CAPE-OPEN

Delivering the power of component software
and open standard interfaces
in Computer-Aided Process Engineering

Thermodynamic and Physical Properties v1.0

www.colan.org

ARCHIVAL INFORMATION

Reference	CO-LaN Thermo Version 1.08.008.DOC
Filename (if different)	
Authors	Daniel Piñol, Hyprotech Juan Carlos Rodriguez, Hyprotech Michael Halloran, AspenTech Werner Drewitz, BASF Richard Szczepanski, Infochem Michel Pons, TotalFinaElf Malcolm Woodman, BP Peter Banks, BP Jasper van Baten, AmsterCHEM
Date	May 2011
Number of Pages	87
Version	Version 1.08.008
Reviewed by (date)	Reviewed by Thermo SIG
Distribution	
Additional Material	
Location on BSCW	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN CAPE-OPEN documents and publications include software in the form of *sample code.* Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN disclaims any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by CO-LaN --- remains with you.

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN CAPE-OPEN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

Microsoft, Microsoft Word, Visual Basic, Visual Basic for Applications, Internet Explorer, Windows and Windows NT are registered trademarks and ActiveX is a trademark of Microsoft Corporation.

Netscape Navigator is a registered trademark of Netscape Corporation.

Adobe Acrobat is a registered trademark of Adobe Corporation.

SUMMARY

This document describes the CAPE-OPEN Thermodynamic and Physical Properties Interfaces. It is an update of the original CAPE-OPEN Thermodynamic and Physical Properties Interface Specification.

The document is consistent with the version 1.0 CAPE-OPEN COM type library as well as with the other elements of the version 1.0 CAPE-OPEN Standard. It also provides recommendations for developers of CO-compliant thermodynamic components and sockets on how to use these deliverables. Interfaces for reactions, electrolytes and petroleum fractions are described in separate specifications.

This document has been updated from published version 1.06 to version 1.08 in order to include errata and clarifications. None of these errata modifies the interface definitions themselves. The version 1.0 IDL available from the CO-LaN web site (http://www.colan.org/) therefore is to be used together with this document.

ACKNOWLEDGEMENTS

We gratefully acknowledge the work of Work Package 2, led by Hans-Horst Mayer of BASF during the CAPE-OPEN project, who produced the original specification from which this document has been developed.

CONTENTS

INTRODUCTION

This document describes the CAPE-OPEN Thermodynamic and Physical Properties Interfaces. It is an update of the original CAPE-OPEN Thermodynamic and Physical Properties Interface Specification and its purpose is to remove the ambiguities identified during interoperability testing.

The remainder of this document recommends an approach to the development and implementation of new Thermodynamic and Physical Property plugs and sockets. The implementation information is based on the current Type Library 1.0, which is also compatible with the previous versions 0.9.0 and 0.9.3. The document finishes with information on units of measure, notes to clarify points that have been found to be confusing in the use of the specification and finally a glossary of terms. Interfaces for reactions, electrolytes and petroleum fractions are described in separate specifications (see http://www.colan.org/index-33.html).

Since the original specification version, a choice has been made to identify chemical species with the term *compounds* rather than *components*. This is to avoid a conflict with the term *components* that is used for software components. This change in terminology is not reflected in the names of methods; e.g. to get the number of compounds from a material object, one would use GetNumComponents.

Implementation of a Unit Operation component is not dealt with here. It is covered by the CAPE-OPEN Unit Operations Interface Specification, plus the example unit operations components used in the Interoperability Demonstration (see http://www.colan.org/index-33.html).

1. Implementation Resources

The following resources are available from the CO-LaN web (http://www.colan.org/) for developers of physical property components and sockets:

Resources for Physical Property Components

- CAPE-OPEN Thermodynamic and Physical Property Interface Specification Version 1.0 (this document)
- The demonstration video, which shows the current look and feel of open simulation interoperability
- The CAPE-OPEN Logging and Testing tool (COLTT), available to CO-LaN members
- The freely available CAPE-OPEN to CAPE-OPEN simulation environment (COCO), http://www.cocosimulator.org

In addition to these specific resources, background information on COM implementation is available in a document called "COM Architecture Overview and Basic Principles". See the CO-LaN web site (http://**www.colan.org**/) for current status and availability of all of these implementation resources.

For new thermodynamic property package implementations, it is advised to use the version 1.1 CAPE-OPEN thermodynamic standard, available from http://www.colan.org/index-16.html.

Resources for Physical Property Sockets

The same resources are available for socket developers. In this case, the first thing required is to develop a Material Object.

2. CAPE-OPEN Identifiers

In this document, CAPE-OPEN Methods and Tools recommendations are followed for property identifiers and method names, i.e. property identifiers start with a lower case letter and method names start with an upper case letter. (See: http://www.colan.org/Spec%2010/Methods&Tools%20Integrated%20Guidelines.pdf)

However, for actual implementation purposes, to avoid uppercase / lowercase problems, all comparisons of CAPE-OPEN identifiers are case insensitive. Namely:

- Properties
- Phase details
- Flash type details
- Calculation type details

Property values are also case insensitive. The only exception to this rule is the compound constant property chemicalFormula, for which character case is important (e.g. to tell the difference between Cobalt (Co) and carbon monoxide (CO)).

3. CAPE-OPEN Thermodynamic and Physical Properties Interface Specification

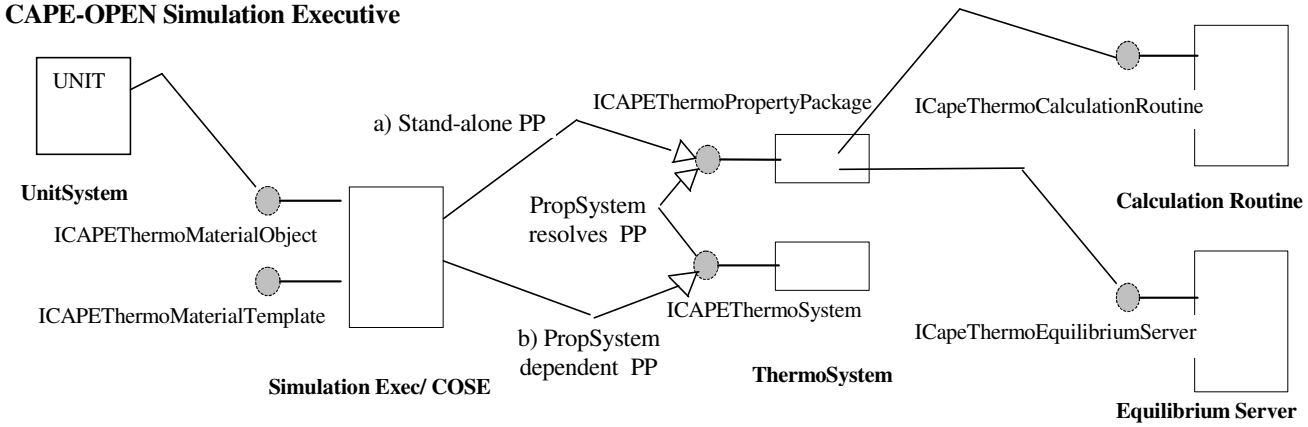
3.1 Introduction

This specification provides practical guidelines for using and implementing CAPE-OPEN Thermodynamic and Physical Properties (THRM) interfaces. As with all CAPE-OPEN Interface Specifications, this specification was developed using Universal Modelling Language (UML). More details of this process can be found in the original THRM document and in other documents available from the CO-LaN web site.

The first sections provide an overview of the interfaces, which is then refined in the subsequent reference section. The document also pseudo-code examples to show how the interfaces are constructed and used.

3.2 Component Diagram & Supporting Interfaces

CAPE-OPEN Component Diagram



The Component Diagram shows the interfaces supported by each of the components in the CAPE-OPEN Thermodynamics and Physical Properties system. The diagram does not show common interfaces that should or may be exposed by software components, like ICapeIdentification and ICapeUtilities and the error common interfaces. The associations, represented by the lines from the components to the interfaces, are also detailed. The way in which these associations of the Component Diagram are implemented is proprietary to the PMC/PME vendor. The diagram shows the two ways that any PME can use to instantiate a Property Package:

Route a), it is a Stand Alone component (not needing to be inside a Thermo System).

Route b), it is Thermo System dependent (needing to be resolved by its Thermo System). In this case it is necessary to resolve the Property Package to get a handle to it.

Each white arrow represents the interface the PME gets.

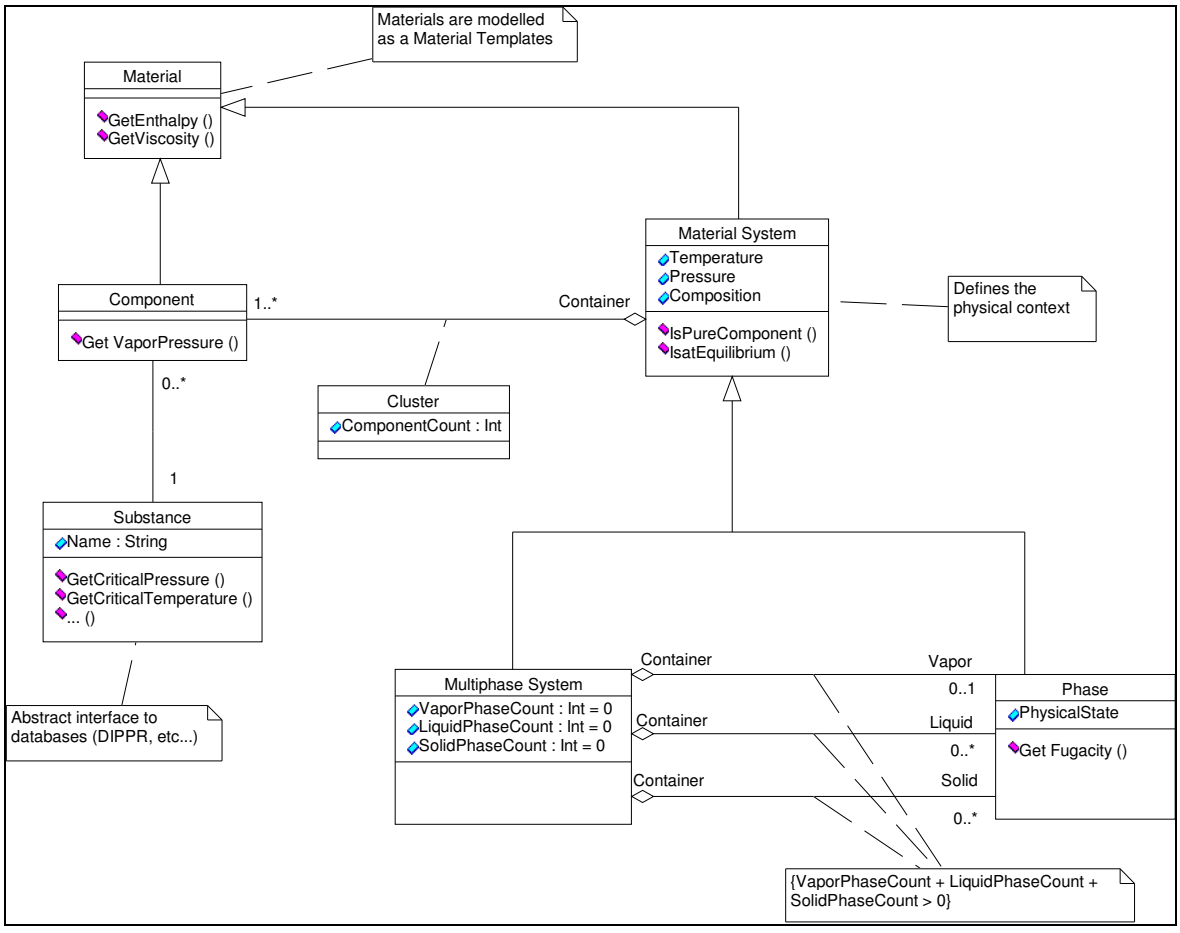
3.2.1 Material Classes - Description

The **UML Material Class Diagram below** shows a possible implementation of a Material Object class. This diagram documents the implementation, rather than the CAPE-OPEN interface view. The Material Template defines the characterisation of a material, and the Material Object defines an instance of material. Material Objects can be created from Material Templates. Material Objects are implemented by a CAPE-OPEN Process Modelling Environment. Process modeling clients like Unit Operations have access to Material Objects that are connected to Material Ports.

Material Template interfaces are – if supported – implemented by a CAPE-OPEN Process Modelling Environment. They are exposed to Process Modeling Clients through the ICapeMaterialTemplateSystem interface, that may be exposed through the simulation context object.

The Material Template definition consists of:

- A compound List ,
- Zero or more present phases,
- Optionally, a reference to a CAPE-OPEN Property Package.



UML Material Class Diagram

The above diagram splits chemical species into compounds and substances. Substances are the actual species, whereas chemical compounds are instances of the substances used in the simulation. We can have more than one compound based on the same concept of a substance. All parameters are ultimately associated with compounds, since any substance parameter could be overridden for a compound. The end user of the

Material Object is only concerned about compounds. The substance branch of the diagram is really implemented *internal* to a property system.

The Material Object is responsible for keeping the total mixture state consistent with the phase states. Assumption: all phases share the same temperature and pressure, so the phase holds composition and phase fraction or amount. Please note that the CalcEquilibrium method is explicitly part of the Material Template/Object definition as an internal implementation detail. Therefore a Material Object can "flash" itself if it needs to.

The Material Object structure is extensible, and will include solids, polymers, etc. So a generic approach was developed to make property calls. A summary of the essential calls follows:

```
CapeError SetProp(in CapeString property, in CapeString phaseQualifier, in
CapeArrayString componentIds, in CapeString calculationType, in CapeString basis, out
CapeArrayDouble propVals);

CapeError CalcProp(in CapeArrayString propList, in CapeArrayString phaseQualifiers, in
CapeString calculationType);

CapeError GetProp(in CapeString property, in CapeString phaseQualifier, in
CapeArrayString componentIds, in CapeString calculationType, in CapeString basis, out
CapeArrayDouble propVals);

CapeError CalcEquilibrium(in CapeString flashType, in CapeArrayString propList);

CapeError GetUniversalConstants(in CapeArrayString constantList, out CapeArrayVariant
propVals);


CapeError GetComponentConstant(in CapeArrayString propList, out CapeArrayDouble
propVals);
```

3.3 General Remarks on Usage of Interface Methods

3.3.1 Convention Note on UNDEFINED and Empty and arrays

Throughout the document UNDEFINED refers to a `NULL BSTR` for string arguments. `UNDEFINED for a VARIANT` argument is of the type `VT_EMPTY`. See below how to implement these values in C++ and in VB for COM:

Type of Data	Declaration and Usage

BSTR	In C++
	Acceptable
	`BSTR strArg = NULL;`
	Not Acceptable
	`BSTR str = ::SysAllocString(L"");`
	In VB
	Acceptable
	`Dim strArg as String`
	`strArg = vbNullString`
	Not Acceptable
	`Dim strArg as String`
	`strArg = ""`

VARIANT	In C++
	`// the vt type of the VARIANT is`
	`// set to VT_EMPTY`
	`VARIANT VarArg;`
	`VariantInit(&VarArg);`
	Remember that VariantInit must always be called after declaring a VARIANT
	In VB
	`Dim VarArg as Variant`

Arrays are defined as VARIANT data types. The element type of an array depends on the data type specified, e.g. CapeArrayDouble contains VT_R8 elements in the array, whereas CapeArrayVariant contains VT_VARIANT elements in the array. For example, property calculactions that can only return real values will use CapeArrayDouble, but result from GetComponentConstant can be strings or real values, hence the return value is CapeArrayVariant. It is advised to have 0 as the lower bound index of an array.

Be aware that the UNDEFINED value depends on the type of the corresponding argument and on the version of IDL used (COM or CORBA). This special value is described in the "Methods & Tools Integrated Guidelines version 1.0" document, available from http://www.colan.org/Spec%2010/Methods&Tools%20Integrated%20Guidelines.pdf.

UNDEFINED is only used when one of the arguments is irrelevant for the particular method, such as basis for the Temperature property. UNDEFINED is never allowed in the property or phases qualifiers.

UNDEFINED must not be used to express a default value.

UNDEFINED must also be used when an argument type is CapeArray and its length is 0 (otherwise VB has problems).

Further information on data types are defined in the “Methods & Tools Integrated Guidelines version 1.0” document, available from
http://www.colan.org/Spec%2010/Methods&Tools%20Integrated%20Guidelines.pdf.

3.3.2 Argument interpretation of Get/Set/CalcProp Standard Methods

See section 5 for more information.

The original specifications of the GetProp and SetProp methods could be interpreted in several ways. This section describes the agreed interpretation of the methods for each property.

VALUES FORMAT

Although the Values argument has VARIANT type in COM and some properties will always return a single value, this argument must always contain an array (possibly with a single element). The proper data type for the Values argument is CapeArrayDouble.

OVERALL INTERPRETATION

If Phases contains "Overall", only the properties that refer to the overall phase will be calculated. To request the values for each particular phase, the particular identifiers of all the phases must be included in the phases argument.

Property	**Phases**	**Comp ID vector**	**Calculation type**	**Return value**
Temperature Pressure etc.	Overall.	UNDEFINED	Mixture	Scalar (i)
enthalpy entropy volume density viscosity thermalConductivity heatCapacity		UNDEFINED	Mixture	Scalar
		UNDEFINED	Pure	Vector of values for all compounds (ii)
		Filled		Vector of values for specified compounds (iii)
fugacityCoefficient activitycoefficient		UNDEFINED	Mixture(*)	Vector of values for all compounds
		Filled		Vector of values for specified compounds
		UNDEFINED	Pure	Vector of values for all compounds
		Filled		Vector of values for specified compounds

Table comments

(i) As stated in the specifications, the value is always an array. Scalar means here that the array will only contain one value

(ii) All compounds means the value of the property for each compound of the Material Object

(iii) The value of the property for each compound specified in argument compIds.

(*)Normally, mixture means that the value is a single scalar that refers to the whole fluid, and pure means that the value is a list of property values for each compound. In these particular properties (`fugacityCoefficient,...`), mixture/pure has a special meaning. In them, "mixture" means that the property values refer to the compounds when they are within a fluid, and "pure" means that the property values refer to the compounds when they are in pure state (not mixed with other compounds).

Although it could sound appealing to use GetComponentConstant instead of GetProp with "pure", it is not the case because these properties depend on the physical conditions of the fluid: temperature, pressure,...

Temperature and Pressure are, as opposed to the version 1.1 specification, not available for individual phases. The proper phase ID to specify for GetProp and SetProp for Temperature and Pressure is Overall. The calculation type should be UNDEFINED.

When a Property Package is requested to perform an overall property calculation, it should not perform an equilibrium calculation. Instead, it should assume that the present phases and compositions, as well as overall pressure and temperature are given. A Property Package typically performs the following operations for performing an overall property calculation

- Get overall temperature and pressure, list of compounds, list of present phases
- Iterate over the present phases
  - Get phase fraction
  - Get the phase composition
  - Calculate the property for the phase at phase composition, pressure and temperature
- Calculate the overall property from contribution of each phase
- Set the calculation result on the overall phase of the Material Object

Derivatives for overall properties should not be provided by Property Packages. The definition of these derivatives is not clear (mostly overall properties are requested at phase equilibrium; the change of the phase equilibrium with the property with respect to which the derivative is taken would in such case need to be taken into account).

HOW TO REQUEST INFORMATION OF NON EXISTING ENTITIES (SUCH AS COMPOUNDS OR PHASES)

Although the CAPE-OPEN standard allows setting a property for a particular phase with the ICapeThermoMaterialObject SetProp method, the Unit Operation developers must be aware that some PMEs may not support this functionality. It can also happen that a CAPE-OPEN compliant Process Modelling Environment (PME) allows setting a property for a particular phase, but only after flashing the stream. It is however highly recommended for a PME to support this calling pattern, as shown in Example 1 (3.8.1).

If any GetProp is called for a particular phase where the phase has not been created, the method call will always fail. To check whether a phase exists or not, it is not safe to use the properties totalFlow and phaseFraction. That is because, at bubble point condition, phaseFraction (with phase=vapor), will be exactly

zero, although the phase exists. One should use ICapeThermoMaterialObject:: PhaseIds to check for phase existence.

There is another similar case where a GetProp call will fail. For example, imagine a particular stream, where only the molar/mass fractions for some (but not all) of the compounds of the stream have been set. If the molar fractions are then requested for all the compounds, the call will fail. That is because, even if the values for some compounds are available, not all of them can be returned. Therefore, it is recommended to always set the molar/mass fractions for all the compounds at the same time, assigning a zero value for the non-existing compounds.

FRACTION AND FLOW

The original specification was ambiguous, since it mentioned properties fraction, flow, molFraction, molFlow. To solve that, the following scheme was agreed.

The thermodynamic standard specification mentions “fraction”, but the list of official properties does not. We have agreed to remove molFraction, molFlow & massFlow from the list of property identifiers, leaving only “flow” and “fraction” and forcing the use of the basis argument. This means that now mass fraction can be requested. Since it was not clear how to specify properties such as the total flow or the vapor fraction, two new properties have been added: totalFlow and phaseFractions.

Property	Calc Type	Comp ID	Phase	Return Value
Fraction	UNDEFINED	UNDEFINED	PH	Vector (all compounds)
Flow		Filled	PH	Vector (specified compounds)
totalFlow	UNDEFINED	UNDEFINED	PH	Vector (one value)
phaseFraction	UNDEFINED	UNDEFINED	PH	Vector (one value)

Note: PH means that phase is defined.

Examples:

- The call to make in order to get a vector with the mole fraction of each compound within the liquid phase (the sum of all values returned will be equal to 1):

```
matObj.GetProp(“fraction”, “liquid”, UNDEFINED, UNDEFINED, "mole")
```

- The call to make to get a vector with the mole fraction of some compounds within the vapor phase (the sum of all values returned will be the fraction that the set of specified compounds represent with respect to the whole vapor phase):

```
matObj.GetProp(“fraction”, “vapor”, (vector of compounds), UNDEFINED,
"mole")
```

- The call to make to get a scalar with the fraction of the fluid that is in the specified phase:

```
matObj.GetProp(“phaseFraction”, “vapor”, UNDEFINED, UNDEFINED, "mole")
```

- The call to make to get a vector with the molar flows of each compound within the liquid phase (the sum of all values should be equal to the total molar flow of the fluid):

```
matObj.GetProp(“flow”, “liquid”, UNDEFINED, UNDEFINED, "mole")
```

- The call to make to get a vector with the molar flow of some compounds within the vapor phase:

```
matObj.GetProp("flow", "vapor", (vector of compounds), UNDEFINED, "mole")
```

- The call to make to get a scalar with the flow (e.g. molar flow) of the fluid that is in the specified phase.

```
matObj.GetProp("totalFlow", "vapor", UNDEFINED, UNDEFINED, "mole")
```

GET/SETINDEPENDENTVAR

In the ICapeThermoMaterialObject interface, since the advantage of using the methods Get/SetIndependentVar instead of GetProp and SetProp is not clear, they should be not be used. Moreover, since they lack the basis argument, the units could be ambiguous for state variables such as enthalpy and flow (molflow & massflow have been removed from the standard).

Without Get/SetIndependentVar, the properties listed in "names of state variables/global variables" (p86 of original specification), namely:

Temperature	gibbsFreeEnergy
Pressure	helmholtzFreeEnergy
Volume	MolFraction (deleted)
Density	moles
Enthalpy	mass
Entropy	Molflow (deleted)
Energy	Massflow (deleted)

have been moved to the Non-constant Properties list, so that all of them can be used in Calc/Set/GetProp

Since in COM it's not feasible to remove a method from an interface and retain binary compatibility, the methods will remain, but must not be used.

SPECIFIC PROPERTIES

The following property identifiers:

enthalpy, enthalpyNF, enthalpyF, entropy, entropyNF, entropyF, energy, gibbsFreeEnergy, helmholtzFreeEnergy, volume

mean intensive properties (which require a specifying "mole" or "mass" for the basis qualifier).

To represent the non-intensive property, such as the energy contained in a given amount of mass, the basis qualifier must have UNDEFINED value. To obtain a non-intensive property however, it is advised to obtain the intensive property and multiply with the amount. For example, to get enthalpy flow, J/s, one can obtain enthalpy using basis "mole" (J/mol) and multiple with totalFlow using basis "mole" (mol/s) to obtain enthalpy flow (J/s).

BASIS CONVERSION

A software component accessing a Material Object may get or set any basis-dependent Physical Property of a Material Object using any basis. It is the Material Object's responsibility to ensure that the software component sees consistent values whatever basis is used. This means that for properties that can have molar or mass basis, the Material Object must:

- Allow a software component to set any basis-dependent Physical Property on either basis.
- Allow a client to get any basis-dependent Physical Property using the basis with which it was stored.
- Perform basis conversions, or delegate basis conversion as necessary. If basis conversion is not meaningful (*e.g.* in the case of cement), the Material Object must be able to return the quantity in its original basis and to return an error should the quantity be requested in a different basis.
- Ensure that quantities set in one basis are consistent with quantities set in another basis, or delegate that function as necessary. Where the basis conversion on a quantity is not feasible, the Material Object must only store the quantity in the basis with which it was set most recently

Basis needs to be specified to GetProp / SetProp for the properties fraction, flow, phaseFraction, totalFlow, density, the intensive properties (see 3.10.3) and any custom properties that are basis dependent.

3.4 CAPE-OPEN Calling Pattern Description

The component interfaces of the thermo system are implemented with The CAPE-OPEN Calling Pattern. The CAPE-OPEN calling pattern provides extensibility by contract between the Process Modelling Environment, the Unit Operations, and the Thermo Systems in a clear and concise way. Open interfaces accessing these components need to be maintained and extended in the context of the CAPE-OPEN project. All existing thermo properties can be supported through this calling pattern. With this approach, new and user defined properties can be added to thermo / unit contract without changing any code.

In the case of the CAPE-OPEN Calling Pattern the contracts between the Process Modelling Environment, Unit Operations, and Thermo Systems are separated from the software calling mechanism. This allows the following advantages:

- Standard Properties and Calculations can be added without software changes.
- Properties are easily bundled for performance (single calculations can still be supported).
- Pattern is consistent for all Thermo System Components. This eases the understanding and usage of the CAPE-OPEN standard.
- Contract is maintained consistently between Unit & Thermo System.
- Complexity is reduced.[1]
- Standard Maintainability & Extensibility is provided.
- Network issues are more easily managed (calculations can be bundled and passed to a server).
- User Defined Properties and Constants are easily supported, maintained and extended.

[1] Brown, Malveau, McCormick, Mowbray; Anti Patterns Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998

3.5 CAPE-OPEN Calling Pattern & Material Object

The following pattern details the way in which the Material Object is used to execute calls on the corresponding Thermo System. This means that the way in which values are set, calculated and retrieved is consistent for all Thermo System components. This pattern is documented below and detailed in code examples later in the document.

1. Step 1: Declare Material Object and set Independent Variables

Independent variables are set on the Material Object for Flash calculations, using the SetProp method. In most cases, two state variables of the material object will be set, for example temperature or pressure.

2. Step 2: Set Values

The client of the Material Object adds properties and corresponding values.

3. Step 3: Calculate

The appropriate calculations are set as strings on the parameter list and the appropriate generic calculation routine is called.

CalcEquilibrium

CalcProp

4. Step 4: Get Results

After results are calculated, the values are then retrieved from the Material Object using the generic Material Object GetProp method. Results are further qualified for phase, compounds, calculation type, and basis. Property results are in SI units (detailed information available at http://www.bipm.fr/enus/3_SI/si.html).

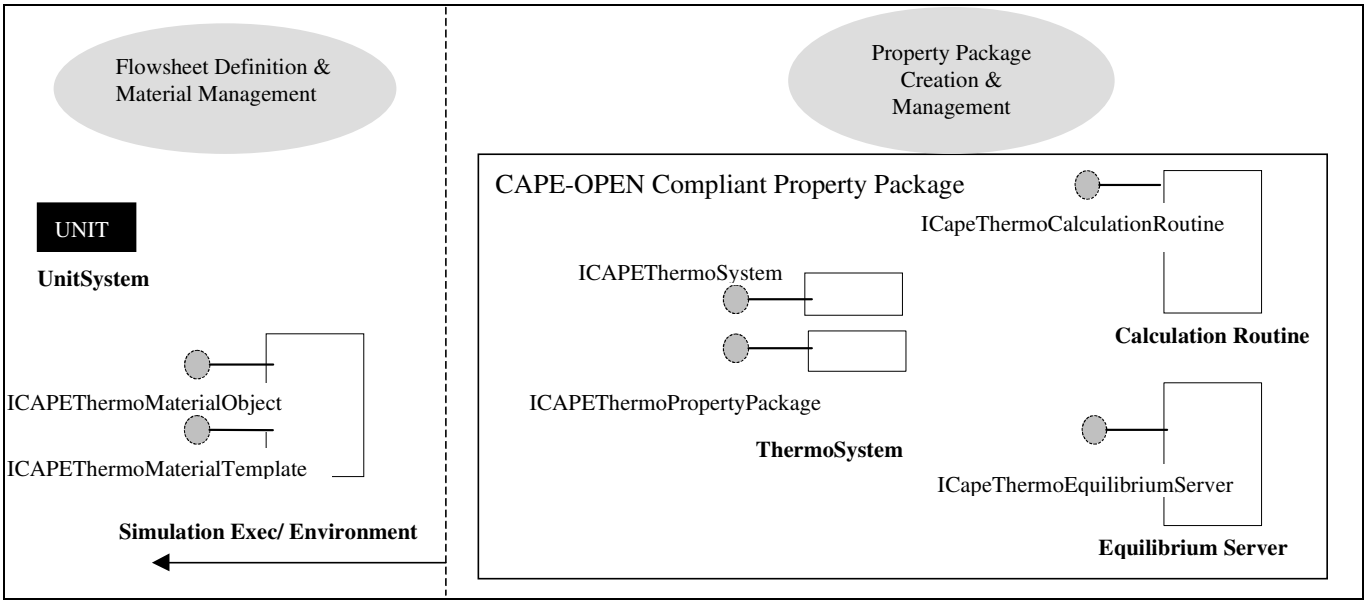
Specific code examples of this pattern are included in this document and are provided by Werner Drewitz of BASF.

The CAPE-OPEN calling pattern significantly reduces the complexity of the integration with existing native Thermo Systems by reducing the number of calls. It also allows for the interfaces and contracts between these systems to be modified without addressing software issues.

3.6 CAPE-OPEN Use Case Driven Component Model

The actual creation and management of the Material Templates is the responsibility of the Simulation Executive/Environment. The Material Template acts as a class factory for the Material Object. The Material Object represents an instance of a Material and provides access to both the state of the Material and the behavior of the Material. The Unit uses the Material Object in the simulation environment in order to calculate properties for a given Material.

CAPE-OPEN Component Model – Use Case Driven

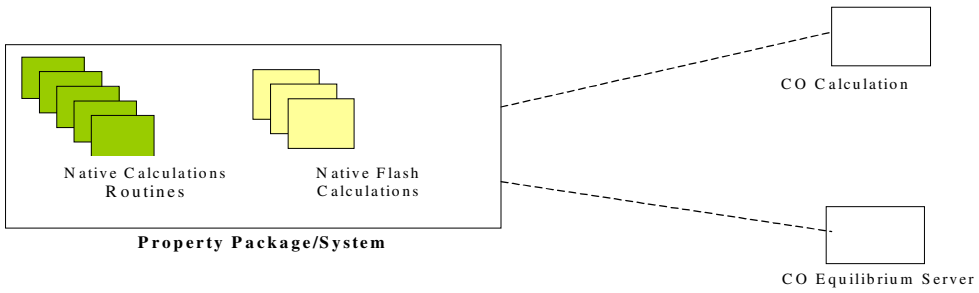


The attached model depicts the CAPE-OPEN components and interfaces in the context of CAPE-OPEN defined workflow. Property Package Creation & Management is the process by which existing Property Packages are made CAPE-OPEN compliant and properly registered. Flowsheet Definition requires that materials be properly defined using CAPE-OPEN Property Packages. These Materials are then assigned to units in the Flowsheet Definition stage. Further details to the functional flow and the actors of this functionality can be found in the Thermo Use Case document. The simulation executive, or CAPE-OPEN Simulation Executive (COSE), is responsible for implementing the interfaces of the MaterialTemplate and MaterialObject. In addition, the COSE provides functionality for defining the Material Template and handles the delegation of the MaterialObject to the appropriate Thermo System and Property Package interfaces.

It is important to point out that the CAPE-OPEN compliant Calculation Routines and Equilibrium Servers (Flash Calculations) are typically only a small portion of the full property package. The majority of the property package is comprised of the native routines, data, parameters and flash calculations of existing and native Property Systems/Packages. A more accurate portrayal of the actual property package follows. The combination of Equilibrium Servers and Calculation Routines, along with the proprietary structure of the property package provide the structure for a CAPE-OPEN compliant property package.

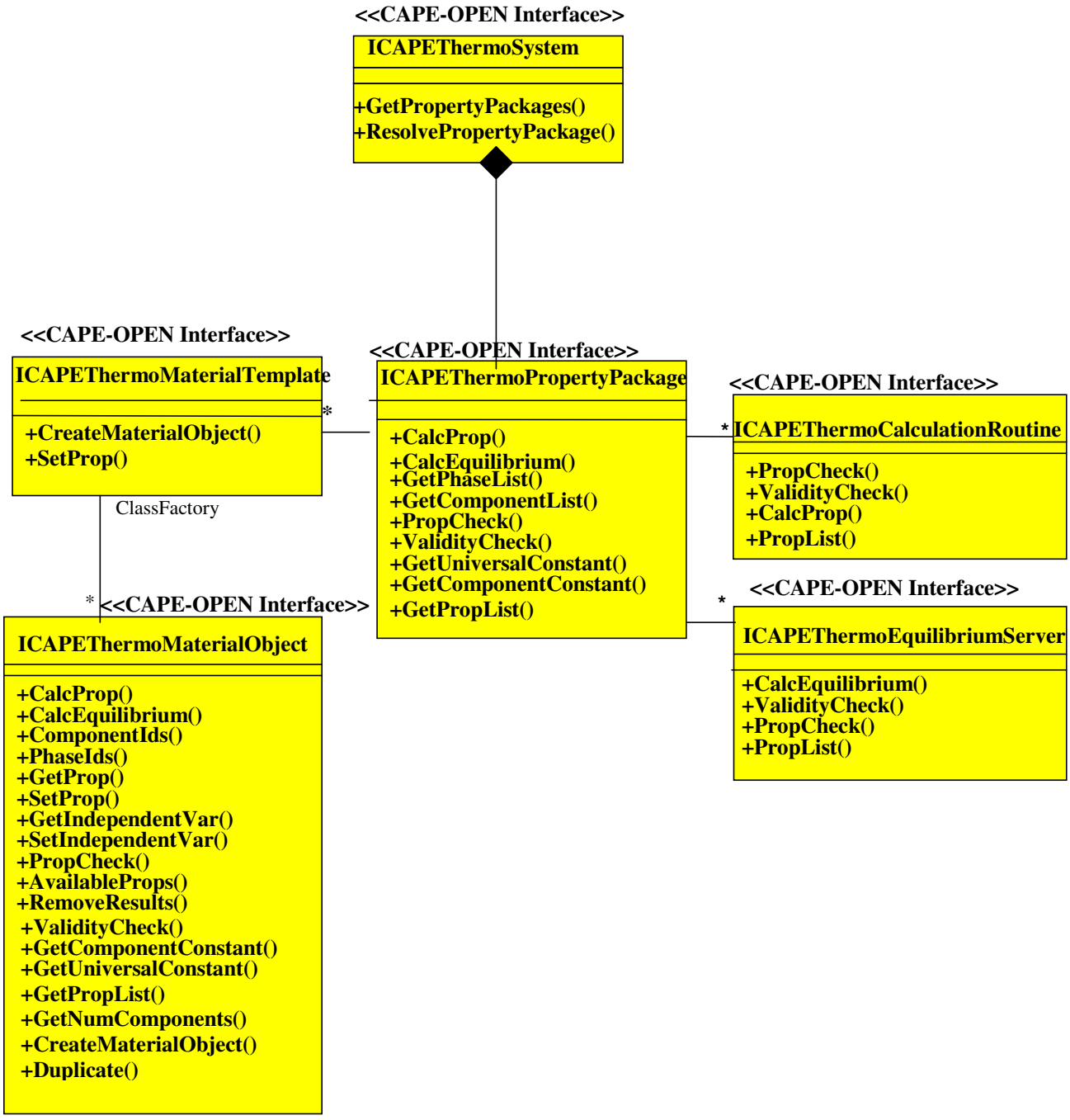
3.6.1 Native Property Package Diagram

It is important to note that making a property package CAPE-OPEN compliant does not upset the native structure of a property package. The full capabilities of a commercial property system are still available in a CAPE-OPEN property package.



3.7 CAPE-OPEN Thermo Interface Diagram

This diagram does not describe how or when these interfaces are executed in the context of a working CAPE-OPEN Environment. The mechanisms by which these underlying associations are executed are proprietary to the simulation environments. This diagram represents a general abstract view of the interfaces. The full overview of the interfaces is described in both the Component Diagram and the Interface Diagram.



A more detailed view can be found in the corresponding IDL and code samples. It is very important to note that the interface diagrams expose the necessary interfaces for using plug and play components. These diagrams do not imply the internal traversal path for how these interfaces are executed.

3.8 Code Sample of CAPE-OPEN Calling Pattern & Material Object

The following pseudo code example is detailed for the Material Object.

Example 1: Calling of liquid enthalpy property of a mixture

3.8.1 Declare Material Object

```
//create a material object
Set Imo = MaterialTemplate.CreateMaterialObject();
```

5. Step 1: Set Values

```
CapeArrayDouble T[1], P[1], F[100];
CapeArrayString phaseQualifiers[1];

T[0] = 373;                          // initialize temperature
P[0] = 101325;                       // initialize pressure

F[0] = 0.1;                          // initialize liquid composition
F[1] = 0.7;                          // initialize liquid composition
F[2] = 0.2;                          // initialize liquid composition

Strcpy(phaseQualifiers[0], ”Liquid”);     // set phase qualifier

// set temperature and pressure on the material object
Imo.SetProp(”temperature”, ”Overall”, UNDEFINED, UNDEFINED, UNDEFINED, T);
Imo.SetProp(”pressure”, ”Overall”, UNDEFINED, UNDEFINED, UNDEFINED, P);
// set liquid composition on the material object
Imo.SetProp(”fraction”, phaseQualifiers, UNDEFINED, UNDEFINED, ”mole”, F);
```

6. Step 2: Calculate Mixture Property

```
CapeArrayString properties[2];      //create array for properties.
CapeString calculationType;

Strcpy(calculationType, ”Mixture”); // set calculation type

Strcpy(properties[0], ”Enthalpy”);  // set property identifier

//calculate properties
Imo.CalcProp(properties, phaseQualifiers, calculationType);
```

7. Step 3: Get Results

```
CapeArrayDouble val[2];                  //double array created.
CapeString basisQualifier;

Strcpy(basisQualifier, ”mole”);          // set basis qualifier

Imo.GetProp(”Enthalpy”, phaseQualifiers, UNDEFINED, calculationType,
basisQualifier, val); //get property enthalpy  in the liquid phase
```

3.8.2 Example 2: Calling a flash and then calculating a viscosity:

```
//Create Material Object
Set Imo     = MaterialTemplate.CreateMaterialObject();
```

8. Step 1: Set Values

```
CapeString phaseQualifiers[1];

T[0] = 373;                          // initialize temperature
P[0] = 101325;                       // initialize pressure

F[0] = 0.1;                          // initialize overall composition
F[1] = 0.7;                          // initialize overall composition
F[2] = 0.2;                          // initialize overall composition

Strcpy(phaseQualifiers[0], ”Overall”);    // set phase qualifier

// set temperature, pressure and composition on the material object
Imo.SetProp(”temperature”, phaseQualifier, UNDEFINED, UNDEFINED, UNDEFINED, T);
Imo.SetProp(”pressure”, phaseQualifier, UNDEFINED, UNDEFINED, UNDEFINED, P);
Imo.SetProp(”fraction”, phaseQualifiers, UNDEFINED, UNDEFINED, ”mole”, F);
```

9. Step 2 : Calculate Flash

```
CapeString flashTypeQualifier;

// set flash type qualifier
Strcpy(flashTypeQualifier, ”TP”);

// call equilibrium server, no addtitional calculation of further
// properties (UNDEFINED)

Imo.CalcEquilibrium(FlashType, UNDEFINED);
```

10. Step 3: Calculate Viscosity

```
CapeString calculationType;

// set calculation type
Strcpy(calculationType, ”Mixture”);

// set phase qualifier
Strcpy(phaseQualifier, ”Liquid”);

//calculate viscosity
Imo.CalcProp("viscosity", phaseQualifier, calculationType);
```

11. Step 4: Get Results

```
CapeDoubleArray val;                              //double created.

//get property viscosity for the liquid phase from the
//Material Object.
Imo.GetProp(”viscosity”, phaseQualifier, UNDEFINED, calculationType,
basisQualifier, val);
```

3.9 Interface Descriptions

3.9.1 ICapeThermoMaterialTemplate

Interface Name	ICapeThermoMaterialTemplate
Method Name	CreateMaterialObject
Returns	CapeInterface

Description

Allows a Material Object to be created from the Material Template interface.

Arguments

Name	Type	Description
[out, retval] *ICapeInterface	CapeInterface	The created and initialized Material Object.

Errors

Notes

The returned Material Object is initialized to have the same compounds and phases as the Material Template that it was created from.

Interface Name	ICapeThermoMaterialTemplate
Method Name	SetProp
Returns	-

Description

Allows properties and values to be set on the Material Template.

Arguments

Name	Type	Description
[in] property	CapeString	The custom property to set.
[in] values	CapeArrayDouble	The actual values of the property.

Errors

3.9.2 ICapeThermoMaterialObject

Interface Name	ICapeThermoMaterialObject
Method Name	ComponentIds
Returns	CapeArrayString

Description

Returns the list of compound IDs of a given Material Object.

Arguments

Name	Type	Description
[out, retval] *compIds	CapeArrayString	Compound IDs

Errors

Notes

Compound IDs are used throughout the communication with the Material Object to identify chemical compounds. Each compound should have a unique ID.

Interface Name	ICapeThermoMaterialObject
Method Name	PhaseIds
Returns	CapeArrayString

Description

It returns the phases existing in the Material Object at that moment.

Arguments

Name	Type	Description
[out, retval] *phaseIds	CapeArrayString	List of phases

Errors

Notes

The list may be empty/UNDEFINED if the Material Object has not been flashed yet. The list can change as a result of flash calculations or SetProp calculations.

For a list of valid phaseIds, see 3.11.1; however the Overall phase and multiphase (i.e. VaporLiquid) identifiers must not be returned by this method. This point is very important because otherwise the communication between a PME and a PropertyPackage may fail, should “Overall” be returned and this phaseId be used for e.g. a CalcProp call. See 5.6.3 for more information.

Interface Name	ICapeThermoMaterialObject
Method Name	GetUniversalConstant
Returns	CapeArrayVariant

Description

Retrieves values of universal constants from the Property Package.

Arguments

Name	Type	Description
[in] props	CapeArrayString	List of universal constants to be retrieved
[out, retval] *propvals	CapeArrayVariant	Values of universal constants

Errors

Notes

See 3.10.2 for a list of authorized universal constant identifiers.

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	GetComponentConstant
Returns	CapeArrayVariant

Description

Retrieve pure compound constants from the Property Package.

Arguments

Name	Type	Description
[in] props	CapeArrayString	List of pure compound constants
[in] compIds	CapeArrayString	List of compound IDs for which constants are to be retrieved. UNDEFINED is to be used when the call applied to all compounds in the Material Object.
[out,retval] *propvals	CapeArrayVariant	Compound Constant values returned from the Property Package for the specified compounds.

Errors

Notes

The list of available identifiers for pure compound constants is given in section 3.10.1.

The return value is a variant containing a one-dimensional array of variants. If P is the number of requested properties and C the number of requested compounds, the array will contain C*P Variants. The C first ones (from position 0 to C-1) will be the values for the first requested property (one variant for each compound). After this first set of values and from position C to 2*C-1, there will be the values of constants for the second requested property, and so on.

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	CalcProp
Returns	-

Description

This method is responsible for doing all property calculations or delegating these calculations to the associated Property Package.

Arguments

Name	Type	Description
[in] props	CapeArrayString	The List of Properties to be calculated.
[in] phases	CapeArrayString	List of phases for which the Properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Compound Property. For partial property, such as fugacity coefficients of compounds in a mixture, use “Mixture” CalcType. For pure compound fugacity coefficients, use “Pure” CalcType.

Errors

Notes

See also sections 3.5, 3.8 and 5.2.

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	GetProp
Returns	CapeArrayDouble

Description

This method is responsible for retrieving the results from calculations from the Material Object.

Arguments

Name	Type	Description
[in] property	CapeString	The Property for which results are requested from the Material Object.
[in] phase	CapeString	The qualified phase for the results.
[in] compIds	CapeArrayString	The qualified compounds for the results. UNDEFINED to specify all compounds in the Material Object. For scalar mixture properties such as liquid enthalpy, this qualifier must not be specified. Use UNDEFINED as place holder.
[in] calcType	CapeString	The qualified type of calculation for the results. (valid Calculation Types: Pure and Mixture)
[in] basis	CapeString	Qualifies the basis of the result (i.e., mass /mole). Use UNDEFINED for default or as place holder for property for which basis does not apply (see also 3.3.1).
[out, retval] *results	CapeArrayDouble	Results vector containing property values in SI units arranged by the defined qualifiers.

Errors

Notes

See 5.1 for a more detailed explanation of the arguments.

Interface Name	ICapeThermoMaterialObject
Method Name	SetProp
Returns	-

Description

This method is responsible for setting the values for properties of the Material Object.

Arguments

Name	Type	Description
[in] property	CapeString	The property for which the values need to be set.
[in] phase	CapeString	Phase for which the property is to be set.
[in] compIds	CapeArrayString	Compounds for which values are to be set. UNDEFINED to specify all compounds in the Material Object. For scalar mixture properties such as liquid enthalpy, this qualifier should not be specified. Use UNDEFINED as place holder.
[in] calcType	CapeString	The calculation type. (valid Calculation Types: Pure and Mixture)
[in] basis	CapeString	Qualifies the basis (mole / mass). See also 3.3.2.
[in] values	CapeArrayDouble	Values to set for the property.

Errors

Notes

See 5.1 for a more detailed explanation of the arguments.

Interface Name	ICapeThermoMaterialObject
Method Name	CalcEquilibrium
Returns	-

Description

This method is responsible for calculating a flash or delegating flash calculations to the associated Property Package or Equilibrium Server.

Arguments

Name	Type	Description
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeArrayString	Properties to be calculated at equilibrium. UNDEFINED for no properties. If a list, then the property values should be set for each phase present at equilibrium (not including the overall phase).

Errors

Notes

The CalcEquilibrium method must set on the Material Object the amounts (phaseFraction) and compositions (fraction) for all phases present at equilibrium, as well as the temperature and pressure for the overall mixture, if not set as part of the calculation specifications. The CalcEquilibrium method must not set on the Material Object any other value - in particular it must not set any values for phases that do not exist. See 5.2.1 for more information.

The available list of flashes is given in section 5.6.1.

It is advised not to combine a flash calculation with a property calculation. Although by the returned error one cannot see which has failed, plus the additional arguments to CalcProp (such as calculation type) cannot be specified. Advice is to perform a CalcEquilibrium, get the phaseIDs and perform a CalcProp on the existing phases.

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	SetIndependentVar
Returns	-

Description

Sets the independent variable for a given Material Object. This method is deprecated.

Arguments

Name	Type	Description
[in] indVars	CapeArrayString	Independent variables to be set
[in] values	CapeArrayDouble	Values of independent variables.

Errors

Notes

This method should not be used, see 3.3.2.

Interface Name	ICapeThermoMaterialObject
Method Name	GetIndependentVar
Returns	CapeArrayDouble

Description

Returns the independent variables of a Material Object. This method is deprecated.

Arguments

Name	Type	Description
[in] indVars	CapeArrayString	Independent variables to be set
[out, retval] *values	CapeArrayDouble	Values of independent variables.

Errors

Notes

This method should not be used, see 3.3.2.

Interface Name	ICapeThermoMaterialObject
Method Name	PropCheck
Returns	CapeArrayBoolean

Description

Checks to see if a list of given properties can be calculated.

Arguments

Name	Type	Description
[in] props	CapeArrayString	Properties to check.
[out, retval] *valid	CapeArrayBoolean	Returns Boolean List associated to list of properties to be checked.

Errors

Notes

As it was unclear from the original specification what PropCheck should exactly be checking, and as the argument list does not include a phase specification, implementations vary. It is generally expected that PropCheck at least verifies that the Property is available for calculation in the Material Object. However, this can also be verified with PropList. It is advised not to use PropCheck.

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	AvailableProps
Returns	CapeArrayString

Description

Gets a list of properties that have been calculated.

Arguments

Name	**Type**	**Description**
[out,retval] *props	CapeArrayString	Properties for which results are available.

Errors

Notes

This function should return a list of properties that are available on any of the phases; if a property is present in at least one phase, it should be included in the list. A Material Object may choose not to support this functionality. In this case it should not return an empty list, but rather return error ECapeNotImpl.

Interface Name	ICapeThermoMaterialObject
Method Name	RemoveResults
Returns	-

Description

Remove all or specified property results in the Material Object.

Arguments

Name	Type	Description
[in] props	CapeArrayString	Properties to be removed. UNDEFINED to remove all properties.

Errors

Notes

Interface Name	ICapeThermoMaterialObject
Method Name	CreateMaterialObject
Returns	CapeInterface

Description

Create a Material Object from the parent Material Template of the current Material Object.

Arguments

Name	Type	Description
[out, retval] *MaterialObject	CapeInterface	The created and initialized Material Object.

Errors

Notes

This method has the same purpose as the CreateMaterialObject method on the parent Material Template.

The created Material Object will have the same configuration (e.g. compound and phase definitions) as the Material Object that created it. As opposed to Duplicate, actual property values may not be available; one must not assume presence of any valid values on the created Material Object.

Interface Name	ICapeThermoMaterialObject
Method Name	Duplicate
Returns	CapeInterface

Description

Creates a duplicate of the current Material Object.

Arguments

Name	Type	Description
[out, retval] *clone	CapeInterface	The duplicated Material Object.

Errors

Notes

This method does the same as CreateMaterialObject, but in addition should copy all values stored on the original Material Object to the duplicated Material Object, including existing phases. If the Material Object supports the petroleum fractions interface, and if the Material Object contains petroleum properties, the petroleum properties are also to be copied to the duplicated Material Object.

Interface Name	ICapeThermoMaterialObject
Method Name	ValidityCheck
Returns	CapeArrayThermoReliability

Description

Checks the validity of the calculation. This method is deprecated.

Arguments

Name	Type	Description
[in] Props	CapeArrayString	The properties for which reliability is checked.
[out, retval] *relist	CapeArrayThermoReliability	Returns the reliability scale of the calculation.

Errors

Notes

The ValidityCheck method must not be used, since the ICapeThermoReliability interface is not yet defined.

Interface Name	ICapeThermoMaterialObject
Method Name	GetPropList
Returns	CapeArrayString

Description

Returns list of properties that can be calculated by the Material Object.

Arguments

Name	Type	Description
[out, retval] *props	CapeArrayString	List of all supported properties of the Material Object.

Notes

GetPropList should return identifiers for the non-constant properties calculated by CalcProp. Standard identifiers are listed in 3.10.1. Other non-standard properties that are supported by the Material Object can also be returned. GetPropList must not return identifiers for compound constant properties returned by GetComponentConstant.

The properties Temperature, Pressure, Fraction, Flow, phaseFraction, totalFlow cannot be returned by GetPropList, since all components must support them. Although the property identifier of derivative properties is formed from the identifier of another property, the GetPropList method will return the identifiers of all supported derivative and non-derivative properties. For instance, a Material Object could return the following list:

enthalpy, enthalpy.Dtemperature, entropy, entropy.Dpressure

The Material Object may or may not delegate this call to a Property Package.

Interface Name	ICapeThermoMaterialObject
Method Name	GetNumComponents
Returns	CapeLong

Description

Returns number of chemical compounds in Material Object.

Arguments

Name	Type	Description
[out, retval] *num	CapeLong	Number of compounds in the Material Object.

Errors

Notes

This method returns the current number of compounds in the Material Object. The number may vary between calls to the method. The number corresponds to the length of the list returned by ComponentIds.

The Material Object may or may not delegate this call to a Property Package.

3.9.3 ICapeThermoSystem

Interface Name	ICapeThermoSystem
Method Name	GetPropertyPackages
Returns	CapeArrayString

Description

Returns list of Property Package names supported by the Thermo System.

Arguments

Name	Type	Description
[out, retval] *propertyPackageList	CapeArrayString	The returned set of supported Property Packages.

Errors

Notes

Interface Name	ICapeThermoSystem
Method Name	ResolvePropertyPackage
Returns	CapeInterface

Description

Resolves referenced Property Package to a Property Package interface.

Arguments

Name	Type	Description
[in] propertyPackage	CapeString	The Property Package to be resolved.
[out, retval] *propPackObject	CapeInterface	The Property Package Interface.

Errors

Notes

3.9.4 ICapeThermoPropertyPackage

Interface Name	ICapeThermoPropertyPackage
Method Name	GetComponentList
Returns	-

Description

Returns the list of compounds of a given Property Package.

Arguments

Name	Type	Description
[in,out] *compsIds	CapeArrayString	List of compound IDs
[in,out] *formulae	CapeArrayString	List of compound formulae
[in,out] *name	CapeArrayString	List of compound names.
[in,out] *boilTemps	CapeArrayDouble	List of boiling point temperatures.
[in,out] *molwt	CapeArrayDouble	List of molecular weights.
[in,out] *casno	CapeArrayString	List of CAS numbers .

Notes

Compound identification could be necessary if the PME has internal knowledge of chemical compounds, or in case of use of multiple Property Packages. In order to identify the compounds of a Property Package, the PME will use the 'casno' argument instead of the compIds. The reason is that different PMEs may give different names to the same chemical compounds, whereas CAS Numbers are universal. Therefore, it is recommended to provide a value for the casno argument wherever available.

See 5.5 for more information.

The same information can also be extracted using the ICapeThermoPropertyPackage GetComponentConstant method, using the casRegistryNumber property identifier.

Interface Name	ICapeThermoPropertyPackage
Method Name	GetUniversalConstant
Returns	CapeArrayVariant

Description

Returns the values of the Universal Constants.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object.
[in] props	CapeArrayString	List of requested Universal Constants;
[out,retval] *propvals	CapeArrayVariant	Values of Universal Constants.

Errors

Notes

See 3.10.2 for a list of authorized universal constant identifiers.

Interface Name	ICapeThermoPropertyPackage
Method Name	GetComponentConstant
Returns	CapeArrayVariant

Description

Returns the values of the Constant properties of the compounds contained in the passed Material Object.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object.
[in] props	CapeArrayString	The list of properties.
[out,retval] *propvals	CapeArrayVariant	Compound Constant values.

Errors

Notes

The return value is a variant containing a one-dimensional array of variants. If P is the number of requested properties and C the number requested compounds, the array will contain C*P variants. The C first ones (from position 0 to C-1) will be the values for the first requested property (one variant for each compound). After this first set of values and from position C to 2*C-1, there will be the values of constants for the second requested property, and so on.

Interface Name	ICapeThermoPropertyPackage
Method Name	CalcProp
Returns	-

Description

This method is responsible for doing property calculations.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the Calculation.
[in] props	CapeArrayString	The List of Properties to be calculated.
[in] phases	CapeArrayString	List of phases for which the properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Compound Property. For partial property, such as fugacity coefficients of compounds in a mixture, use “Mixture” CalcType. For pure compound fugacity coefficients, use “Pure” CalcType.

Notes

See also section 5.2.

Interface Name	ICapeThermoPropertyPackage
Method Name	CalcEquilibrium
Returns	-

Description

Method responsible for calculating/delegating phase equilibria.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object
[in] flashType	CapeString	Flash calculation type.
[in] Props	CapeArrayString	Properties to be calculated at equilibrium. UNDEFINED for no properties. If a list, then the property values should be set for each phase present at equilibrium. (not including the overall phase).

Errors

Notes

On the Material Object the CalcEquilibrium method must set the amounts (phaseFraction) and compositions (fraction) for all phases present at equilibrium, as well as the temperature and pressure for the overall mixture, if these are not set as part of the calculation specifications. The CalcEquilibrium method must not set on the Material Object any other value - in particular it must not set any values for phases that do not exist. See 5.2.1 for more information.

The available list of flashes is given in section 5.6.1.

When calling this method, it is advised not to combine a flash calculation with a property calculation. Through the returned error one cannot see which has failed, plus the additional arguments available in a CalcProp call (such as calculation type) cannot be specified in a CalcEquilibrium call. Advice is to perform a CalcEquilibrium, get the phaseIDs and perform a CalcProp for the existing phases.

Interface Name	ICapeThermoPropertyPackage
Method Name	PropCheck
Returns	CapeArrayBoolean

Description

Check to see if properties can be calculated.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] Props	CapeArrayString	List of Properties to check.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for each property.

Errors

Notes

As it was unclear from the original specification what PropCheck should exactly be checking, and as the argument list does not include a phase specification, implementations vary. It is generally expected that PropCheck at least verifies that the Property is available for calculation in the property Package. However, this can also be verified with PropList. It is advised not to use PropCheck.

Interface Name	ICapeThermoPropertyPackage
Method Name	ValidityCheck
Returns	CapeArrayThermoReliability

Description

Checks the validity of the calculation. This method is deprecated.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] Props	CapeArrayString	The list of properties to check.
[out, retval] *relist	CapeArrayThermoReliability	The properties for which reliability is checked.

Errors

Notes

The ValidityCheck method must not be used, since the ICapeThermoReliability interface is not yet defined.

Interface Name	ICapeThermoPropertyPackage
Method Name	GetPropList
Returns	CapeArrayString

Description

Returns list of properties supported by the Property Package.

Arguments

Name	Type	Description
[out, retval] *props	CapeArrayString	List of all supported Properties.

Errors

Notes

GetPropList should return identifiers for the non-constant properties calculated by CalcProp. Standard identifiers are listed in 3.10.1. Other non-standard properties that are supported by the Property Package can also be returned. GetPropList must not return identifiers for compound constant properties returned by GetComponentConstant.

The properties temperature, pressure, fraction, flow, phaseFraction, totalFlow cannot be returned by GetPropList, since all thermodynamic software components must support them. Although the property identifier of derivative properties is formed from the identifier of another property, the GetPropList method must return the identifiers of all supported derivative and non-derivative properties. For instance, a Property Package could return the following list:

enthalpy, enthalpy.Dtemperature, entropy, entropy.Dpressure.

Interface Name	ICapeThermoPropertyPackage
Method Name	GetPhaseList
Returns	CapeArrayString

Description

Provides the list of the supported phases. When supported for one or more property calculations, the Overall phase and multiphase identifiers must be returned by this method.

Arguments

Name	Type	Description
[out, retval] *phases	CapeArrayString	The list of phases supported by the Property Package.

Notes

3.9.5 ICapeThermoCalculationRoutine

Interface Name	ICapeThermoCalculationRoutine
Method Name	CalcProp
Returns	-

Description

This method is responsible for doing all calculations on behalf of the Calculation Routine component.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object of the calculation.
[in] props	CapeArrayString	The list of properties to be calculated.
[in] phases	CapeArrayString	List of phases for which the properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Compound Property. For partial property, such as fugacity coefficients of compounds in a mixture, use “Mixture” CalcType. For pure compound fugacity coefficients, use “Pure” CalcType.

Errors

Notes

See also, section 5.2.

Interface Name	ICapeThermoCalculationRoutine
Method Name	PropCheck
Returns	CapeArrayBoolean

Description

Checks to see if a given property can be calculated.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] Props	CapeArrayString	List of properties to check.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for each property.

Errors

Notes

As it was unclear from the original specification what PropCheck should exactly be checking, and as the argument list does not include a phase specification, implementations vary. It is generally expected that PropCheck at least verifies that the Property is available for calculation in the Calculation Routine. However, this can also be verified with PropList. It is advised not to use PropCheck.

Interface Name	ICapeThermoCalculationRoutine
Method Name	PropList
Returns	-

Description

Returns the set of Properties, Phases, and Calculation Types that are supported by a given Calculation Routine.

Arguments

Name	Type	Description
[in,out] *props	CapeArrayString	List of supported properties.
[in,out] *phases	CapeArrayString	List of supported phases.
[in,out] *calcType	CapeArrayString	List of supported calculation types. (Pure & Mixture)

Notes

The property list should contain all properties that can be calculated by the component. If the component is restricted to certain phases, these phases should be in the phase list. If UNDEFINED is returned, it will be assumed that the supported properties can be calculated for any phase.

Interface Name	ICapeThermoCalculationRoutine
Method Name	ValidityCheck
Returns	CapeArrayThermoReliability

Description

Checks the validity of the calculation. The method is deprecated.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] props	CapeArrayString	The list of properties to check.
[out, retval] *rellist	CapeArrayThermoReliability	The properties for which reliability is checked.

Notes

The ValidityCheck method must not be used, since the ICapeThermoReliability interface is not yet defined.

3.9.6 ICapeThermoEquilibriumServer

Interface Name	ICapeThermoEquilibriumServer
Method Name	CalcEquilibrium
Returns	-

Description

Method responsible for calculating phase equilibria.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	Material Object of the calculation
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeArrayString	Properties to be calculated at equilibrium. UNDEFINED for no properties. If a list, then the property values should be set for each phase present at equilibrium. (not including the overall phase).

Notes

The CalcEquilibrium method must set on the Material Object the amounts (phaseFraction) and compositions (fraction) for all phases present at equilibrium, as well as the temperature and pressure for the overall mixture, if not set as part of the calculation specifications. The CalcEquilibrium method must not set on the Material Object any other value - in particular it must not set any values for phases that do not exist. See 5.2.1 for more information.

The available list of flashes is given in section 5.6.1.

It is advised not to combine a flash calculation with a property calculation. By the returned error one cannot see which has failed, plus the additional arguments to CalcProp (such as calculation type) cannot be specified. Advice is to perform a CalcEquilibrium, get the phaseIDs and perform a CalcProp for those phases.

Interface Name	ICapeThermoEquilibriumServer
Method Name	PropCheck
Returns	CapeArrayBoolean

Description

Checks to see if a given type of flash calculations can be performed and whether the properties can be calculated after the flash calculation.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] flashType	CapeString	Type of flash calculation to check
[in] Props	CapeArrayString	List of Properties to check. UNDEFINED for none.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for flash and property. First element is reserved for flashType.

Notes

As it was unclear from the original specification what PropCheck should exactly be checking, and as the argument list does not include a phase specification, implementations vary. It is generally expected that PropCheck at least verifies that the Property is available for calculation in the Material Object. However, this can also be verified with PropList. It is advised not to use PropCheck.

Interface Name	ICapeThermoEquilibriumServer
Method Name	ValidityCheck
Returns	CapeArrayThermoReliability

Description

Checks the reliability of the calculation.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] props	CapeArrayString	The list of properties to check. UNDEFINED for none.
[out, retval] *rellist	CapeArrayThermo Reliability	The properties for which reliability is checked. First element reserved for reliability of flash calculations.

Notes

This method should not be used since the ICapeReliability interface is not yet defined.

Interface Name	ICapeThermoEquilibriumServer
Method Name	PropList
Returns	-

Description

Returns the flash types, properties, phases, and calculation types that are supported by a given Equilibrium Server Routine.

Arguments

Name	Type	Description
[in,out] *flashType	CapeArrayString	Type of flash calculations supported.
[in,out] *props	CapeArrayString	List of supported properties.
[in,out] *phases	CapeArrayString	List of supported phases.
[in,out] *calcType	CapeArrayString	List of supported calculation types. (Pure & Mixture)

Errors

Notes

3.10 CAPE-OPEN Properties List

3.10.1 Constant Properties Identifiers

See section 5 for more explanation on the table and section 2 for naming conventions. Units of measure can be found in the section 4.

To identify a pure compound there are some attributes, which are not really 'properties', but are nevertheless needed:

Identifier	**Character string for identification e.g. in a flowsheet or PPS**
iupacName	Complete IUPAC Name
casRegistryNumber	Chemical Abstract Sequencing Number
chemicalFormula	Chemical formula (nomenclature according to Hill, Hill, J. Am. Chem. Soc. 22(8), 478-494 (1900).)
structureFormula	Chemical structure formula

The ID of a compound should be the same in the whole flowsheet. There is a need in every flowsheet calculation for a global compound list. If one uses different Property Packages, it is very probable that there are different names used for the same compound. So a translation list may help in which, for each name of the global compound list, there are the names for the correspondent compounds in the Property Packages being used.

Identifiers	Meaning	SI Units
molecularWeight	Relative molecular mass	
criticalTemperature	Critical Temperature	K
criticalPressure	Critical Pressure	Pa
criticalVolume	Critical Volume	m^3/mol
criticalCompressibilityFactor	Critical Compressibility Factor	
criticalDensity	Critical Density	mol/m^3
acentricFactor	Pitzer Acentric Factor	
dipoleMoment	Dipole Moment	Cm
parachor	Parachor	m^3 $kg^{0.25}$/($s^{0.5}$ mol)
gyrationRadius	Radius of Gyration	m
associationParameter	Association-Parameter (Hayden-O'Connell)	
diffusionVolume	Diffusion volume	m^3
vanderwaalsVolume	Van der Waals Volume	m^3/mol
vanderwaalsArea	Van der Waals Area	m^2/mol
energyLennardJones	Lennard-Jones energy parameter divided by Boltzmann constant	K
lengthLennardJones	Lennard-Jones length parameter	m

normalBoilingPoint	Temperature at boiling point (1.01325 bar)	K
heatOfVaporizationAtNormalBoilingPoint	Heat of Vaporization at boiling point (1.01325 bar)	J/mol
normalFreezingPoint	Temperature of normal melting point (1.01325 bar)	K
heatOfFusionAtNormalFreezingPoint	Heat of Melting at melting point (1.01325 bar)	J/mol
liquidDensityAt25C	Liquid Density at 25 C	mol/m^3
liquidVolumeAt25C	Liquid Volume at 25 C	m^3/mol
idealGasGibbsFreeEnergyOfFormationAt25C		J/mol
idealGasEnthalpyOfFormationAt25C		J/mol
standardFormationEnthalpySolid	Standard Formation Enthalpy of Solid	J/mol
standardFormationEnthalpyLiquid	Standard Formation Enthalpy of Liquid	J/mol
standardFormationEnthalpyGas	Standard Formation Enthalpy of Gas	J/mol
standardFreeFormationEnthalpySolid	Standard Free Formation Enthalpy of Solid	J/mol
standardFreeFormationEnthalpyLiquid	Standard Formation Enthalpy of Liquid	J/mol
standardFreeFormationEnthalpyGas	Standard Formation Enthalpy of Gas	J/mol
standardEntropySolid	Standard Entropy of Solid	J/mol
standardEntropyLiquid	Standard Entropy of Liquid	J/mol
standardEntropyGas	Standard Entropy of Gas	J/mol
triplePointTemperature	Triple Point Temperature	K
triplePointPressure	Triple Point Pressure	Pa
BornRadius		m
charge		
StandardEnthalpyAqueousDilution	Standard aqueous infinite dilution enthalpy	J/mol
StandardGibbsAqueousDilution	Standard aqueous infinite Gibbs energy	J/mol

Standard is at 25 C and 1.01325 bar (= 1 atm).

3.10.2 Universal constant properties

standardAccelerationOfGravity	9.806 65 m/s^2
avogadroConstant	6.022 141 99(47) 10^{23} mol^{-1}
boltzmannConstant	1.380 6503(24) 10^{-23} J K^{-1}
molarGasConstant	8.314 472(15) J mol^{-1} K^{-1}

Note: Only the units of measure and the identifiers of the universal constants are specified in the standard, not the values.

3.10.3 Non-constant Properties (or Model Dependent Properties)

See section 6 for more explanation of the revised table and section 2 for general naming conventions. Units of measure can be found in the section. Mole based units have been listed in this table. For any property in this table that mol in the unit of measure in this table, the unit of measure is affected by the basis argument to SetProp and GetProp. The mol in this table would be replaced by kg for mass basis (with the exception of properties moles and mass, that are basis independent).

Identifiers	Meaning	SI Units
vaporPressure	Vapor Pressure. Only for Pure calcType	Pa
sublimationPressure	Sublimation Pressure	Pa
meltingPressure	Melting Pressure	Pa
glassTransitionTemperature	Glass Transition Temperature	K
glassTransitionPressure	Glass Transition Pressure	Pa
solidSolidPhaseTransitionTemperature	SolidSolidPhaseTransitionPressure	Pa
virialCoefficient	Second Virial Coefficient	m^3/mol
surfaceTension	Surface Tension	N/m
Expansivity	coefficient of linear expansion (Expansivity) $\frac{1}{L}\frac{\partial L}{\partial T}\Big\vert$	1/K
Compressibility	$\frac{1}{V}\frac{\partial V}{\partial P}\Big\vert_T$	1/Pa
compressibilityFactor	Compressibility Factor $Z = \frac{PV}{RT}$	
diffusionCoefficient***	Binary diffusion coefficient for all species in mixture relative to all other species.	m^2/s
jouleThomsonCoefficient	$\frac{\partial T}{\partial P}\Big\vert_H$	K/Pa
heatOfVaporization	**	J/mol
heatOfSublimation	**	J/mol
heatOfFusion	**	J/mol
heatOfSolidSolidPhaseTransition	**	J/mol
volumeChangeUponVaporization	**	m^3/mol
volumeChangeUponSublimation	**	m^3/mol
volumeChangeUponMelting	**	m^3/mol
volumeChangeUponSolidSolidPhaseTr	**	m^3/mol

ansition		
heatCapacity	Heat Capacity (Cp)**	J/(mol K)
heatCapacityCv	Heat Capacity (Cv)**	J/(mol K)
idealGasHeatCapacity	Heat Capacity of ideal Gas**	J/(mol K)
idealGasEnthalpy	Enthalpy of ideal Gas*	J/mol
excessEnthalpy	Excess enthalpy*	J/mol
excessEnergy	Excess energy*	J/mol
excessGibbsFreeEnergy	Excess Gibbs Free Energy*	J/mol
excessHelmholtzFreeEnergy	Excess Helmholtz Free Energy*	J/mol
excessEntropy	Excess entropy*	J/(mol K)
excessVolume	Excess volume*	m^3/mol
partialMolarEnthalpy		J/mol
partialMolarEnergy	Partial molar internal energy	J/mol
partialGibbsFreeEnergy	Partial molar Gibbs energy	J/mol
partialHelmholtzFreeEnergy	Partial molar Helmholtz energy	J/mol
partialMolarVolume		m^3/mol
viscosity	Viscosity	Pa s
thermalConductivity	Thermal Conductivity	W/(m K)
fugacity	Fugacity	Pa
fugacityCoefficient	Fugacity Coefficient	
activity	Activity	
activityCoefficient	Activity Coefficient	
bubblePointPressure		Pa
bubblePointTemperature		K
dewPointPressure		Pa
dewPointTemperature		K
kvalues	Ratio of fugacity coefficients for a pair of phases defined as follows: $K_i = \phi_{i2} / \phi_{i1}$ where ϕ_{i1} is the fugacity coefficient of compound i in phase 1 and ϕ_{i2} is the fugacity coefficient in phase 2. The VaporLiquid kvalues are defined as $K_{VL} = \phi_L / \phi_V$	
logFugacityCoefficient	Logarithm of fugacity coefficients	
logkvalues	Logarithm of kvalues	
temperature		K
pressure		Pa
volume	Volume*	m^3/mol
density	Density **	mol/m^3

enthalpy	Enthalpy* (may or may not include heat of formation)	J/mol
enthalpyNF	Enthalpy which is guaranteed not to include the heat of formation*	J/mol
enthalpyF	Enthalpy which is guaranteed to include the heat of formation*	J/mol
Entropy	Entropy* (may or may not include entropy of formation)	J/(mol K)
EntropyNF	Entropy which is guaranteed not to include the entropy of formation*	J/(mol K)
EntropyF	Entropy which is guaranteed to include the entropy of formation*	J/(mol K)
energy	Internal energy*	J/mol
gibbsFreeEnergy	Gibbs Free Energy*	J/mol
helmholtzFreeEnergy	Helmholtz Free Energy*	J/mol
moles	Number of moles of a given amount of matter	Mol
mass	Total mass of a given amount of matter	kg
flow	List of the partial molar (or mass) flows of each compound within a given phase (or the whole mixture)**	mol/s
fraction	List of the partial molar (or mass) fractions of each compound within a given phase (or the whole mixture)	
phaseFraction	The fraction of the fluid that is in the specified phase.	
totalFlow	Matter flow of a phase or the whole mixture**	mol/s
molecularWeight	It is recommended to be used only with CalcType=”mixture”. For pure, GetComponentConstant is preferred. It is up to the package to calculate it by whatever means it chooses	
boilingPointTemperature	Only supported for “pure” CalcType	K
dielectricConstant	The ratio of the capacity of a condenser with a particular substance as dielectric to the capacity of the same condenser with a vacuum for dielectric	
cpAqueousInfiniteDilution	Heat capacity of a solute in an infinitely dilute aqueous solution	J/(mol K)
DissociationConstant	Chemical equilibrium cinstant corresponding to a dissociation reaction	

OsmoticCoefficient	A measure of water activities, defined as, $\phi = - n_W \ln (x_W f_W)/(n_S \sum \nu_i)$ where, n_W is the moles of water; n_S is the moles of solute; x_W is the mole fraction of water; f_W is the symmetric activity coefficient of water; ν_I is the stoichiometric coefficient of compound i.	
PH		
POH		
MeanActivityCoefficient	The geometrical mean of the activity coefficients of the ions in an electrolyte solution	
SolubilityIndex		
SolubilityProduct		

Notes: * per mole, or kg, or total depending on basis.

**** per mole, or kg depending on basis**.

***** value has the dimension of a matrix**

Derivatives:

Derivatives are built from the property identifier: a point with a D meaning "Derivative" and the name for the independent variable. The only independent variables that may be specified are temperature, pressure, and mole numbers or mole fractions, as shown in the table below.

Derivative identifier	meaning	units
property.Dtemperature	derivative of property with respect to temperature with pressure and composition fixed	[property]/K
property.Dpressure	derivative of property with respect to pressure with temperature and composition fixed	[property]/Pa
property.Dmoles	derivatives of property with respect to mole number keeping pressure and temperature and other mole numbers fixed for a mixture containing a total of one mole of material. For some property H the ith element of derivative is $\mathrm{H.Dmoles_i} = \bar{h}_i = \left(\frac{\partial H}{\partial n_i}\right)_{p,T,n_{j\neq i}}$	[property]/mol

Property.DmolFraction	derivatives of property with respect to mole fraction, keeping pressure and temperature and other mole fractions fixed. The mole fractions are therefore treated as independent variables. These derivatives are a mathematical construction and do not necessarily have a physical meaning. The derivatives depend on the specific implementation of the property in the property package and may therefore not be unique. So mole fraction derivatives from different Property Packages can't be expected to coincide in general. However they should coincide as directional derivatives with directions d that lie in the plane $\sum_{i=1}^{N} x_i = 1$, i.e. $\sum_{i=1}^{N} d_i = 0$. The directional derivative is the scalar product of the derivative ("gradient") and the direction d: $\nabla_x H(\vec{x},T,P)\cdot\vec{d} = \sum_{i=1}^{N} \frac{\partial H(x_1,\ldots,x_N,T,P)}{\partial x_i}\cdot d_i$ For some property H, the i[th] element of the derivative is $\text{H.DmolFraction} = \left(\frac{\partial H}{\partial x_i}\right)_{x_{j\neq i},T,P} = \frac{\partial H(x_1,\ldots,x_N,T,P)}{\partial x_i}$	[property]

BASIS AND UNITS:

The units for a derivative property depend on the units of the property itself, the basis specified and the type of derivative, as shown in the table above. For example, enthalpy.Dtemperature on a molar basis has units of J/(K mol) and on a mass basis it has units of J/(K kg).

For mole number derivatives the combination of basis and property type leads to a number of possible combinations. The table below gives examples of all the possibilities.

Property type	Basis	Example of property	Units of .Dmoles derivative
Intensive	UNDEFINED	logFugacityCoefficient	1/mol
	mole	density	(mol/m3)/mol
	mass	density	(kg/m3)/mol
Extensive	mole	enthalpy	J/mol

For extensive properties only the molar basis is allowed. This should be interpreted as the mole number derivative of the extensive property for one mole of substance and it corresponds to a partial molar property.

NUMBER OF VALUES RETURNED AND ORDER:

The following rules apply:

- Dtemperature and Dpressure derivatives return the same number of values and in the same order as the corresponding property. For example, enthalpy.Dtemperature will return a single value, whereas fugacityCoefficient.Dtemperature will return a vector of values.

- Dmoles derivatives of scalar properties return a vector of values with the same number of elements as there are compounds in the mixture. For example, enthalpy.Dmoles with a basis of Mole will return a vector of the partial molar enthalpies containing as many values as there are compounds.

- Dmoles derivatives of vector-valued mixture properties are, conceptually, a sequence of vectors. The first vector returns the derivatives of all properties with respect to the mole number of the first compound. The second vector returns the derivative with respect to the mole number of the second compound, and so on. The actual representation of these values will be a single sequence that contains all the values of these vectors as a one-dimensional array.

- For example, activity.Dmoles with respect to all compounds will return the following values:

$$\{\overline{a}_{11}, \overline{a}_{21}, \overline{a}_{31} \ldots \overline{a}_{n1}, \overline{a}_{12}, \overline{a}_{22}, \overline{a}_{32} \ldots \overline{a}_{n2}, \overline{a}_{13}, \overline{a}_{23}, \overline{a}_{33} \ldots \overline{a}_{n3}, \cdots, \overline{a}_{1n}, \overline{a}_{2n}, \overline{a}_{3n} \ldots \overline{a}_{nn}, \}$$

- where the derivative of the activity of compound i with respect to the mole number of compound j is

$$\overline{a}_{ij} = \left(\frac{\partial a_i}{\partial n_j} \right)_{p,T,n_{k \neq j}}$$

- In general, Dmoles derivatives of a rank-m quantity are returned as a sequence of values containing the compounds of a rank-m+1 quantity.

- The remarks for Dmoles derivatives in this section also apply to DmoleFraction derivatives.

3.11 CAPE-OPEN Phase List

3.11.1 Phase Details

Permitted phases have been restricted to the following:

Phase	Description
Vapor	Vapor phase
Liquid	Liquid phase
LiquidX	Liquid phase X
Solid	Solid phase
SolidX	Solid phase X
Overall	All phases

Multiple liquid phases can be achieved by using different names for a liquid. All liquid-phase names must start with “Liquid” so that they can be identified as a liquid. Multiple solid phases can be achieved by using different names for a solid. All solid-phase names must start with “Solid” so that they can be identified as a solid. It is advised to use the name “Liquid” for the first liquid phase, and the name “Solid” for the first solid phase. In the above table, X is a place holder for any name, so “LiquidX” could be for example “LiquidWater”, “Liquid2”, …

See section 5.6.2 for more information and section 5.6.3 for information on phase checking.

4. SI Units

We have added the units used in the interoperability demonstration implementations to the CAPE-OPEN Properties List in this document. We suggest referring to the *Bureau International des Poids et Mesures* website (http://www.bipm.fr/enus/3_SI/si.html) for more information.

5. Notes

5.1 Notes on Argument interpretation of Get/Set/CalcProp Standard Methods)

5.1.1 Non-constant properties

If vapor fraction is part of a flash specification (e.g. PVF flash), it must be set on the Material Object for phase “vapor” before performing a flash calculation, using basis “mole”. All other properties that can be part of flash specification must be set on the Overall phase prior to the equilibrium calculation (in addition to overall composition).

5.2 Notes on CalcProp description

5.2.1 CalcProp and CalcEquilibrium

There is no interaction between CalcProp and CalcEquilibrium, so CalcProp should never invoke CalcEquilibrium.

CalcProp is used to calculate properties in the specified phase at the current values of T, P and x; it does not perform phase equilibrium calculations.

CalcEquilibrium is used to calculate state variables from others, such as enthalpy, entropy or phase fraction.

5.2.2 Multiple calculations

If a client uses multiple properties in a call and one of them fails, then the whole call should be considered to have failed. Therefore, no value should be written back to the material object by the Property Package until it is known that the whole request can be satisfied. For this reason, to simplify error handling or debugging, it is recommended that clients only request one property at a time to make error handling simpler. For performance however, calculation of as many properties at the same time is recommended. Especially if the properties are related (e.g. all resulting from the evaluation of an equation of state), combined property calculations will perform much better than individual property calculations.

5.2.3 Side-effects during calculation

It is important to note that Property Packages are NOT allowed to calculate and (more important) to store the values of properties that have not been specifically requested.

5.3 Notes on Constant Properties Identifiers

An important role of the Compound Constants is to identify the compounds supported by a Property Package. The ICapeThermoPropertyPackage GetComponentList method was designed for this purpose.

5.3.1 Compounds supported by a Property Package

Use GetComponentList for a list of the compounds supported by a Physical Properties Package. It is a specific entity tailored to a specific application, rather than a general Physical Properties System.

5.4 Notes on GetComponentConstant

Equivalences between GetComponentList arguments and Compound Constant properties:

GetComponentList Arguments	Compound Constant Property Idenfifier	Comments
Casno	casRegistryNumber	
compIds	--	This string has to be used in all the arguments of the materialObject and Property Package methods which are named compIds.
Formulae	chemicalFormula	
Name	iupacName	
BoilTemps	normalBoilingPoint	
Molwt	molecularWeight	

The problem was that “casRegistryNumber” and other properties have values, which are not numbers but strings, whereas the specification states that getComponentConstants returns a list of numbers. As stated in the section 2.14.1 of the 1999 Thermodynamic and Physical Properties Specification, the following constant properties are not supported by the current specification of GetComponentConstant:

- iupacName complete IUPAC Name
- casRegistryNumber Chemical Abstract Sequencing Number
- chemicalFormula Chemical formula (nomenclature according to Hill)
- structureFormula Chemical structure formula

For the same reasons, GetUniversalConstant should also return a CapeArrayVariant.

5.5 Description of Compound Constants

5.5.1 CasRegistryNumber

The value of this constant is a variable-length character string that contains a sequence of 3 numbers separated by hyphens. There must be no leading zeros and no leading spaces. The intention is that it should be possible to compare two CAS numbers with a simple string comparison

CAS numbers and other properties are accessible at

http://webbook.nist.gov/chemistry

Compounds can be accessed directly with

http://webbook.nist.gov/cgi/cbook.cgi?Formula=ch4&NoIon=on&Units=SI

or

http://webbook.nist.gov/cgi/cbook.cgi?Name=water&Units=SI

CAS numbers can be undefined, for example for petroleum fractions, in which case compound comparison has to be done by looking at constant properties.

5.6 Notes on Phases

The list of phases in 2.15.1 of the original specification assumes that 2 liquid fractions are supported, since, for instance, there is a LiquidLiquid phase detail. However, at present there is no way to refer to each one of the liquid phases separately. For this reason, it is advised to use version 1.1 of the Thermodynamic and Physical Properties standard.

5.6.1 Equilibrium calculation types

The following list of flash types is defined. Each flash type may or may not be supported, depending on the component that provides the equilibrium calculations.

The most commonly supported flash types are:

Flash Type	Descriptions
TP	Temperature-Pressure
PH	Pressure-Enthalpy
TH	Temperature-Enthalpy
TVF	Temperature-Vapor Fraction (mole basis)
PVF	Pressure-Vapor Fraction (mole basis)

Other flash types for which support may or may not be present:

Flash Type	Descriptions
PS	Pressure-Entropy
TS	Temperature-Entropy
HS	Enthalpy-Entropy
UV	Energy-Volume
SV	Entropy-Volume
PV	Pressure-Volume

TV	Temperature-Volume
HVF	Enthalpy-Vapor fraction
SVF	Entropy-Vapor fraction

Additional flash types that may or may not be supported use EnthalpyF rather than Enthalpy and EntropyF rather than Entropy:

Flash Type	**Descriptions**
PSF	Pressure-EntropyF
PHF	Pressure-EnthalpyF
THF	Temperature-EnthalpyF
TSF	Temperature-EntropyF
HFSF	EnthalpyF-EntropyF
SFV	EntropyF-Volume
HFVF	EnthalpyF-Vapor fraction
SFVF	EntropyF-Vapor fraction

5.6.2 Notes on two-phase properties

A two-phase property can – in principle – be accessed by concatenation of the single phase names. Therefore, to calculate a vapor-liquid property, the appropriate phase name would be 'VaporLiquid'. Similarly, phases 'VaporSolid', 'LiquidSolid', 'LiquidLiquid1' could be used to identify two-phase combinations.

As multiple liquid phases are an extension to the original version 1.0 standard documentation, it is advised to avoid using it where possible. For example, calcprop("kvalues","LiquidLiquid1") may not be supported. However, in the particular case of "kvalues", if "fugacityCoefficient" is used instead, it works around the problem, since the phases can be calculated independently:

```
mo.SetProp("fraction","liquid", ...,liquid1FractionValue)

mo.calcProp("fugacityCoefficient","liquid")

mo.GetProp("fraction","liquid", ...,liquid1fugacityCoefficient)


mo.SetProp("fraction","liquid", ...,liquid2FractionValue)

mo.calcProp("fugacityCoefficient","liquid")

mo.GetProp("fraction","liquid", ...,liquid2fugacityCoefficient)
```

kvalues = liquid2fugacityCoefficient/liquid1fugacityCoefficient

SurfaceTension is in principle a two-phase property, so phase descriptor VaporLiquid is to be used. However, SurfaceTension can also be considered a liquid property, so Liquid is also acceptable as phase descriptor.

5.6.3 Existence of a phase

As already described, that phaseFraction or totalFlow cannot be used for checking existence of a phase, since for instance in the case of a bubble point both properties would return a 0 value for the vapor phase.

Instead, materialObject.PhaseIDs() must be used, since it returns only the phases existing in the MO at that moment. Note that materialObject.PhaseIDs() does not return the list of phases supported by the Property Package relating to the MO. The latter information is provided by the PropPack.getPhaseList() method.

This approach has the limitation that, currently, the Property Packages don't have any mechanism to change the list phases existing in a material object during the calculation of an equilibrium. Only the COSE could do it. The next version of the standard will add new methods to solve this shortcoming.

It is recommended that a material object sets all phase fractions to UNDEFINED (CapeDoubleUNDEFINED) before the equilibrium calculation (with the exception of phaseFraction of the vapor phase in case of a flash that includes a vapor fraction specification). During the equilibrium calculation a property package should only set the phase fractions and compositions for phases that exist. After the equilibrium calculation a material object can determine the list of phases that are present by inspecting the phase fractions. It should be noted that phase fractions for liquid and vapor phases can be zero at equilibrium for dew- and bubble point calculations.

5.7 Notes on enumeration of Property Packages and Thermo Systems

Thermo Systems and Stand-alone Property Packages on Windows systems are registered as a COM object in the Windows registry, each having their own category ID. The HKEY_CURRENT_USER\Software\Classes key stores the COM classes that have been installed for the current user. The HKEY_LOCAL_MACHINE\Software\Classes key stores system wide COM classes. To modify the HKEY_LOCAL_MACHINE\Software\Classes, one may require administration rights. Generally one does not require administration rights to modify HKEY_CURRENT_USER\Software\Classes. It is therefore up to the installer of the software component to choose whether it is installed for the user only, or for all users. Property Packages and Thermo Systems should be listed from both the HKEY_CURRENT_USER\Software\Classes and HKEY_LOCAL_MACHINE\Software\Classes keys, with the keys in the former taking precedence over the same keys in the latter.

The COM classes root key (HKEY_CLASSES_ROOT) provides exactly that union. It is therefore generally sufficient for a CAPE-OPEN simulation environment to list components that appear in HKEY_CLASSES_ROOT. For more information, see http://msdn2.microsoft.com/en-us/library/ms724475.aspx.

5.8 Notes on recommended error codes

The version 1.0 thermodynamic standard specification does not provide recommended error codes for common error scenarios.

5.9 Implementation of the Persistence Interface

It is expected that simulation environments will allow the possibility to store the current state of a simulation case in order to be able to restore it at any time in the future. Thermodynamic software components such as Property Packages are a part of the simulation case being saved. In addition to storing the information required to recreate the thermodynamic software components (such as CLSID and Property Package name), the information specific to the software component (such as the content of the Property Package) should be stored along with the simulation case. This is important to ensure consistency between simulation sessions in case the content of the Property Package has been changed, but also to allow transfer of the simulation case from one computer system to another.

Persistence may also help to preserve consistency of the simulation case over time. For example, when thermodynamic software component versions change, a stored state of a previous version may allow the thermodynamic software component to function in a manner consistent with the saved version or, alternatively, it may issue an appropriate warning.

Therefore, it is recommended for thermodynamic software components (such as Property Packages) to implement the Persistence Common Interfaces.

The following methods should be implemented: Load, Save, GetClassID, IsDirty, GetSizeMax and, optionally, InitNew. Implementation of persistence also allows modification of the thermodynamic software component's configuration while it is part of the simulation case, e.g. via the Edit method of ICapeUtilities. Note that Property Packages should implement ICapeUtilities; as such, it is possible for a Property Package to have Edit functionality as well as to expose parameters (see Utility Common Interface Specification). For Property Packages allowing Edit functionality or modification by means of changing parameter values, persistence is a prerequisite, as otherwise the modifications will be lost between simulation sessions.

Implementation tips:

- The Load or (if Implemented) InitNew methods should be called before ICapeUtitilities::Initialize.
- The GetSizeMax method must be properly implemented as it is called by PMEs to allocate space for storing the content of a PMC . If GetSizeMax returns too small a value, this may lead to allocation failure.

5.10 Initialization and termination

A Thermo System software component is a top-level CAPE-OPEN object (or *PMC primary object* as outlined in the Methods & Tools integrated Guidelines documentation) and should therefore implement the ICapeUtilities interface. For a stand-alone Property Package, this also holds. The simulation environment is expected to call Initialize and Terminate on such objects.

A Property Package that is created from a Thermo System is, in accordance with the definition of *PMC primary objects* and *PMC secondary object* in the also a top-level CAPE-OPEN object. Therefore, it is expected of the Property Package to implement ICapeUtilities, and it is expected from the simulation environment that initialization and termination (and possibly persistence) are performed for such Property Packages as if it were a stand-alone Property Package.

6. Glossary of Terms Used

Chemical Compound (Compound) refers to a chemical species as defined by a particular set of physical properties calculation methods and data. In a sense, we are using the term Chemical Compound to refer to a mathematical model of the properties of a particular chemical species.

Chemical Species refers to a unique chemical substance, for example, "water".

Material. (sometimes referred to as Material Object). "Material" refers to a unique material with a specific composition, state, and set of physical properties. It may be a mixture or a pure compound, and be in one or multiple phases. It may be in any state of division (for example particulate material). Materials occur both in streams and within units (for example the liquid on a particular stage of a distillation column). We will generally be referring to a mathematical model of a material, when every material will be associated with a specific physical properties package. A material can be derived from a material template (see below). Specifically, the information defining a material will be the information in its associated Material Template plus, for a uniform molecular fluid, its temperature, pressure and the mole fraction of each compound. For multiple fluid phases, the same information will suffice if the Template defines that the phases are in equilibrium, otherwise a separate composition etc may be required for each phase. For dispersed phases, the Template will include the equations defining the general form of the size and shape distribution, the Material will include the parameters that define a specific material with a given mean particle size etc. (Such template distribution equations may be continuous, or be defined piece-wise over size ranges). It should be emphasized that the definition is only as complete as required by the user. Thus, where flow rate is required, Material will include flow rate, where (for transport properties) it requires scale and intensity of turbulence, it will contain this information. Where properties are not required, they will (or may) not be held. For example, heat transfer calculations may not require detailed composition information. Similarly, a mass balance only calculation may not include temperature or thermodynamic properties.

Material Template. A material template defines a complete set of chemical compounds and the associated properties package. Thus, for a single-phase molecular mixture, it normally only requires a definition of the composition, temperature and pressure to define a material completely. In many cases, the material template may define the permitted phase, or phases, of the material. Restriction on phases may be applied for several reasons. For example, the user may be confident that, at convergence, the material will be a vapor. It will make the computation faster and more reliable if the simulation avoids complex phase equilibrium computations for the intermediate iterations when spurious multiphase conditions might arise. As a further example, a user may be interested in computing a transfer rate between a vapor and a liquid. To compute any finite rate of transfer, there must be a lack of equilibrium, so that the liquid phase must be superheated and the vapor super-cooled. These non-equilibrium conditions can only be computed by performing separate property computations for the 2 phases, and restricting each calculation to a single phase calculation. (Although, of course, it will often be appropriate to compute the composition of the vapor that would be in equilibrium with the liquid and/or vice-versa). Computationally, there is likely to be a material class corresponding to a template, a material (object) will then inherit its interfaces from the material class.

Mixtures. The term "Mixture" is only used informally in this document.

Neutral Format. A data format that can be read and recognised by software other than the one that created it. There may be several neutral format standards defined under CAPE-OPEN (it is not restricted to properties packages).

Physical Property. In this document "physical property" includes all relevant properties. It thus includes both transport and thermodynamic properties of pure compounds and mixtures. If relevant, it would include other properties, such as colour.

Physical Property Calculation Method. An equation or algorithm, which can be used to calculate one or more physical properties. It should be emphasised that, except in the very limited number of cases defined in

(Ref), CAPE-OPEN will not define standard methods. In referring to calculation methods, this document includes both any standard methods to be defined and all proprietary methods that may be included in commercially available, or other, packages.

Physical Property Calculation Routine. A particular implementation of a physical property calculation method.

Physical Properties Executive. The physical properties executive is a component of a physical properties system that provides the user interface by which the methods, data and compounds can be selected. It also organizes the computation so that, in calculating material properties, the correct methods are employed for the specific material conditions. The executive provides access to additional services, such as the ability to correlate raw data to generate parameters for selected methods.

Physical Properties Package. A Properties Package (PP) is a complete, consistent, reusable, ready-to-use collection of methods, compounds and model parameters for calculating any of a set of known properties for the phases of a multiphase system. It includes all the pure compound methods and data, together with the relevant mixing rules and interaction parameters. A package normally covers only a small subset of the chemical compounds and methods accessible through a Properties System. It is thus established by selecting methods etc from within a larger system, possibly adding to or replacing these methods by third party compounds. These additional methods will normally be CAPE-OPEN compliant methods which may have been specially written, or may come from another properties system. (They can only come from another system where that system provides them as CAPE-OPEN compliant components).

Physical Properties Parameters. Numerical values, which either give physical properties directly (for example, molecular, or formula, weight), or permit properties to be computed by defined methods. For example, the coefficients of the Antoine equation for a particular chemical compound.

Physical Properties System. A software system that includes a physical properties executive, a set of physical properties routines and access to data for a number of chemical compounds. A Physical Properties System exposes one or more Physical Properties Packages. It will often access a large properties data bank. The system is likely to include text information, which the user can access to help select the most appropriate properties, methods and data for the particular application.

Software Component. A compiled piece of software which presents its services through well-specified interfaces, and is capable of being used and re-used in different software applications. In this context, a simulator could be a component, which itself makes use of other components such as physical properties systems, and calculation routines.

Stream. In this document, “stream” usually refers to “material stream”, namely a material and its flow rate. It may contain one or more compounds, and be made up of one or several phases. The material stream used here refers to the steam conditions at a particular point. For example, we may be referring to the material fed to or delivered from a process unit at a particular point in time. We are not referring to the whole of a stream in a length of pipe which may differ in condition from point to point. “Stream” is also used in describing the topology of a process, namely a connection between two unit models. Thus, where physico-chemical changes take place in a connecting pipe, the pipe itself will be represented by a unit model and the topological connections at the 2 ends of the pipe will be separate streams.