stream, double Time)*
Performs a mixing of this material stream with material stream *Stream* for the specified time point *Time*.

void AddStream *(CMaterialStream *Stream, double Start, double End, unsigned TPType = BOTH_TP)*
Performs a mixing of this material stream with material stream *Stream* for the specified time interval. Boundary points *Start* and *End* are included into this interval. Parameter *TPType* specifies which time

points will be present in the resulting stream (combining points from two streams (*BOTH_TP*), only from the first stream (*DST_TP*), or only from the second stream (*SRC_TP*)). Data for non-existent points are obtained by linear interpolation.

void CopyFromHoldup *(CHoldup *SrcHoldup, double Time, bool DeleteDataAfter = true)*

Copies all holdup data from *SrcHoldup* for the specified time point *Time* to the current holdup. Before copying, all data after the time point *Time* in the destination holdup can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromHoldup *(CHoldup *SrcHoldup, double Start, double End,*
bool DeleteDataAfter = true)

Copies all holdup data from *SrcHoldup* on the certain time interval to the current holdup. Boundary points *Start* and *End* are included into this interval. Before copying, all data after the time point *Start* in the destination holdup can be removed if flag *DeleteDataAfter* is set to *true*.

void CopyFromHoldup *(double TimeDst, CHoldup *SrcHoldup, double TimeSrc,*
bool DeleteDataAfter = true)

Copies all holdup data from time point *TimeSrc* of holdup *SrcHoldup* to the time point *TimeDst* of this holdup. Before copying, all data after the time point *TimeDst* in the destination holdup can be removed if flag *DeleteDataAfter* is set to *true*.

void AddHoldup *(CHoldup *Holdup, double Time)*

Performs a mixing of this holdup with holdup *Holdup* for the specified time point *Time*.

void AddHoldup *(CHoldup *Holdup, double Start, double End, unsigned TPType = BOTH_TP)*

Performs a mixing of this holdup with holdup *Holdup* for the specified time interval. Boundary points *Start* and *End* are included into this interval. Parameter *TPType* specifies which time points will be present in the resulting holdup (combining points from two holdups (*BOTH_TP*), only from the first holdup (*DST_TP*), or only from the second holdup (*SRC_TP*)). Data for non-existent points are obtained by linear approximation.

void AddStream *(CMaterialStream *Stream, double Start, double End)*

Performs a mixing of this holdup with material stream *Stream* for the specified time interval. Boundary points *Start* and *End* are included into this interval. Data for non-existent points are obtained by linear interpolation.

Tables of properties

Table 1 – Overall mixture properties

Name	Units	Define
Mass flow / Mass	kg/s, mol/s or kg, mol	FLOW, TOTAL_FLOW / MASS, TOTAL_MASS
Temperature	K	TEMPERATURE
Pressure	Pa	PRESSURE
Molar mass	kg/mol	MOLAR_MASS
Enthalpy	J/kg/s, J/mol/s	ENTHALPY

Table 2 – Single-phase mixture properties

Name	Units	Define
Mass flow / Mass	kg/s, mol/s or kg, mol	FLOW / MASS
Temperature	K	TEMPERATURE
Pressure	Pa	PRESSURE
Phase fraction	-	PHASE_FRACTION, FRACTION
Molar mass	kg/mol	MOLAR_MASS
Density	kg/m^3	DENSITY
Heat capacity Cp	J/(kg·K)	HEAT_CAPACITY_CP
Thermal conductivity	W/(m·K)	THERMAL_CONDUCTIVITY
Viscosity	Pa·s	VISCOSITY
Vapor pressure	Pa	VAPOR_PRESSURE
Enthalpy	J/kg/s, J/mol/s or J/kg, J/mol	ENTHALPY
Permittivity	F/m	PERMITTIVITY
User defined property	-	TP_PROP_USER_DEFINED_XX

Table 3 – Constant properties for pure compounds

Name	Units	Define
State of aggregation at normal conditions	-	SOA_AT_NORMAL_CONDITIONS
Normal boiling point	K	NORMAL_BOILING_POINT
Normal freezing point	K	NORMAL_FREEZING_POINT
Critical temperature	K	CRITICAL_TEMPERATURE
Critical pressure	Pa	CRITICAL_PRESSURE
Molar mass	kg/mol	MOLAR_MASS
Standard formation enthalpy	J/mol	STANDARD_FORMATION_ENTHALPY
Heat of fusion at normal freezing point	J/mol	HEAT_OF_FUSION_AT_NORMAL_FREEZING_POINT
Heat of vaporization at normal boiling point	J/mol	HEAT_OF_VAPORIZATION_AT_NORMAL_BOILING_POINT
Reactivity type	-	REACTIVITY_TYPE
User defined property	-	CONST_PROP_USER_DEFINED_XX

Table 4 – Temperature-dependent compound properties

Name	Units	Define
Density	kg/m^3	DENSITY
Heat capacity Cp	J/(kg·K)	HEAT_CAPACITY
Vapor pressure	Pa	VAPOR_PRESSURE
Viscosity	Pa·s	VISCOSITY
Thermal conductivity	W/(m·K)	THERMAL_CONDUCTIVITY
Permittivity	F/m	PERMITTIVITY
Enthalpy	J/kg/s, J/mol/s or J/kg, J/mol	ENTHALPY
User defined property	-	TP_PROP_USER_DEFINED_XX

Table 5 – Interaction properties between two pure compounds

Name	Units	Define
Interface tension	N/m	INTERFACE_TENSION
User defined property	-	INT_PROP_USER_DEFINED_XX