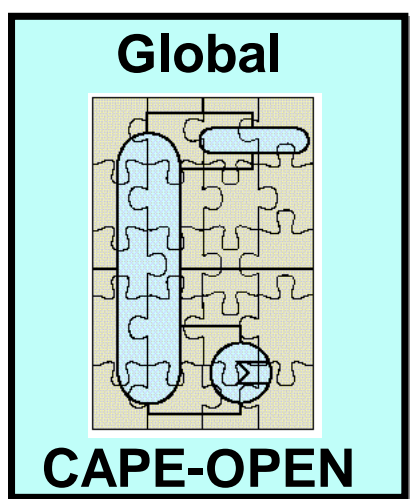


# Global CAPE-OPEN

Delivering the power of component software  
and open standard interfaces  
in computer-aided process engineering

---



## *CAPE-OPEN Open Interface Specifications*

### *Thermodynamic and Physical Properties Version 1.0*

## Archival Information

<b>Reference</b>	GCO Thermo Version 1.06.DOC
<b>Filename (if different)</b>	
<b>Authors</b>	
<b>Date</b>	30 <sup>th</sup> March 2002
<b>Number of Pages</b>	
<b>Version</b>	Version 1.06
<b>Reviewed by (date)</b>	<b>Reviewed by (name) (greyed out means not yet)</b>
<b>Distribution</b>	
<b>Additional Material</b>	
<b>Location on BSCW</b>	

## IMPORTANT NOTICES

### **Disclaimer of Warranty**

Global CAPE-OPEN documents and publications include software in the form of *sample code*. Any such software described or provided by Global CAPE-OPEN --- in whatever form --- is provided "as-is" without warranty of any kind. Global CAPE-OPEN and its partners and suppliers disclaim 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 the Global CAPE-OPEN project --- remains with you.

**Copyright © 2002 Global CAPE-OPEN and project partners and/or suppliers.** All rights are reserved unless specifically stated otherwise.

Global CAPE-OPEN is a collaborative research project established under BE 3512 "Industrial and Materials Technologies" (Brite-EuRam III), under contract BPR-CT98-9005.

### **Trademark Usage**

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in Global 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 part of a series of deliverables that together describe how to implement the level of commercially-supported, thermodynamic interoperability demonstrated in December 2001 in Cambridge at the Global CAPE-OPEN (GCO) Final Meeting. It is an update of the original CAPE-OPEN Thermodynamic and Physical Properties Interface Specification. The other deliverables are: example Physical Property Packages; the CAPE-OPEN Tester; the Thermo Wizard; and the video of the demonstration. All are available via the CAPE-OPEN Laboratories Network web site ([www.colan.org](http://www.colan.org)).

The document is consistent with the version 1.0 CAPE-OPEN COM type library, the other elements of the version 1.0 CAPE-OPEN Standard and the versions of Aspen Plus, HYSYS, gPROMS and Multiflash that were used for the Final Meeting demonstration (see [www.colan.org](http://www.colan.org) for current details). 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.



## **Acknowledgements**

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



# Contents

1	Introduction.....	10
2	Implementation Resources.....	10
3	CAPE-OPEN Identifiers.....	11
4	CAPE-OPEN Thermodynamic and Physical Properties Interface Specification.....	12
4.1	Introduction.....	12
4.2	Component Diagram & Supporting Interfaces.....	13
4.2.1	Material Classes - Description.....	13
4.3	General Remarks on Usage of Interface Methods.....	16
4.3.1	Convention Note on NULL and Empty.....	16
4.3.2	Argument interpretation of Get/Set/CalcProp Standard Methods.....	16
4.3.2.1	Values format.....	16
4.3.2.2	UNDEFINED interpretation.....	16
4.3.2.3	Overall interpretation.....	17
4.3.2.4	How to request information of non existing entities (such as components or phases).....	18
4.3.2.5	Fraction and Flow.....	18
4.3.2.6	Get/SetIndependentVar.....	19
4.3.2.7	Specific properties.....	19
4.3.2.8	Validity Check.....	20
4.4	CAPE-OPEN Calling Pattern Description.....	20
4.5	CAPE-OPEN Calling Pattern & Material Object.....	21
4.6	CAPE-OPEN Use Case Driven Component Model.....	22
4.6.1	Native Property Package Diagram.....	22
4.7	CAPE-OPEN Thermo Interface Diagram.....	24
4.8	Code Sample of CAPE-OPEN Calling Pattern & Material Object.....	25
4.8.1	Declare Material Object.....	25
4.8.2	Example 2: Calling a flash and then calculating a viscosity:.....	26
4.9	Interface Reference.....	28
4.9.1	ICapeThermoMaterialTemplate.....	28
4.9.1.1	CreateMaterialObject.....	28
4.9.1.2	SetProp.....	28
4.9.2	ICapeThermoMaterialObject.....	29
4.9.2.1	ComponentIds.....	29
4.9.2.2	PhaseIds.....	29
4.9.2.3	GetUniversalConstant.....	30
4.9.2.4	GetComponentConstant.....	30
4.9.2.5	CalcProp.....	31
4.9.2.6	GetProp.....	32
4.9.2.7	SetProp.....	33
4.9.2.8	CalcEquilibrium.....	34
4.9.2.9	SetIndependentVar.....	34
4.9.2.10	GetIndependentVar.....	35
4.9.2.11	PropCheck.....	35
4.9.2.12	AvailableProps.....	36
4.9.2.13	RemoveResults.....	36
4.9.2.14	CreateMaterialObject.....	37
4.9.2.15	Duplicate.....	38
4.9.2.16	ValidityCheck.....	38
4.9.2.17	GetPropList.....	39
4.9.2.18	GetNumComponents.....	39
4.9.3	ICapeThermoSystem.....	40
4.9.3.1	GetPropertyPackages.....	40
4.9.3.2	ResolvePropertyPackage.....	40
4.9.4	ICapeThermoPropertyPackage.....	41
4.9.4.1	GetComponentList.....	41
4.9.4.2	GetUniversalConstant.....	42
4.9.4.3	GetComponentConstant.....	43

4.9.4.4	CalcProp.....	44
4.9.4.5	CalcEquilibrium.....	45
4.9.4.6	PropCheck.....	46
4.9.4.7	ValidityCheck.....	47
4.9.4.8	GetPropList.....	47
4.9.4.9	GetPhaseList.....	48
4.9.5	ICapeThermoCalculationRoutine.....	49
4.9.5.1	CalcProp.....	49
4.9.5.2	PropCheck.....	50
4.9.5.3	PropList.....	50
4.9.5.4	ValidityCheck.....	51
4.9.6	ICapeThermoEquilibriumServer.....	52
4.9.6.1	CalcEquilibrium.....	52
4.9.6.2	PropCheck.....	53
4.9.6.3	ValidityCheck.....	54
4.9.6.4	PropList.....	55
4.10	COM MIDL.....	56
4.10.1	interface ICapeThermoSystem : IDispatch.....	56
4.10.2	interface ICapeThermoPropertyPackage : IDispatch.....	56
4.10.3	interface ICapeThermoMaterialTemplate : IDispatch.....	59
4.10.4	interface ICapeThermoMaterialObject : IDispatch.....	59
4.10.5	#endif //_COSE_IDL_interface ICapeThermoCalculationRoutine : IDispatch.....	62
4.10.6	interface ICapeThermoEquilibriumServer : IDispatch.....	63
4.11	CORBA CIDL.....	65
4.11.1	Base.....	65
4.11.2	COSE.....	65
4.11.3	Thermo System.....	70
4.11.4	Calculation routine.....	72
4.11.5	Equilibrium Server.....	73
4.12	CAPE-OPEN Properties List.....	75
4.12.1	Constant Properties Identifiers.....	75
4.12.2	Universal constant properties.....	76
4.12.3	Non-constant Properties (or Model Dependent Properties).....	77
4.12.3.1	Derivatives:.....	79
4.13	CAPE-OPEN Phase List.....	81
4.13.1	Phase Details.....	81
5	SI Units.....	82
6	Notes.....	83
6.1	Notes on Configuration of a Material Template.....	83
6.2	Notes on Argument interpretation of Get/Set/CalcProp Standard Methods).....	83
6.2.1	Non-constant properties.....	83
6.3	Notes on CalcProp description.....	83
6.3.1	CalcProp and CalcEquilibrium.....	83
6.3.2	Multiple calculations.....	83
6.3.3	Side-effects during calculation.....	83
6.4	Notes on Non-constant Properties (or Model Dependent Properties).....	84
6.5	Notes on Constant Properties Identifiers.....	84
6.5.1	Components supported by a Property Package.....	84
6.6	Notes on 2.11.2.4 GetComponentConstant.....	84
6.7	Description of component constants.....	85
6.7.1	CasRegistryNumber.....	85
6.8	Notes on Phases.....	85
6.8.1	Examples of valid phase equilibrium details.....	85
6.8.2	Existence of a phase.....	86
7	Glossary of Terms Used.....	87

## List of Figures

# 1 Introduction

This document describes the CAPE-OPEN Thermodynamic and Physical Properties Interfaces. It is part of a series of deliverables that together describe how to implement the level of commercially-supported, thermodynamic interoperability demonstrated in December 2001 in Cambridge at the Global CAPE-OPEN (GCO) Final Meeting. It is an update of the original CAPE-OPEN Thermodynamic and Physical Properties Interface Specification, to remove the ambiguities identified during interoperability testing and to add common services, such as error handling. The other deliverables are: the example Physical Property Packages; the CO Tester; the Thermo Wizard; and the video of the demonstration. All are available via the CAPE-OPEN Laboratories Network web site ([www.colan.org](http://www.colan.org)). The Interoperability Demonstration at the Final Meeting showed both unit operation and physical property components being used for steady-state simulation in an industrial flowsheet in Aspen Plus, HYSYS.Process and gPROMS CAPE-OPEN compliant Simulator Executives (COSE's), using components produced by AspenTech, Hyprotech, PSE and Infochem. In the case of gPROMS, a simple dynamic demonstration was also given. Details of the exact versions of these products that should be used for interoperability purposes can be found on the respective web sites of the companies concerned. Links to these web sites can be found on the CO-LaN web site ([www.colan.org](http://www.colan.org)).

The remainder of this document recommends an approach to the development and implementation of new property components and sockets and describes the specification of the Thermodynamic and Physical Properties Interfaces. The implementation information is based on the current Type Library 1.0, which is also compatible with the previous version 0.9.0. 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.

Implementation of a unit 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 [www.colan.org](http://www.colan.org)).

## 2 Implementation Resources

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

### 1. Physical Property Components

#### Resources:

- CAPE-OPEN Thermodynamic and Physical Property Interface Specification Version 1.0 (this document)
- The Hyprotech Physical Property Packages (HT PP's) in VB and C++
- The CO Tester
- The Thermo Wizard

- The demonstration video, which shows the current look and feel of open simulation interoperability

We recommend using the HT PP's as templates, with the two specification documents as references to resolve any difficulties. The C++ HT PP is more limited in scope, so is clearer to follow initially. The VB version provides a fuller example of the range of facilities possible. The CO Tester provides a means of testing that the new component conforms to CAPE-OPEN standards and the Thermo Wizard provides an alternative means of generating a thermo component.

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

## 2. Physical Property Sockets

The same resources are available for socket developers. In this case, the first thing required is to develop a Material Object. A suitable example is in the CO Tester and the HT PP's provide a means to test if the socket works correctly.

## 3 CAPE-OPEN Identifiers

In this document, we are following CO Methods and Tools recommendations 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: [www.colan.org/archive/specs/v09x/doc/03\\_CO\\_Methods\\_and\\_Tools\\_Recommendations.pdf](http://www.colan.org/archive/specs/v09x/doc/03_CO_Methods_and_Tools_Recommendations.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

We recommend that this should be adopted as CO policy.

## **4 CAPE-OPEN Thermodynamic and Physical Properties Interface Specification**

### **4.1 Introduction**

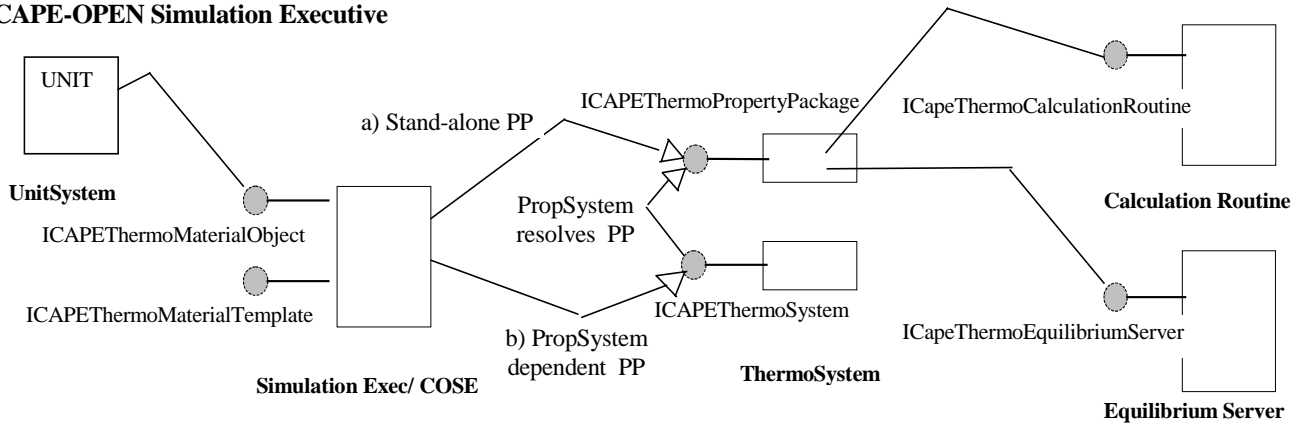
This Specification provides practical guidelines for using and implementing CAPE-OPEN Thermodynamic and Physical Properties (THRM) interfaces. It has been derived from the original THRM Specification published at the end of the CAPE-OPEN project, which can be found on the CO-LaN web site. As with all CAPE-OPEN Interface Specifications, it 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 includes COM and CORBA IDL with code examples to show how the interfaces are constructed and used.

## 4.2 Component Diagram & Supporting Interfaces

### CAPE-OPEN Component Diagram

#### CAPE-OPEN Simulation Executive



The Component Diagram shows the interfaces supported by each of the components in the CAPE-OPEN Thermodynamics and Physical Properties system. 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 component/simulation vendor. The diagram shows the two ways that any COSE can use to instantiate a Property Package:

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

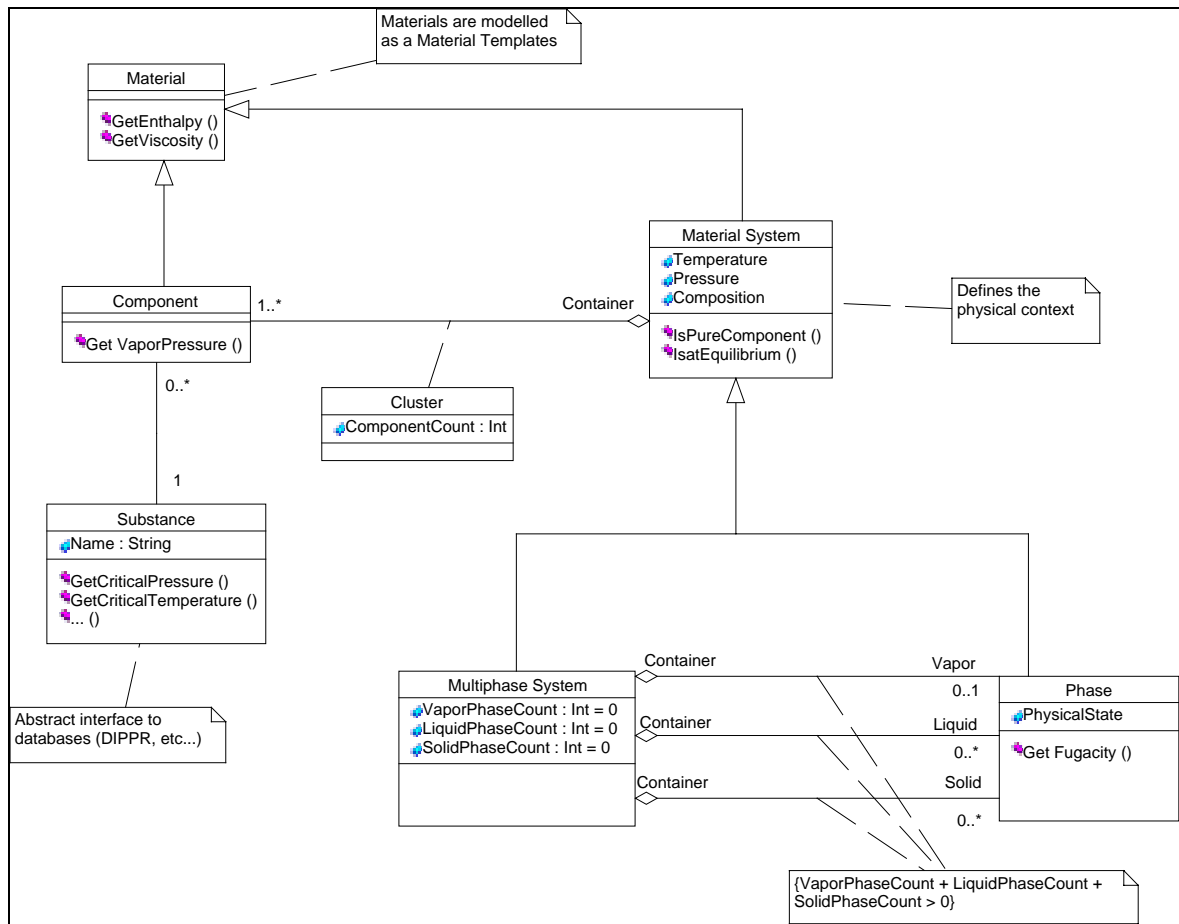
Route b), it is a Property System dependent (needing to be resolved by its Property System). In this case it is necessary to resolve the property package to get a handle to it.

Each white arrow represents the interface the COSE gets.

#### 4.2.1 Material Classes - Description

The creation of the Material Interfaces is consistent with the **UML Material Class Diagram below**. 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 are created from Material Templates. The Material Template definition consists of:

- A Component List ,
- A Phase Assumption,
- Reference to CAPE-OPEN property package,
- List of custom attributes for pseudo components



**UML Material Class Diagram**

The above diagram splits components into components and substances. Substances are the actual species, whereas components are instances of the substances used in the simulation. Since we can have more than one component based on the same concept of a substance. All parameters are ultimately associated with components, since any substance parameter could be overridden for a component. The end user of the Material Object is only concerned about components. The substance and database branch of 8-2 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 equilibrium server 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 by the Thermo Workpackage group to make property calls. This approach was developed in Thermo Workpackage. 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  
CapeArrayDouble propVals);
```

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

### 4.3 General Remarks on Usage of Interface Methods

#### 4.3.1 Convention Note on NULL and Empty

(See [Notes](#) for more information.)

Throughout the document NULL refers to a NULL BSTR for string arguments. emptyVariant is a VARIANT argument of the type VT\_EMPTY. See below how to implement these values in C++ and in VB.

Type of Data	Declaration and Usage
<b>BSTR</b>	<p><b>In C++</b></p> <p><b>Acceptable</b></p> <pre>BSTR strArg = NULL;</pre> <p><b>Not Acceptable</b></p> <pre>BSTR str = ::SysAllocString(L"");</pre> <p><b>In VB</b></p> <p><b>Acceptable</b></p> <pre>Dim strArg as String strArg = vbNullString</pre> <p><b>Not Acceptable</b></p> <pre>Dim strArg as String strArg = ""</pre>
<b>VARIANT</b>	<p><b>In C++</b></p> <pre>// the vt type of the VARIANT is // set to VT_EMPTY VARIANT VarArg; VariantInit(&amp;VarArg);</pre> <p><b>Remember that VariantInit must always be called after declaring a VARIANT</b></p> <p><b>In VB</b></p> <pre>Dim VarArg as Variant</pre>

#### 4.3.2 Argument interpretation of Get/Set/CalcProp Standard Methods

See [Notes](#) for more information.

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

##### 4.3.2.1 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).

##### 4.3.2.2 UNDEFINED interpretation

Be aware that the UNDEFINED value depends on the type of the corresponding argument. This special value is described in “Update on Types, Interfaces Naming and Undefined Values”, Global CAPE-OPEN, 2001.

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/ies 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).

### 4.3.2.3 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.	Typically Overall. (&)	Empty	Mixture	Scalar (1)
Enthalpy Entropy volume density viscosity thermal cond. HeatCapacity		Empty	Mixture	Scalar
		Empty	Pure	Vector (2:all components)
		Filled		Vector (3:specified comps)
fugacityCoefficient diffusivitycoeff activitycoeffi.		Empty	Mixture(*)	Vector (all components)
		Filled		Vector (specified comp)
		Empty	Pure	Vector (all components)
		Filled		Vector (specified comp)

#### Table comments

1. As stated in the specifications, the value is always an array. Scalar means here that the array will only contain 1 value
2. All components means the value of the property for each component of the material object
3. The value of the property for each component specified in argument complds.

(&) Other phases also allowed, since in the case of a multiphase system which is not thermodynamically in equilibrium (ex: rate-based approach on a distillation column stage), phases of a given system may be at different temperatures

(\*)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 component. In these particular properties (fugacityCoefficient, . . .), mixture/pure has a special meaning. In them, ‘mixture’ means that the property values refer to the components when they are within a fluid, and “pure” means that the property values refer to the components when they are in pure state (not mixed with other components).

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,...

#### 4.3.2.4 How to request information of non existing entities (such as components 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 COSEs may not support this functionality. It can also happen that a CAPE-OPEN-compliant Simulator Executive (COSE) allows setting a property for a particular phase, but only after flashing the stream.

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 identically zero, although the phase exists.

There's 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 components of the stream have been set. If the molar fractions are then requested for all the components, the call will fail. That's because, even if the values for some components are available, not all of them can be returned. Therefore, it is recommended to always set the molar/mass fractions for all the components at the same time, assigning a 0 value for the non-existing components.

#### 4.3.2.5 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 Thermo specification mentions “fraction”, but the list of official properties does not. We have agreed to remove molFraction, molFlow & massFlow, 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 vapour fraction, two new properties have been added: totalFlow and phaseFractions.

Property	Calc Type	Comp ID	Phase	Return Value
fraction	NULL	Empty	PH	Vector (all components)
flow		Filled	PH	Vector (specified components)
totalFlow	NULL	Empty	PH	Scalar
phaseFraction	NULL	Empty	PH	Scalar

Note: PH means that phase is defined.

Examples:

- To get a vector with the mole fraction of each component within the liquid phase. The sum of all values will be 1.

```
matObj.GetProp("fraction", "liquid", Empty, NULL, "mole")
```

- To get a vector with the mole fraction of some components within the vapor phase. The sum of all values will be the fraction that the set of specified components represent with respect to the whole vapor phase.

matObj.GetProp("fraction", "vapor", (vector of components), NULL, "mole")

- To get a scalar with the fraction of the fluid that is in the specified phase.

matObj.GetProp("phaseFraction", "vapor", Empty, NULL, "mole")

- To get a vector with the molar flows of each component within the liquid phase. The sum of all values should be equal to the total molar flow of the fluid

matObj.GetProp("flow", "liquid", Empty, NULL, "mole")

- To get a vector with the molar flow of some components within the vapor phase.

matObj.GetProp("flow", "vapor", (vector of components), NULL, "mole")

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

matObj.GetProp("totalFlow", "vapor", Empty, NULL, "mole")

#### 4.3.2.6 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 not be used. They will be removed in the next version of the Specification. 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.

#### 4.3.2.7 Specific properties

So far, several examples and implementations have used the following property identifiers:

enthalpy, entropy, energy, gibbsFreeEnergy, helmholtzFreeEnergy, volume

to 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 NULL value.

### 4.3.2.8 Validity Check

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

## 4.4 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 Simulation Executive, the Unit, 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 Simulation Executive, Unit, 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.<sup>1</sup>
- 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

---

<sup>1</sup> Brown, Malveau, McCormick, Mowbray; Anti Patterns Refactoring Software, Architectures, and Projects in Crisis, John Wiley & Sons, 1998

## 4.5 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.

### 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.

### Step 2: Set Values

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

### Step 3: Calculate

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

CalcEquilibrium

CalcProp

### 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, components, calculation type, and basis. Property results are in SI units (detailed information available at [http://www.bipm.fr/enus/3\\_SI/si.html](http://www.bipm.fr/enus/3_SI/si.html)).

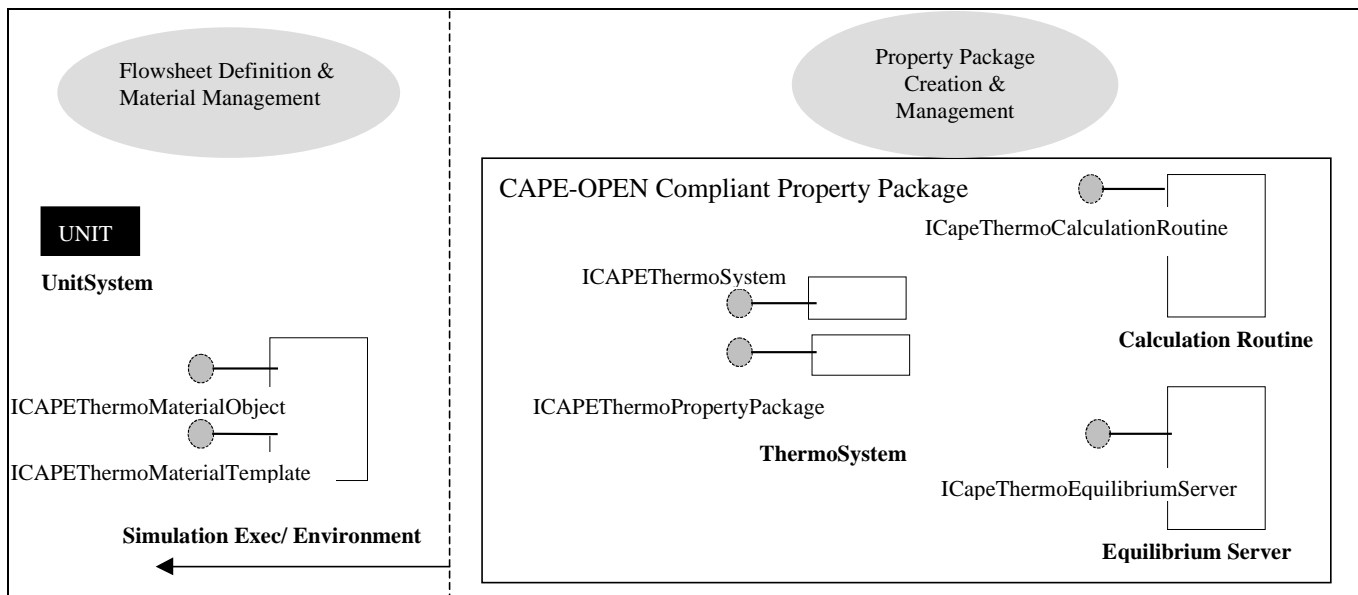
Specific code examples of this pattern are included in this document and are provided by Werner Drewitz of BASF.

The CAPE-OPEN calling pattern significantly reduces the complexity of the integration with existing native Thermo Systems by reducing the number of calls to the Open System Components. It also allows for the interfaces and contracts between these systems to be modified without addressing software issues. See notes on [Configuration of a Material Template](#) for more information.

## 4.6 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.

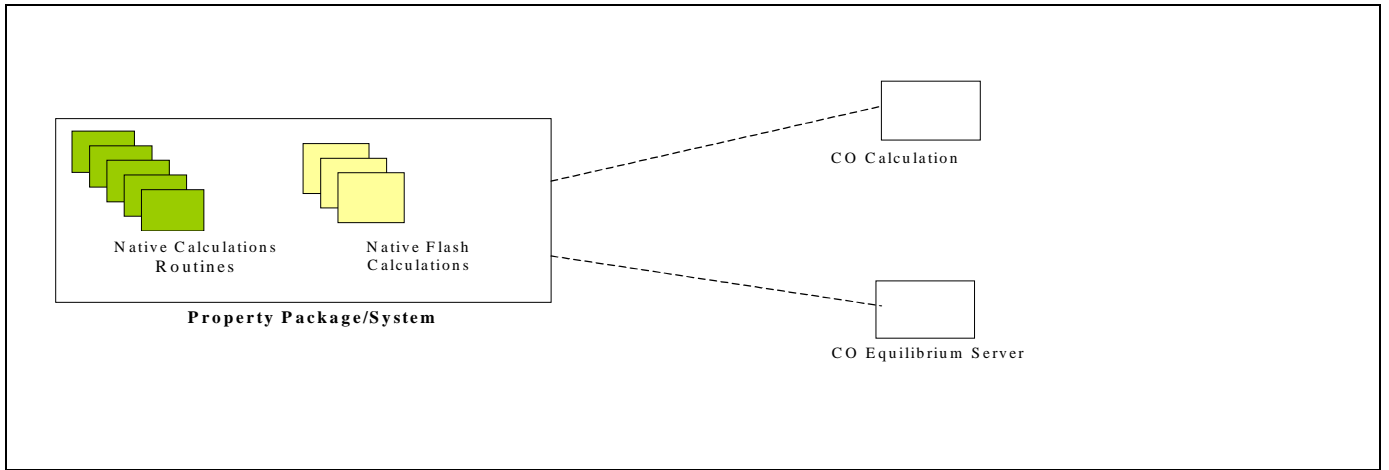
### CAPE-OPEN Component Model – Use Case Driven



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.

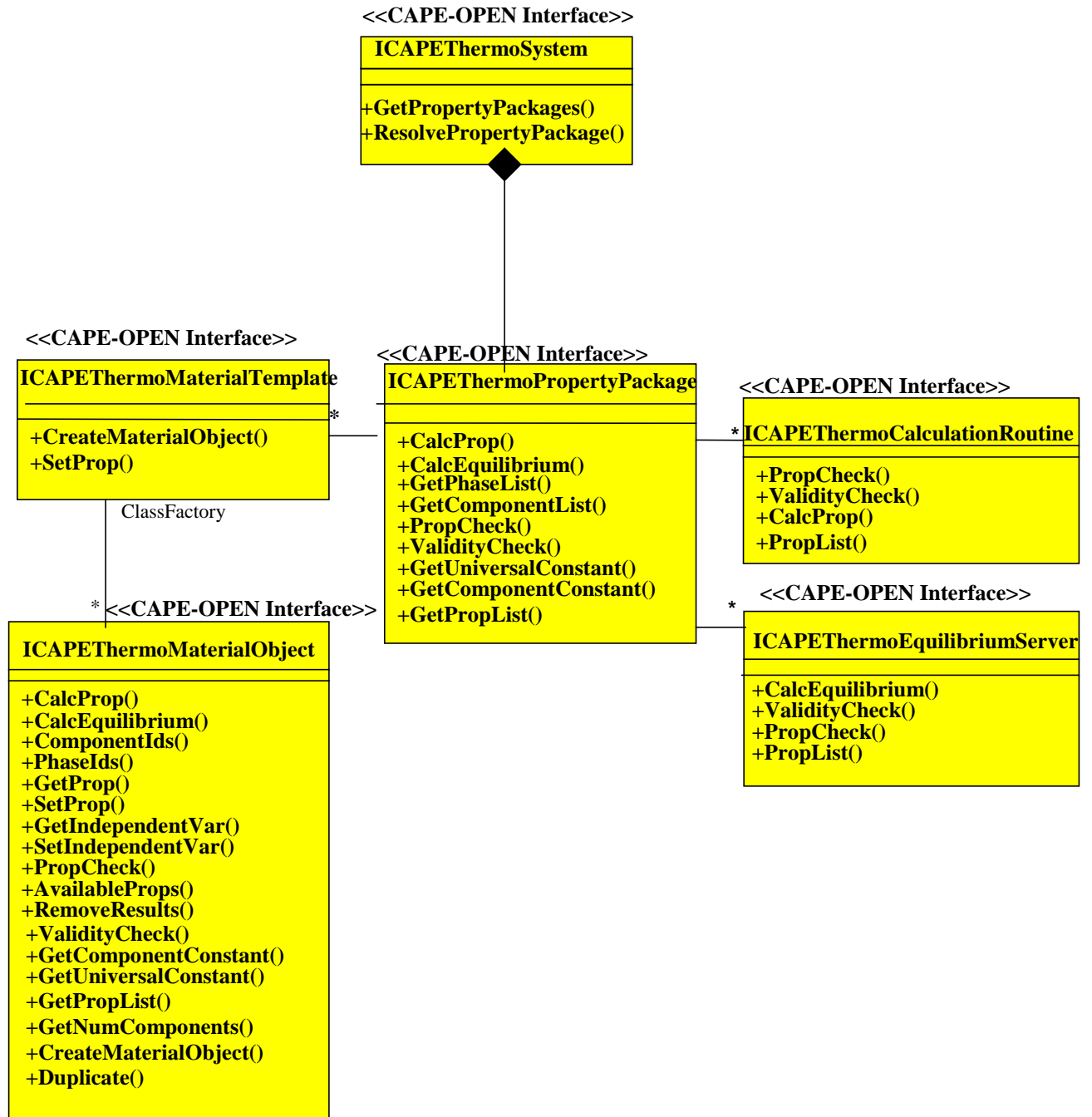
### 4.6.1 Native Property Package Diagram

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.



## 4.7 CAPE-OPEN Thermo Interface Diagram

This diagram does not describe how or when these interfaces are executed in the context of a working Open System / 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. Note: The results are stored in the Material Object and accessed through the `GetProp` method.

## 4.8 Code Sample of CAPE-OPEN Calling Pattern & Material Object

The following pseudo code example is detailed for the Material Object. However, the pattern by which values are set, calculated and retrieved is consistent for all Thermo System components. Example 1: Calling of liquid enthalpy property of a mixture

### 4.8.1 Declare Material Object

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

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

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

#### Step 3: Get Results

```
CapeArrayDouble val[2]; //double array created.  
CapeString basisQualifier;  
  
Strcpy(basisQualifier, "mole"); // set basis qualifier  
  
Imo.GetProp("Enthalpy", phaseQualifiers, emptyVariant, calculationType,  
basisQualifier, val);  
//get property enthalpy in the liquid phase
```

## 4.8.2 Example 2: Calling a flash and then calculating a viscosity:

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

### 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, emptyVariant, NULL, NULL, T);
Imo.SetProp("pressure", phaseQualifier, emptyVariant, NULL, NULL, P);
Imo.SetProp("fraction", phaseQualifiers, emptyVariant, NULL, "mole", F);
```

### Step 2 : Calculate Flash

```
CapeString flashTypeQualifier;

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

// call equilibrium server, no additional calculation of further
// properties ("NULL")

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

### 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);
```

#### Step 4: Get Results

```
CapeDoubleArray val;  
//double created.  
  
Imo.GetProp("viscosity", phaseQualifier, emptyVariant, calculationType,  
basisQualifier, val);  
  
//get property viscosity for the liquid phase from the  
//Material Object.
```

## 4.9 Interface Reference

### 4.9.1 ICapeThermoMaterialTemplate

#### 4.9.1.1 CreateMaterialObject

**Interface Name** ICapeThermoMaterialTemplate  
**Method Name** CreateMaterialObject  
**Returns** CapeError

---

**Description**

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

---

**Arguments**

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

---

#### 4.9.1.2 SetProp

**Interface Name** ICapeThermoMaterialTemplate  
**Method Name** SetProp  
**Returns** CapeError

---

**Description**

Allows custom property and values to be set on the Material Template to support pseudo components.

---

**Arguments**

Name	Type	Description
[in] property	CapeString	The custom property to set.
[in] values	CapeVariant (Double Array)	The actual values of the property.

---

## 4.9.2 ICapeThermoMaterialObject

### 4.9.2.1 ComponentIds

**Interface Name** ICapeThermoMaterialObject  
**Method Name** ComponentIds  
**Returns** CapeError

---

#### Description

Returns the list of components Ids of a given Material Object.

---

#### Arguments

Name	Type	Description
[out, retval] *compIds	CapeVariant (String Array)	Component IDs

---

### 4.9.2.2 PhaseIds

**Interface Name** ICapeThermoMaterialObject  
**Method Name** PhaseIds  
**Returns** CapeError

---

#### Description

It returns the phases existing in the MO at that moment. The Overall phase and multiphase identifiers cannot be returned by this method. See notes on [Existence of a phase](#) for more information.

---

#### Arguments

Name	Type	Description
[out, retval] *phaseIds	CapeVariant (String Array)	List of phases

---

### 4.9.2.3 GetUniversalConstant

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetUniversalConstant  
**Returns** CapeError

---

#### Description

Retrieves universal constants from the Property Package.

#### Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	List of universal constants to be retrieved
[out, retval] *propvals	CapeArrayVariant	Values of universal constants

---

### 4.9.2.4 GetComponentConstant

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetComponentConstant  
**Returns** CapeError

---

#### Description

Retrieve component constants from the Property Package. See [Notes](#) for more information.

---

#### Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	List of component constants
[in] compIds	CapeVariant (String Array)	List of component IDs for which constants are to be retrieved. emptyVariant for all components in the Material Object.
[out,retval] *propvals	CapeVariant (Variant Array)	Component Constant values returned from the Property Package for all the components in the Material Object It is a variant containing a 1 dimensional array of variants. If we call P to the number of requested properties and C to the number requested components 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 component). After them (from position C to 2*C-1) there will be the values of constants for the second requested property, and so on.

---

## 4.9.2.5 CalcProp

**Interface Name**            **ICapeThermoMaterialObject**  
**Method Name**             CalcProp  
**Returns**                    CapeError

---

### Description

This method is responsible for doing all property calculations and delegating these calculations to the associated thermo system. This method is further defined in the descriptions of the CAPE-OPEN Calling Pattern and the User Guide Section. See [Notes](#) for a more detailed explanation of the arguments and [CalcProp description](#) in the notes for a general discussion of the method.

### Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	The List of Properties to be calculated.
[in] phases	CapeVariant (String Array)	List of phases for which the properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Component Property. For partial property, such as fugacity coefficients of components in a mixture, use "Mixture" CalcType. For pure component fugacity coefficients, use "Pure" CalcType.

---

### 4.9.2.6 GetProp

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetProp  
**Returns** CapeError

---

#### Description

This method is responsible for retrieving the results from calculations from the MaterialObject. See [Notes](#) for a more detailed explanation of the arguments.

---

#### Arguments

Name	Type	Description
[in] property	CapeString	The Property for which results are requested from the MaterialObject.
[in] phase	CapeString	The qualified phase for the results.
[in] compIds	CapeVariant (String Array)	The qualified components for the results. emptyVariant to specify all components in the Material Object. For mixture property such as liquid enthalpy, this qualifier is not required. Use emptyVariant 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). Default is mole. Use NULL for default or as place holder for property for which basis does not apply (see also <a href="#">Specific properties</a> ).
[out, retval] *results	CapeVariant (Double Array)	Results vector containing property values in SI units arranged by the defined qualifiers.

### 4.9.2.7 SetProp

**Interface Name** ICapeThermoMaterialObject  
**Method Name** SetProp  
**Returns** CapeError

---

#### Description

This method is responsible for setting the values for properties of the Material Object. See [Notes](#) for a more detailed explanation of the arguments.

---

#### Arguments

Name	Type	Description
[in] property	CapeString	The property for which the values need to be set.
[in] phase	CapeString	Phase, if applicable. Use NULL for place holder.
[in] compIds	CapeVariant (String Array)	Components for which values are to be set. emptyVariant to specify all components in the Material Object. For mixture property such as liquid enthalpy, this qualifier is not required. Use emptyVariant 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 <a href="#">Specific properties</a> .
[in] values	CapeVariant (Double Array)	Values to set for the property.

---

### 4.9.2.8 CalcEquilibrium

**Interface Name** ICapeThermoMaterialObject  
**Method Name** CalcEquilibrium  
**Returns** CapeError

---

#### Description

This method is responsible for delegating flash calculations to the associated Property Package or Equilibrium Server. It must set the amounts, compositions, temperature and pressure 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. See [CalcProp](#) and [CalcEquilibrium](#) for more information.

---

#### Arguments

Name	Type	Description
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. emptyVariant for no properties. If a list, then the property values should be set for each phase present at equilibrium.

### 4.9.2.9 SetIndependentVar

**Interface Name** ICapeThermoMaterialObject  
**Method Name** SetIndependentVar  
**Returns** CapeError

---

#### Description

Sets the independent variable for a given Material Object.

---

#### Arguments

Name	Type	Description
[in] indVars	CapeVariant (String Array)	Independent variables to be set (see <b>names for state variables</b> for list of valid variables)
[in] values	CapeVariant (Double Array)	Values of independent variables.

---

#### 4.9.2.10 GetIndependentVar

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetIndependentVar  
**Returns** CapeError

---

##### Description

Returns the independent variables of a Material Object.

---

##### Arguments

Name	Type	Description
[in] indVars	CapeVariant (String Array)	Independent variables to be set (see <b>names for state variables</b> for list of valid variables)
[out, retval] *values	CapeArrayDouble	Values of independent variables.

---

#### 4.9.2.11 PropCheck

**Interface Name** ICapeThermoMaterialObject  
**Method Name** PropCheck  
**Returns** CapeError

---

##### Description

Checks to see if given properties can be calculated.

---

##### Arguments

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

#### 4.9.2.12 AvailableProps

**Interface Name** ICapeThermoMaterialObject  
**Method Name** AvailableProps  
**Returns** CapeError

---

##### Description

Gets a list properties that have been calculated.

---

##### Arguments

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

#### 4.9.2.13 RemoveResults

**Interface Name** ICapeThermoMaterialObject  
**Method Name** RemoveResults  
**Returns** CapeError

---

##### Description

Remove all or specified property results in the Material Object.

---

##### Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	Properties to be removed. emptyVariant to remove all properties.

---

#### 4.9.2.14 CreateMaterialObject

**Interface Name** ICapeThermoMaterialObject  
**Method Name** CreateMaterialObject  
**Returns** CapeError

---

##### Description

Create a Material Object from the parent Material Template of the current Material Object. This is the same as using the CreateMaterialObject method on the parent Material Template.

---

##### Arguments

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

---

#### 4.9.2.15 Duplicate

**Interface Name** ICapeThermoMaterialObject  
**Method Name** Duplicate  
**Returns** CapeError

---

##### Description

Create a duplicate of the current Material Object.

---

##### Arguments

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

---

#### 4.9.2.16 ValidityCheck

**Interface Name** ICapeThermoMaterialObject  
**Method Name** ValidityCheck  
**Returns** CapeError

---

##### Description

Checks the validity of the calculation.

---

##### Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	The properties for which reliability is checked.
[out, retval] *rellist	CapeArrayThermoReliability	Returns the reliability scale of the calculation.

---

#### 4.9.2.17 GetPropList

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetPropList  
**Returns** CapeError

---

##### Description

Returns list of properties supported by the property package and corresponding CO Calculation Routines. 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 Property Package could return the following list:

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

##### Arguments

Name	Type	Description
[out, retval] *props	CapeArrayString	String list of all supported properties of the property package.

---

#### 4.9.2.18 GetNumComponents

**Interface Name** ICapeThermoMaterialObject  
**Method Name** GetNumComponents  
**Returns** CapeError

---

##### Description

Returns number of components in Material Object.

##### Arguments

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

---

### 4.9.3 ICapeThermoSystem

#### 4.9.3.1 GetPropertyPackages

**Interface Name** ICapeThermoSystem  
**Method Name** GetPropertyPackages  
**Returns** CapeError

---

**Description**

Returns StringArray of property package names supported by the thermo system.

---

**Arguments**

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

#### 4.9.3.2 ResolvePropertyPackage

**Interface Name** ICapeThermoSystem  
**Method Name** ResolvePropertyPackage  
**Returns** CapeError

---

**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.

## 4.9.4 ICapeThermoPropertyPackage

### 4.9.4.1 GetComponentList

<b>Interface Name</b>	<b>ICapeThermoPropertyPackage</b>
<b>Method Name</b>	GetComponentList
<b>Returns</b>	CapeError

---

#### Description

Returns the list of components of a given property package.

In order to identify the components of a Property Package, the Executive will use the 'casno' argument instead of the compIds. The reason is that different COSEs may give different names to the same chemical compounds, whereas CAS Numbers are universal. Nevertheless, GetProp/SetProp... will still require their compIds argument to have the usual contents ("hydrogen","methane",...). Be aware that some simulators may have a limitation on the length of the names for pure components. Hence, it is recommended that each identifier returned by the compIds argument should not contain more than 8 characters. See notes on [Description of component constants](#) for more information.

If the package does not return a value for the 'casno' argument, or its value is not recognised by the Executive, then the compIds will be interpreted as the component's English name: such as "benzene", "water",... Obviously, it is recommended to provide a value for the casno argument.

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

---

#### Arguments

Name	Type	Description
[in,out] *compIds	CapeVariant (String Array)	List of component IDs
[in,out] *formulae	CapeVariant (String Array)	List of component formulae
[in,out] *name	CapeVariant (String Array)	List of component names.
[in,out] *boilTemps	CapeVariant (Double Array)	List of boiling point temperatures.
[in,out] *molwt	CapeVariant (Double Array)	List of molecular weight.

[in,out] *casno	CapeVariant (String Array)	List of CAS number .
--------------------	-------------------------------	----------------------

#### 4.9.4.2 GetUniversalConstant

**Interface Name** ICapeThermoPropertyPackage

**Method Name** GetUniversalConstant

**Returns** CapeError

---

#### Description

Returns the values of the Universal Constants.

---

#### Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The Material object.
[in] props	CapeVariant (String Array)	List of requested universal constants.
[out,retval] *propvals	CapeArrayVariant	Values of universal constants.

---

### 4.9.4.3 GetComponentConstant

**Interface Name**      **ICapeThermoPropertyPackage**

**Method Name**        **GetComponentConstant**

**Returns**              **CapeError**

---

#### Description

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

---

#### Arguments

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The material object.
[in] props	CapeVariant (String Array)	The list of properties.
[out,retval] *propvals	CapeVariant (Variant Array)	Component Constant values. See description of return value of the ICapeThermoMaterialObject GetComponentConstant method.

---

#### 4.9.4.4 CalcProp

**Interface Name** ICapeThermoPropertyPackage  
**Method Name** CalcProp  
**Returns** CapeError

---

##### Description

This method is responsible for doing all calculations and is implemented by the associated thermo system. This method is further defined in the descriptions of the CAPE-OPEN Calling Pattern and the User Guide Section.

---

##### Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The MaterialObject for the Calculation.
[in] props	CapeVariant (String Array)	The List of Properties to be calculated.
[in] phases	CapeVariant (String Array)	List of phases for which the properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Component Property. For partial property, such as fugacity coefficients of components in a mixture, use "Mixture" CalcType. For pure component fugacity coefficients, use "Pure" CalcType.

---

#### 4.9.4.5 CalcEquilibrium

**Interface Name** ICapeThermoPropertyPackage  
**Method Name** CalcEquilibrium  
**Returns** CapeError

---

##### Description

Method responsible for calculating/delegating flash calculation requests. It must set the amounts, compositions, temperature and pressure 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. See [CalcProp](#) and [CalcEquilibrium](#) for more information.

---

##### Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The MaterialObject
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. emptyVariant for no properties. If a list, then the property values should be set for each phase present at equilibrium.

---

#### 4.9.4.6 PropCheck

**Interface Name**      **ICapeThermoPropertyPackage**  
**Method Name**        PropCheck  
**Returns**                CapeError

---

##### **Description**

Check to see if properties can be calculated.

---

##### **Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] props	CapeVariant (String Array)	List of Properties to check.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for each property.

---

#### 4.9.4.7 ValidityCheck

**Interface Name** ICapeThermoPropertyPackage  
**Method Name** ValidityCheck  
**Returns** CapeError

---

**Description**

Checks the validity of the calculation.

---

**Arguments**

Name	Type	Description
[in] materialObject	CapeInterface *	The material object for the calculations.
[in] props	CapeVariant (String Array)	The list of properties to check.
[out, retval] *rellist	CapeArrayThermoReliability	The properties for which reliability is checked.

---

#### 4.9.4.8 GetPropList

**Interface Name** ICapeThermoPropertyPackage  
**Method Name** GetPropList  
**Returns** CapeError

---

**Description**

Returns list of Thermo System supported properties. 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 Property Package could return the following list:

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

---

**Arguments**

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

#### 4.9.4.9 GetPhaseList

**Interface Name**        **ICapeThermoPropertyPackage**  
**Method Name**         **GetPhaseList**  
**Returns**                **CapeError**

---

##### **Description**

Provides the list of the supported phases. When supported, the Overall phase and multiphase identifiers must be returned by this method.

---

##### **Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[out, retval] *phases	CapeArrayString	The list of phases supported by the property package.

---

## 4.9.5 ICapeThermoCalculationRoutine

### 4.9.5.1 CalcProp

**Interface Name** ICapeThermoCalculationRoutine  
**Method Name** CalcProp  
**Returns** CapeError

---

#### Description

This method is responsible for doing all calculations on behalf of the calculation routine component. This method is further defined in the descriptions of the CAPE-OPEN Calling Pattern and the User Guide Section.

#### Arguments

Name	Type	Description
[in] materialObject	CapeInterface *	The material object of the calculation.
[in] props	CapeVariant (String Array)	The List of Properties to be calculated.
[in] phases	CapeVariant (String Array)	List of phases for which the properties are to be calculated.
[in] calcType	CapeString	Type of calculation: Mixture Property or Pure Component Property. For partial property, such as fugacity coefficients of components in a mixture, use "Mixture" CalcType. For pure component fugacity coefficients, use "Pure" CalcType.

---

### 4.9.5.2 PropCheck

**Interface Name**      **ICapeThermoCalculationRoutine**  
**Method Name**        PropCheck  
**Returns**                CapeError

---

**Description**

Checks to see if a given property can be calculated.

---

**Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] props	CapeVariant (String Array)	List of Properties to check.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for each property.

---

### 4.9.5.3 PropList

**Interface Name**      **ICapeThermoCalculationRoutine**  
**Method Name**        PropList  
**Returns**                CapeError

---

**Description**

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

---

**Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in,out] *props	CapeVariant (String Array)	List of supported properties.
[in,out] *phases	CapeVariant (String Array)	List of supported phases.
[in,out] *calcType	CapeVariant (String Array)	List of supported calculation types. (Pure & Mixture)

#### 4.9.5.4 ValidityCheck

**Interface Name**            **ICapeThermoCalculationRoutine**  
**Method Name**             **ValidityCheck**  
**Returns**                    **CapeError**

---

**Description**

Checks the validity of the calculation.

---

**Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The material object for the calculations.
[in] props	CapeVariant (String Array)	The list of properties to check.
[out, retval] *rellist	CapeArrayThermoReliability	The properties for which reliability is checked.

---

## 4.9.6 ICapeThermoEquilibriumServer

### 4.9.6.1 CalcEquilibrium

**Interface Name** ICapeThermoEquilibriumServer  
**Method Name** CalcEquilibrium  
**Returns** CapeError

---

#### Description

Calculates the equilibrium properties requested. It must set the amounts, compositions, temperature and pressure 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. See [CalcProp](#) and [CalcEquilibrium](#) for more information.

---

#### Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	MaterialObject of the calculation
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. emptyVariant for no properties. If a list, then the property values should be set for each phase present at equilibrium.

---

## 4.9.6.2 PropCheck

**Interface Name**            **ICapeThermoEquilibriumServer**

**Method Name**             **PropCheck**

**Returns**                    **CapeError**

---

### 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

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The Material Object for the calculations.
[in] flashType	CapeString	Type of flash calculation to check
[in] props	CapeVariant (String Array)	List of Properties to check. emptyVariant for none.
[out, retval] *valid	CapeArrayBoolean	The array of booleans for flash and property. First element is reserved for flashType.

---

### 4.9.6.3 ValidityCheck

**Interface Name**        **ICapeThermoEquilibriumServer**  
**Method Name**         **ValidityCheck**  
**Returns**                **CapeError**

---

#### **Description**

Checks the reliability of the calculation.

---

#### **Arguments**

<b>Name</b>	<b>Type</b>	<b>Description</b>
[in] *materialObject	CapeInterface	The material object for the calculations.
[in] props	CapeVariant (String Array)	The list of properties to check. NULL for none.
[out, retval] *rellist	CapeArrayThermo Reliability	The properties for which reliability is checked. First element reserved for reliability of flash calculations.

---

#### 4.9.6.4 PropList

**Interface Name** ICapeThermoEquilibriumServer

**Method Name** PropList

**Returns** CapeError

---

#### 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	CapeVariant (String Array)	Type of flash calculations supported.
[in,out] *props	CapeVariant (String Array)	List of supported properties.
[in,out] *phases	CapeVariant (String Array)	List of supported phases.
[in,out] *calcType	CapeVariant (String Array)	List of supported calculation types. (Pure & Mixture)

## 4.10 COM MIDL

### 4.10.1 interface ICapeThermoSystem : IDispatch

```
#ifndef _THERMOSYSTEM_IDL_
#define _THERMOSYSTEM_IDL_
// Provide an interface for Thermo System
// Definition of the Thermo System configuration
import "oaidl.idl";
import "ocidl.idl";

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

// Material Template and Material Object
#include "Cose.idl"

// ICapeThermoSystem interface
[
    object,
    uuid(ICapeThermoSystem_IID),
    dual,
    helpstring("ICapeThermoSystem Interface"),
    pointer_default(unique)
]
interface ICapeThermoSystem : IDispatch
{
    // Get the list of available property packages
    [id(1), helpstring("method GetPropertyPackages")]
    HRESULT GetPropertyPackages([out, retval] CapeArrayString
*propPackageList);

    // Resolve a particular property package
    [id(2), helpstring("method ResolvePropertyPackage")]
    HRESULT ResolvePropertyPackage(
        [in] CapeString propertyPackage,
        [out, retval] CapeInterface *propPackObject);
};
#endif //_THERMOSYSTEM_IDL_
```

### 4.10.2 interface ICapeThermoPropertyPackage : IDispatch

```
#ifndef _THERMOPROPERTYPACKAGE_IDL_
#define _THERMOPROPERTYPACKAGE_IDL_
// Provide an interface for Thermo System
// Definition of the Thermo System configuration
import "oaidl.idl";
import "ocidl.idl";

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"
```

```

// Material Template and Material Object
#include "Cose.idl"

// Definition of the Property Package configuration
// interface: ICapeThermoPropertyPackage
[
    object,
    uuid(ICapeThermoPropertyPackage_IID),
    dual,
    helpstring("ICapeThermoPropertyPackage Interface"),
    pointer_default(unique)
]

interface ICapeThermoPropertyPackage : IDispatch
{
    // Get the phase list
    [id(1), helpstring("method GetPhaseList")]
    HRESULT GetPhaseList([out, retval]CapeArrayString *phases);

    // Get the component list
    [id(2), helpstring("method GetComponentList")]
    HRESULT GetComponentList( [in,out] CapeVariant *compIds,
                              [in,out] CapeVariant *formulae,
                              [in,out] CapeVariant *name,
                              [in,out] CapeVariant *boilTemps,
                              [in,out] CapeVariant *molwt,
                              [in,out] CapeVariant *casno);

    // Get some universal constant(s)
    [id(3), helpstring("method GetUniversalConstant")]
    HRESULT GetUniversalConstant([in] CapeInterface materialObject,
                                  [in] CapeVariant props,
                                  [out, retval] CapeArrayDouble *propVals);

    // Get some pure component constant(s)
    [id(4), helpstring("method GetComponentConstant")]
    HRESULT GetComponentConstant([in] CapeInterface materialObject,
                                  [in] CapeVariant props,
                                  [out, retval] CapeArrayDouble *propVals);

    // Calculate some properties
    [id(5), helpstring("method CalcProp")]
    HRESULT CalcProp( [in] CapeInterface materialObject,
                     [in] CapeVariant props,
                     [in] CapeVariant phases,
                     [in] CapeString calcType);

    // Calculate some equilibrium values
    [id(6), helpstring("method CalcEquilibrium")]
    HRESULT CalcEquilibrium( [in] CapeInterface materialObject,
                             [in] CapeString flashType,
                             [in] CapeVariant props);

    // Check a property is valid
    [id(7), helpstring("method PropCheck")]
    HRESULT PropCheck( [in] CapeInterface materialObject,
                      [in] CapeVariant props,
                      [out, retval] CapeArrayBoolean *valid);

    // Check the validity of the given properties

```

```
        [id(8), helpstring("method ValidityCheck")]
        HRESULT ValidityCheck([in] CapeInterface materialObject,
                               [in] CapeVariant props,
                               [out, retval] CapeArrayThermoReliability
*valid);

        // Get the list of properties
        [id(9), helpstring("method GetPropList")]
        HRESULT GetPropList([out, retval] CapeArrayString *props);
};

#endif // _THERMOPROPERTYPACKAGE_IDL_
```

### 4.10.3 interface ICapeThermoMaterialTemplate : IDispatch

### 4.10.4 interface ICapeThermoMaterialObject : IDispatch

```
#ifndef _COSE_IDL_
#define _COSE_IDL_
//
// The key contents of this file are the open interfaces for the
// ICapeMaterialObject and ICapeMaterialTemplate.
// All of these are fundamental interfaces for the physical properties
// CAPE-OPEN standard.
import "oidl.idl";
import "ocidl.idl";

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

// The ThermoReliability object is still an uncertain
// interface. This object holds some measure of the reliability of
// the physical property calculation. It might be a boolean. It
// might be an enumerated type, or it might be a real number.
[
    object,
    uuid(ICapeThermoReliability_IID),
    dual,
    helpstring("ICapeThermoReliability Interface"),
    pointer_default(unique)
]
interface ICapeThermoReliability : IDispatch
{
    // TO BE DEFINED
};

// Type definition of the interface for subsequent use
typedef LPDISPATCH CapeThermoReliabilityInterface;

// Array of ICapeThermoReliability interfaces (as IDispatch)
typedef VARIANT CapeArrayThermoReliability;

// Material Template interface
[
    object,
    uuid(ICapeThermoMaterialTemplate_IID),
    dual,
    helpstring("ICapeThermoMaterialTemplate Interface"),
    pointer_default(unique)
]

interface ICapeThermoMaterialTemplate : IDispatch
{
    // Create a material object from this Template
    [id(1), helpstring("method CreateMaterialObject")]
    HRESULT CreateMaterialObject(
        [out, retval] CapeInterface *materialObject);

    // Set some property value(s)
```

```

        [id(2), helpstring("method SetProp")]
        HRESULT SetProp( [in] CapeString property,
                        [in] CapeVariant values);
};

// Type definition of the interface for subsequent use
typedef LPDISPATCH CapeThermoMaterialTemplateInterface;

// Material object interface
[
    object,
    uuid(ICapeThermoMaterialObject_IID),
    dual,
    helpstring("ICapeThermoMaterialObject Interface"),
    pointer_default(unique)
]
interface ICapeThermoMaterialObject : IDispatch
{
    // Get the component ids for this MO
    [propget, id(1), helpstring("property ComponentIds")]
    HRESULT ComponentIds([out, retval]CapeVariant *compIds);

    // Get the phase ids for this MO
    [propget, id(2), helpstring("property PhaseIds")]
    HRESULT PhaseIds([out, retval] CapeVariant *phaseIds);

    // Get some universal constant(s)
    [id(3), helpstring("method GetUniversalConstant")]
    HRESULT GetUniversalConstant( [in] CapeVariant props,
                                [out, retval] CapeArrayDouble
*propVals);

    // Get some pure component constant(s)
    [id(4), helpstring("method GetComponentConstant")]
    HRESULT GetComponentConstant([in] CapeVariant props,
                                [in] CapeVariant compIds,
                                [out, retval] CapeArrayDouble *propVals);

    // Calculate some properties
    [id(5), helpstring("method CalcProp")]
    HRESULT CalcProp( [in] CapeVariant props,
                    [in] CapeVariant phases,
                    [in] CapeString calcType);

    // Get some properties values
    [id(6), helpstring("method GetProp")]
    HRESULT GetProp( [in] CapeString property,
                    [in] CapeString phase,
                    [in] CapeVariant compIds,
                    [in] CapeString calcType,
                    [in] CapeString basis,
                    [out, retval] CapeArrayDouble *results);

    // Set some properties values
    [id(7), helpstring("method SetProp")]
    HRESULT SetProp( [in] CapeString property,
                    [in] CapeString phase,
                    [in] CapeVariant compIds,
                    [in] CapeString calcType,
                    [in] CapeString basis,

```

```

[in] CapeVariant values);

// Calculate some equilibrium values
[id(8), helpstring("method CalcEquilibrium")]
HRESULT CalcEquilibrium([in] CapeString flashType,
[in] CapeVariant props);

// Set the independent variable for the state
[id(9), helpstring("method SetIndependentVar")]
HRESULT SetIndependentVar([in]CapeVariant indVars,
[in]CapeVariant values);

// Get the independent variable for the state
[id(10), helpstring("method GetIndependentVar")]
HRESULT GetIndependentVar([in] CapeVariant indVars,
[out, retval] CapeArrayDouble
*values);

// Check a property is valid
[id(11), helpstring("method PropCheck")]
HRESULT PropCheck([in] CapeVariant props,
[out, retval] CapeArrayBoolean *valid);

// Check which properties are available
[id(12), helpstring("method AvailableProps")]
HRESULT AvailableProps([out, retval] CapeArrayString *props);

// Remove any previously calculated results for given
// properties
[id(13), helpstring("method RemoveResults")]
HRESULT RemoveResults([in] CapeVariant props);

// Create another empty material object
[id(14), helpstring("method CreateMaterialObject")]
HRESULT CreateMaterialObject([out, retval] CapeInterface
*materialObject);

// Duplicate this material object
[id(15), helpstring("method Duplicate")]
HRESULT Duplicate([out, retval] CapeInterface *clone);

// Check the validity of the given properties
[id(16), helpstring("method ValidityCheck")]
HRESULT ValidityCheck(
[in] CapeVariant props,
[out, retval] CapeArrayThermoReliability *relList);

// Get the list of properties
[id(17), helpstring("method GetPropList")]
HRESULT GetPropList([out, retval] CapeArrayString *props);

// Get the number of components in this material object
[id(18), helpstring("method GetNumComponents")]
HRESULT GetNumComponents([out, retval] CapeLong *numComp);
};

// Type definition of the interface for subsequent use
typedef LPDISPATCH CapeThermoMaterialObjectInterface;

```

#### 4.10.5 #endif //\_COSE\_IDL\_interface ICapeThermoCalculationRoutine : IDispatch

```
#ifndef _CALCROUTINE_IDL_
#define _CALCROUTINE_IDL_
// Title: CAPE-OPEN Foreign Calculation Routine
//
// Provides an interface for foreign calculation routines.
import "oaidl.idl";
import "ocidl.idl";

// Include GUIDs
#include "COGuids.idl"

// Fundamental CAPE-OPEN types and interfaces
#include "Fundamental.idl"

// Material Templates and Material Objects
#include "Cose.idl"

// Definition of interface: ICapeThermoCalculationRoutine
// ICapeThermoCalculationRoutine is a mechanism for adding foreign
// calculation routines to a physical property package.
[
    object,
    uuid(ICapeThermoCalculationRoutine_IID),
    dual,
    helpstring("ICapeThermoCalculationRoutine Interface"),
    pointer_default(unique)
]
interface ICapeThermoCalculationRoutine : IDispatch
{
    HRESULT CalcProp( [in] CapeInterface materialObject,
                     [in] CapeVariant props,
                     [in] CapeVariant phases,
                     [in] CapeString calcType);

    HRESULT PropCheck([in] CapeInterface materialObject,
                     [in] CapeVariant props,
                     [out, retval] CapeArrayBoolean *valid );

    HRESULT PropList( [in,out] CapeVariant *props,
                     [in,out] CapeVariant *phases,
                     [in,out] CapeVariant *calcType);

    HRESULT ValidityCheck( [in] CapeInterface materialObject,
                           [in] CapeVariant props,
                           [out, retval] CapeArrayThermoReliability
*rellist);
};

// Type definition of the interface for subsequent use
typedef LPDISPATCH CapeThermoCalculationRoutineInterface;

#endif //_CALCROUTINE_IDL_
```

#### 4.10.6 interface ICapeThermoEquilibriumServer : IDispatch

```
#ifndef _EQSRVR_IDL_
#define _EQSRVR_IDL_
//
// An external routine to perform flash calculations.
import "oaidl.idl";
import "ocidl.idl";
// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

// Material Template and Material Object
#include "Cose.idl"

// Definition of interface: ICapeThermoCalculationRoutine
// ICapeThermoCalculationRoutine is a mechanism for adding foreign
// calculation routines to a physical property package.
[
    object,
    uuid(ICapeThermoEquilibriumServer_IID),
    dual,
    helpstring("ICapeThermoEquilibriumServer Interface"),
    pointer_default(unique)
]
interface ICapeThermoEquilibriumServer : IDispatch
{
    // Calculate some equilibrium values
    [id(1), helpstring("method CalcEquilibrium")]
    HRESULT CalcEquilibrium([in] CapeInterface materialObject,
                            [in] CapeString flashType,
                            [in] CapeVariant props);

    // Check a property is valid
    [id(2), helpstring("method PropCheck")]
    HRESULT PropCheck([in] CapeInterface materialObject,
                     [in] CapeString flashType,
                     [in] CapeVariant props,
                     [out, retval] CapeArrayBoolean *valid );

    // Check the validity of the given properties
    [id(3), helpstring("method ValidityCheck")]
    HRESULT ValidityCheck( [in] CapeInterface materialObject,
                           [in] CapeVariant props,
                           [out, retval] CapeArrayThermoReliability
*relList);

    // Get the list of properties
    [id(4), helpstring("method PropList")]
    HRESULT PropList( [in,out] CapeVariant *flashType,
                     [in,out] CapeVariant *props,
                     [in,out] CapeVariant *phases,
                     [in,out] CapeVariant *calcType);
};

// Type definition of the interface for subsequent use
typedef LPDISPATCH CapeThermoEquilibriumServerInterface;
```

```
#endif // _EQSRVR_IDL_
```

## 4.11 CORBA CIDL

### 4.11.1 Base

```
// always put something like this in to prevent multiple processing

#ifndef CAPEBASE_IDL
#define CAPEBASE_IDL

module Cape {

    // Forward declarations

    interface Identification;

    // sequence of ICapeIdentification objects
    typedef sequence<ICapeIdentification> ICapeIdentificationSequence;

    // elementary type definitions
    typedef long      CapeLong;
    typedef double   CapeDouble;
    typedef string   CapeString;
    typedef boolean  CapeBoolean;
    typedef string   CapeDate;
    typedef any      CapeVariant;

    // sequence definitions

    typedef sequence<CapeLong>      CapeLongSequence;
    typedef sequence<CapeDouble>   CapeDoubleSequence;
    typedef sequence<CapeString>   CapeStringSequence;
    typedef sequence<CapeBoolean>  CapeBooleanSequence;
    typedef sequence<CapeDate>     CapeDateSequence;
    typedef sequence<CapeVariant>  CapeVariantSequence;

    // interface definitions

    interface ICapeIdentification {
        CapeString GetComponentName();
        CapeString GetComponentDescription();
        CapeString GetVersionNumber();
    };

};

#endif /* CAPEBASE_IDL */
```

### 4.11.2 COSE

```
/ The key contents of this file are the interfaces for the
// ICapeMaterialObject and ICapeMaterialTemplate.
// All of these are fundamental interfaces for the CAPE-OPEN
// standard.

#include "base.idl"

#ifndef THERMO_IDL
#define THERMO_IDL
```

```

module Cose {

    /* Forward declaration of interfaces */

    interface ICapeThermoMaterialObject;
    interface ICapeThermoReliability;
    interface ICapeThermoMaterialTemplate;

    /* Type definitions and sequences */

    typedef sequence<ICapeThermoReliability> CapeThermoReliabilitySequence;

    /* Exceptions */

    exception CapeThermoUnknownObject {
        // This exception is raised by destroyObject when obj is not
        // owned
        // by this material template.
    };

    exception CapeThermoBoundsException {
        // The ICapeBoundsError exception is thrown when there is an
        // error in the calculation or if the results are totally
        // unreliable (e.g. the state variables are completely outside
        // the physical property model's range of applicability).

        Cape::CapeString message;
    };

    exception CapeThermoUnknownIdentifierException {
        // Thrown when an identifier for phase, component, or property
        // cannot be recognized

        // kind holds the category that has been asked for ("Phase",
        // "Component", "IndependentVariable", or "Property"

        // id holds the actual id that could not be found

        Cape::CapeString kind;
        Cape::CapeString id;
    };

    // The ThermoReliability object is still an uncertain
    // interface. This object holds some measure of the reliability of
    // the physical property calculation. It might be a boolean. It
    // might be an enumerated type, or it might be a real number.

    /* Interfaces */

    interface ICapeThermoReliability {
        // contents to be filled in later
    };

    interface ICapeThermoMaterialObject {

        // MATERIAL OBJECT DATA

        Cape::CapeStringSequence GetComponentIds();
    };
}

```

```

// GetComponentIds provides two useful pieces of information:
// 1. the names of each component in the material object
// 2. the ordering used for liquidComposition,
//    vaporComposition, and any result that has a separate
//    value for each component.

Cape::CapeStringSequence GetPhaseIds();
// The list of possible phases represented by the Material
//Object

// MATERIAL OBJECT FUNCTIONS

Cape::CapeDoubleSequence
GetUniversalConstant(in Cape::CapeStringSequence props)
raises (CapeThermoUnknownIdentifierException);
/* Retrieves universal constants specified in props from
the property package */

Cape::CapeDoubleSequence
GetComponentConstant(in Cape::CapeStringSequence props,
                    in Cape::CapeStringSequence compIds)
raises (CapeThermoUnknownIdentifierException);
/* Retrieves component constants specified in props from
the property package */

void CalcProp(in Cape::CapeStringSequence props,
              in Cape::CapeStringSequence phases,
              in Cape::CapeString calcType)
raises(CapeThermoBoundsException,
       CapeThermoUnknownIdentifierException);

// This routine will calculate properties
// requested in the list props for the given material object.
// The results are returned in ICapeResultSet. An error in the
// calculation is indicated with a ICapeBoundsError exception.

// props: identifiers of properties to calculate

// phases: identifiers of phases for which to calculate

// calcType: calculation mode, "Mixture" or "Pure"

Cape::CapeDoubleSequence
GetProp(in Cape::CapeString property,
        in Cape::CapeString phase,
        in Cape::CapeStringSequence compIds,
        in Cape::CapeString calcType,
        in Cape::CapeString basis)
raises(CapeThermoBoundsException,
       CapeThermoUnknownIdentifierException);

/* This routine will retrieve calculated properties
requested in the list properties for the given material
object.
The results are accessed using GetProp call. An error in
The calculation is indicated with a ICapeBoundsError
exception.

props                the list of properties to be calculated

phases              the list of phases for which properties

```

```

                                are to be calculated

    calcType                    "Mixture" or "Pure", indicating type
                                of calculation

    Returns:                    Results vector containing property values.
                                To be freed by Caller.

    */

void SetProp(in Cape::CapeString property,
             in Cape::CapeString phase,
             in Cape::CapeStringSequence compIds,
             in Cape::CapeString calcType,
             in Cape::CapeString basis,
             in Cape::CapeDoubleSequence values)
raises(CapeThermoBoundsException,
       CapeThermoUnknownIdentifierException);

/* This routine will set properties to calculate
   requested in the list properties for the given material
   object.
   The results are accessed using GetProp call. An error in
   The calculation is indicated with a ICapeBoundsError
   exception.

   props                        the list of properties to be calculated

   phases                      the list of phases for which properties
                               are to be calculated

   calcType                    "Mixture" or "Pure", indicating type of
                               calculation

   Values                      Vector of property values. Allocated and
                               freed by caller.

   returns:                    Nothing.

    */

void
CalcEquilibrium(in Cape::CapeString flashType,
                in Cape::CapeStringSequence props)
raises(CapeThermoBoundsException,
       CapeThermoUnknownIdentifierException);

/* This routine will calculate the equilibrium properties
   requested in the list props for the given material object
   by performing a flash calculation. An error in the
   calculation is indicated with a CapeThermoBoundsException or
   a CapeThermoUnknownIdentifierException

   props                        the list of properties to be calculated

   flashType                   the type of flash calculation to be
                               performed according to list of
                               flashTypes in specification

    */
void SetIndependentVar(in Cape::CapeStringSequence indVars,
                      in Cape::CapeDoubleSequence values)
raises (CapeThermoBoundsException,

```

```

        CapeThermoUnknownIdentifierException);

/* Sets the values of independent variables

    indVars      identifier for the independent variables to set
    values       values of the variables

    throws       CapeThermoBoundsException if bounds are
                 inappropriate
                 CapeThermoUnknownIdentifierException if
                 variable is not independent or unknown. */

Cape::CapeDouble GetIndependentVar(in Cape::CapeStringSequence
indVars)
    raises (CapeThermoUnknownIdentifierException);

/* retrieves values independent Variables

    indVars      identifier for the independent variables to set

    throws       CapeThermoUnknownIdentifierException if
                 variable is not independent or unknown. */

Cape::CapeBooleanSequence
PropCheck(in Cape::CapeStringSequence props);

/* before performing any calculations with a material
   objective,
   this routine should be called at least once

    props        the list of properties to be calculated

    returns      The result for each property (TRUE
                 indicates
                 that the property can be successfully
                 calculated).
                 The value at index i is the result for
                 property props[i]
*/

CapeThermoReliabilitySequence
ValidityCheck(in Cape::CapeStringSequence props)
raises(CapeThermoUnknownIdentifierException);

/* This function is used to check the reliability of the
   property
   calculations at the converged point of the solution strategy
   or
   presumably at any point along the way.  rellist[i] provides
   some measure of the reliability of the calculation of
   property
   props[i].

    props        the list of properties to validate
    returns      the reliability measure for each
*/
};

interface ICapeThermoMaterialTemplate : Cape::ICapeIdentification {

```

```

// the only open interface for a material template is the
// ability to create MaterialObjects.

ICapeThermoMaterialObject CreateMaterialObject();

    // Creates a new material object

void DestroyMaterialObject(in ICapeThermoMaterialObject obj)
    raises(CapeThermoUnknownObject);

    /* When a client is completely done with a material object, it
    should ask the material template to destroy it. It is the
    clients responsibility to make sure that all of the references
    to obj have been removed. */

void
SetProp (in Cape::CapeString property,
         in Cape::CapeDoubleSequence values)
    raises (CapeThermoBoundsException,
           CapeThermoUnknownIdentifierException);
    // Allows custom property and values to be set on the material
    //template to support pseudo components

};
};
#endif

```

### 4.11.3 Thermo System

```

#include "base.idl"
#include "cose.idl"

#ifndef THERMO_SYSTEM_IDL
#define THERMO_SYSTEM_IDL

module ThermoSystem {

    interface ICapeThermoPropertyPackage;
    interface ICapeThermoSystem;

    /* Sequence definitions */
    typedef sequence<ICapeThermoPropertyPackage>
        CapeThermoPropertyPackageSequence;

    typedef sequence<ICapeThermoSystem>
        CapeThermoSystemSequence;

    /* Interface definitions */

    interface ICapeThermoSystem : Cape::ICapeIdentification {

        Cape::CapeStringSequence GetPropertyPackages();
        // returns a name for all available Property Packages

        ICapeThermoPropertyPackage
        ResolvePropertyPackage(in Cape::CapeString propPkg)
            raises (Cose::CapeThermoUnknownIdentifierException);
    };
};

```

```

    // This method returns Property Package Interface pointer for
    // given Property Package. Exception indicates that package is
    //not available
};

interface ICapeThermoPropertyPackage : Cape::ICapeIdentification {

    // PROPERTIES OF THE CONFIGURED PHYSICAL PROPERTY PACKAGE

    Cape::CapeStringSequence GetPhaseList();

    // provide the list of phases understood by the
    // physical property package

    void GetComponentList (out Cape::CapeStringSequence compIds,
                           out Cape::CapeStringSequence formulae,
                           out Cape::CapeStringSequence names,
                           out Cape::CapeDoubleSequence boilTemps,
                           out Cape::CapeDoubleSequence molWeight,
                           out Cape::CapeStringSequence casNo);

    Cape::CapeDoubleSequence
    GetUniversalConstant(in Cose::ICapeThermoMaterialObject matObj,
                        in Cape::CapeStringSequence props)
        raises (Cose::CapeThermoUnknownIdentifierException);

    Cape::CapeDoubleSequence
    GetComponentConstant(in Cose::ICapeThermoMaterialObject matObj,
                        in Cape::CapeStringSequence props)
        raises (Cose::CapeThermoUnknownIdentifierException);

    // calculate component properties

    void
    CalcProp(in Cose::ICapeThermoMaterialObject mobject,
            in Cape::CapeStringSequence props,
            in Cape::CapeStringSequence phases,
            in Cape::CapeString CalcType)
        raises(Cose::CapeThermoBoundsException,
            Cose::CapeThermoUnknownIdentifierException);

    // This routine will calculate properties
    // requested in the list props for the given material object.
    // The results are stored in the material object. An error in
    // the calculation is indicated with a Bounds exception. If a
    // requested property is unknown, a UnknownPropertyException is
    // thrown
    // props          the list of properties to be calculated

    void
    CalcEquilibrium(in Cose::ICapeThermoMaterialObject matObj,
                   in Cape::CapeString flashType,
                   in Cape::CapeStringSequence props)
        raises(Cose::CapeThermoBoundsException,
            Cose::CapeThermoUnknownIdentifierException);

    // This routine will calculate the equilibrium properties
    // requested in the list props for the given material object
    // by performing a flash calculation. The results are returned

```

```

// in ICAPEResultSet. An error in the calculation is indicated
// with a ICAPEBoundsError exception.
//
// matObj      the material object where the results are stored
//             in
// flashType   the flash calculation to be performed
//
// props       the list of properties to be calculated

Cape::CapeBooleanSequence
PropCheck(in Cose::ICapeThermoMaterialObject matObj,
          in Cape::CapeStringSequence props);

// before performing any calculations with a material
// objective, this routine should be called at least once
//
// props       the list of properties to be calculated
//
// valid       the result for each property (TRUE
//             indicates that the property can be
//             successfully be calculated).
//             valid[i] is the result for
//             property props[i]

Cose::CapeThermoReliabilitySequence
ValidityCheck(in Cose::ICapeThermoMaterialObject matObj,
              in Cape::CapeStringSequence props)
raises(Cose::CapeThermoUnknownIdentifierException);

// This function is used to check the reliability of the
// property calculations at the converged point of the solution
// strategy or presumably at any point along the way.

// rellist[i] provides some =measure of the reliability of the
// calculation of property
// props[i].
//
// props       the list of properties to validate
//
// returns     the reliability measure for each
//             calculation

Cape::CapeStringSequence GetPropList();

// Returns the list of properties supported by the thermo
// system
};

};

#endif

```

#### **4.11.4 Calculation routine**

```

#ifndef CALCROUTINE_IDL
#define CALCROUTINE_IDL

#include "base.idl"
#include "cose.idl"

```

```

module CalculationRoutine {

    // ICapeThermoCalculationRoutine is a mechanism for adding foreign
    // calculation routines to a physical property package.

    interface ICapeThermoCalculationRoutine : Cape::ICapeIdentification {

        void
        CalcProp(in Cose::ICapeThermoMaterialObject matObj,
                in Cape::CapeStringSequence props,
                in Cape::CapeStringSequence phases,
                in Cape::CapeString calcType)
        raises(Cose::CapeThermoBoundsException,
              Cose::CapeThermoUnknownIdentifierException);

        // This function is expected to calculate a list of properties
        // and stores the result in the Material Object matObj
        // If an error occurs or if the results are utterly meaningless
        // (e.g. the material object's state is outside the physical
        // property calculation's range of applicability), an
        // CapeBoundsException exception is raised.
        // If unknown properties or phases are requested, a
        // CapeUnknownIdentifierException

        void
        PropList(out Cape::CapeStringSequence props,
                out Cape::CapeStringSequence phases,
                out Cape::CapeStringSequence calcType);

        // List of properties supported by the Calculation Routine.
        // and phases and types.

        Cape::CapeBooleanSequence
        PropCheck(    in Cose::ICapeThermoMaterialObject matObj,
                    in Cape::CapeStringSequence props);

        // check that this external calculation routine can calculate
        // the property for this material object. Return TRUE if the
        // physical property can be evaluated; and otherwise, return
        // FALSE.

        Cose::CapeThermoReliabilitySequence
        ValidityCheck(    in Cose::ICapeThermoMaterialObject matObj,
                        in Cape::CapeStringSequence props)
        raises (Cose::CapeThermoUnknownIdentifierException);

        // This function is used to check the reliability of the
        // property calculation at the converged point of the solution
        // strategy or presumably at any point along the way to
        // convergence. The return value indicates the reliability of
        // this foreign calculation on matObj
    };
};

#endif

```

#### 4.11.5 Equilibrium Server

```

#include "base.idl"
#include "cose.idl"

```

```

#ifndef EQSERVER_IDL
#define EQSERVER_IDL

module EquilibriumServer {

    interface ICapeThermoEquilibriumServer : Cape::ICapeIdentification {

        void
        CalcEquilibrium(in Cose::ICapeThermoMaterialObject matObj,
                       in Cape::CapeString flashType,
                       in Cape::CapeStringSequence props)
        raises(Cose::CapeThermoBoundsException,
              Cose::CapeThermoUnknownIdentifierException);

        // This routine will calculate the equilibrium properties
        // requested in the list props for the given material object
        // by performing a flash calculation. The results are returned
        // in
        // ICapeResultSet. An error in the calculation is indicated
        // with a ICapeBoundsError exception.
        //
        // mobject                provides the state information
        // props                   the list of properties to be
        //                         calculated =
        //
        // props is a generalization of "property" that allows
        // calculation of multiple properties simultaneously.

        void
        PropList(out Cape::CapeStringSequence flashType,
                out Cape::CapeStringSequence props,
                out Cape::CapeStringSequence phases,
                out Cape::CapeStringSequence calcType);

        // List of properties supported by the Calculation Routine.
        // and phases and types.

        Cape::CapeBooleanSequence
        PropCheck(in Cose::ICapeThermoMaterialObject matObj,
                 in Cape::CapeString flashType,
                 in Cape::CapeStringSequence props);

        // check that this external calculation routine can calculation
        // the property for this material object. Return TRUE if the
        // physical property can be evaluated; and otherwise, return
        // FALSE.

        Cose::CapeThermoReliabilitySequence
        ValidityCheck(in Cose::ICapeThermoMaterialObject matObj,
                     in Cape::CapeStringSequence props)
        raises (Cose::CapeThermoUnknownIdentifierException);

        // This function is used to check the reliability of the
        // property calculation at the converged point of the solution
        // strategy or presumably at any point along the way to
        // convergence. The return value indicates the reliability of
        // this foreign calculation on mobject.
    };
};

#endif

```

## 4.12 CAPE-OPEN Properties List

### 4.12.1 Constant Properties Identifiers

See [Notes](#) for more explanation on the table and [CAPE-OPEN Identifiers](#) for naming conventions. Units of measure can be found in the section on [SI Units](#).

To identify a pure component there are some attributes, which are not really 'properties', but are never the less needed:

Nidentifier	Character string for identification e.g. in a flowsheet or PPS
iupacName	Complete IUPAC Name
casRegistryNumber	Chemical Abstract Sequencing Number
chemicalFormula	Chemical formula (brutto, nomenclature according to Hill)
structureFormula	Chemical structure formula

The name of a component should be the same in the whole flowsheet. There is a need in every flowsheet calculation for a global component list! If one uses different Property Packages, it is very probable that there are different names used for the same component. So a translation list may help in which, for each name of the global component list, there are the names for the correspondent components 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 <sup>3</sup> /mol
criticalCompressibilityFactor	Critical Compressibility Factor	
criticalDensity	Critical Density	mol/m <sup>3</sup>
acentricFactor	Pitzer Acentric Factor	
dipoleMoment	Dipole Moment	Cm
parachor	Parachor	m <sup>3</sup> kg <sup>0.25</sup> /(s <sup>0.5</sup> mol)
gyrationRadius	Radius of Gyration	m
associationParameter	Association-Parameter (Hayden-O'Connell)	
diffusionVolume	Diffusion volume	m <sup>3</sup>
diffusionCoefficient	Diffusion coefficient	m <sup>2</sup> /s
vanderwaalsVolume	Van der Waals Volume	m <sup>3</sup>
vanderwaalsArea	Van der Waals Area	m <sup>2</sup>
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 <sup>3</sup>
liquidVolumeAt25C	Liquid Volume at 25 C	m <sup>3</sup> /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).

#### **4.12.2 Universal constant properties**

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

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

### 4.12.3 Non-constant Properties (or Model Dependent Properties)

See [Notes](#) for more explanation of the revised table and [CAPE-OPEN Identifiers](#) for general naming conventions. Units of measurement can be found in the section on [SI Units](#).

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 <sup>3</sup> /mol
surfaceTension	Surface Tension	N/m
expansivity	coefficient of linear expansion (Expansivity) $\frac{1}{L} \left. \frac{\partial L}{\partial T} \right _P$	1/K
compressibility	$\frac{1}{V} \left. \frac{\partial V}{\partial P} \right _T$	1/Pa
compressibilityFactor	Compressibility Factor $Z = \frac{PV}{RT}$	
jouleThomsonCoefficient	$\left. \frac{\partial T}{\partial P} \right _H$	K/Pa
heatOfVaporization	**	J/mol
heatOfSublimation	**	J/mol
heatOfFusion	**	J/mol
heatOfSolidSolidPhaseTransition	**	J/mol
volumeChangeUponVaporization	**	m <sup>3</sup> /mol
volumeChangeUponSublimation	**	m <sup>3</sup> /mol
volumeChangeUponMelting	**	m <sup>3</sup> /mol
volumeChangeUponSolidSolidPhaseTransition	**	m <sup>3</sup> /mol
heatCapacity	Heat Capacity (Cp)**	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 <sup>3</sup> /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 <sup>3</sup> /mol
viscosity	Viscosity	Pa s
thermalConductivity	Thermal Conductivity	W/(m K)
selfDiffusionCoefficient		m <sup>2</sup> /s
fugacity	Fugacity	Pa

fugacityCoefficient	Fugacity Coefficient	
activity	Activity	
activityCoefficient	Activity Coefficient	
dewPointPressure		Pa
dewPointTemperature		K
kvalues	K Factors of a pair of phases in equilibrium	
logFugacityCoefficient	Logarithm of fugacity coefficients	
logkvalues	Logarithm of kvalues	
temperature		K
pressure		Pa
volume	Volume*	m <sup>3</sup> /mol
density	Density **	mol/m <sup>3</sup>
enthalpy	Enthalpy*	J/mol
entropy	Entropy*	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 component within a given phase (or the whole mixture)**	mol/s
fraction	List of the partial molar (or mass) fractions of each component 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 ration of the capacity of a condenser with a particaulr 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, $\phi = - n_w \ln (x_w f_w) / (n_s \sum v_i)$ where, n <sub>w</sub> is the moles of water; n <sub>s</sub> is the moles of solute; x <sub>w</sub> is the mole fraction of	

	water; $f_w$ is the symmetric activity coefficient of water; $\nu_i$ is the stoichiometric coefficient of component 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.

#### 4.12.3.1 Derivatives:

Derivatives are built from the property identifier, a point with a D meaning "Derivative" and the name for the variable:

property.Dtemperature	derivative of property according to Temperature
property.Dpressure	derivative of property according to Pressure
property.DmolFraction	derivative of property according to mole fraction
property.Dmoles	derivative of property according to mole numbers

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 Property Package could return the following list:

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



## **4.13 CAPE-OPEN Phase List**

### **4.13.1 Phase Details**

Permitted phases have been restricted to the following:

<b>Phase</b>	<b>Description</b>
Vapour	Vapour phase
Liquid	Liquid phase
Solid	Solid phase
Overall	All phases

At the moment, only one phase of each type is allowed to be present, in general. See [Notes](#) for information on a partial work around and [Existence of a phase](#) for information on phase checking.

## 5 SI Units

The current standards do not specifically state which SI unit has to be used for each dimension. It will be added to the specification documentation in a future revision. For the moment, 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](http://www.bipm.fr/enus/3_SI/si.html)) for more information.

## **6 Notes**

### **6.1 Notes on Configuration of a Material Template**

Although the CAPE-OPEN specifications defined the ICapeThermoMaterialTemplate interface, no methodology was defined to access or create instances of objects that implemented these interfaces.

We will add information on using the Material Template interface in future versions of the Specification.

### **6.2 Notes on Argument interpretation of Get/Set/CalcProp Standard Methods)**

#### **6.2.1 Non-constant properties**

There is not enough definition for passing information on vapour fraction, as well as an inconsistency in the specification. There is no way to choose vapour fraction among the state variables, while the specification provides for a pressure-vapour fraction flash. At the moment, Global CAPE-OPEN is not addressing the generalisation of the flash. It could mean changing the attributes of GetIndependentVar and SetIndependentVar in a future version of the specification.

### **6.3 Notes on CalcProp description**

#### **6.3.1 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.

#### **6.3.2 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.

#### **6.3.3 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.

## 6.4 Notes on Non-constant Properties (or Model Dependent Properties)

Modifications:

a) Since there exist the “compressibilityCoefficient” and “compressibilityFactor” properties, it is not clear what the “compressibility” property means. On looking in some handbooks (e.g., Perry), compressibility and isothermal compressibility are used to refer to the same quantity.  $(1/V)(dV/dP)$  at constant T. We have therefore:

- Renamed the property identifier “compressibilityCoefficient” to “compressibility” leaving its meaning  $(1/V)(dV/dP)$  at constant T
- Removed the “compressibility” property on page 85 of the original specification

b) ‘kvalues’ carries similar information to ‘fugacityCoefficient’, but it’s more logical to publish the value of the former property.

## 6.5 Notes on Constant Properties Identifiers

The most important role of the component constants is to identify the components supported by a Property Package. The ICapeThermoPropertyPackage GetComponentList method was designed for this purpose.

### 6.5.1 Components supported by a Property Package

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

## 6.6 Notes on 2.11.2.4 GetComponentConstant

Equivalences between GetComponentList arguments and component constant properties:

GetComponentList Arguments	Component Constant Property Identifier	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 (brutto, nomenclature according to Hill)  
structureFormula Chemical structure formula

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

## **6.7 Description of component constants**

### **6.7.1 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>

Components can be accessed directly with  
<http://webbook.nist.gov/cgi/cbook.cgi?Formula=ch4&Nolon=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 comparison has to be done by looking at constant properties.

## **6.8 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, the permitted phases have been restricted, as shown, pending an improvement to phase treatment in a future update.

### **6.8.1 Examples of valid phase equilibrium details**

VaporLiquid	Vapor-liquid equilibrium
LiquidSolid	Solid-liquid equilibrium
VaporLiquidSolid	3 coexisting phase: vapor, liquid and solid
VaporSolid	Solid-vapor equilibrium, sublimation

However, Type Library 1.0 does allow the calculation of single-phase properties for streams with any number of liquid phases, since we can calculate them independently. For example:

```
mo.SetProp("fraction","liquid", ...,liquid1FractionValue)  
mo.calcProp("enthalpy","liquid")  
mo.GetProp("fraction","liquid", ...,liquid1EnthalpyValue)
```

```
mo.SetProp("fraction","liquid", ...,liquid2FractionValue)  
mo.calcProp("enthalpy","liquid")  
mo.GetProp("fraction","liquid", ...,liquid2EnthalpyValue)
```

Unfortunately, multi-phase properties are not supported for 2 liquid phases, since the Material Object cannot cope simultaneously with 2 liquid phases. So, calcprop("kvalues","liquidLiquid") is not 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
```

### **6.8.2 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 (the COSE knows this information). 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.

## 7 Glossary of Terms Used

**Chemical Component (Component)** 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 Component 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 component, 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 is 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 component. 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 components 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 vapour. 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 vapour 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 vapour supercooled. 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 vapour 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).

**Option Set.** An Option Set is a Simple Properties Package. It is used when 2 or more Simple Properties Packages refer the same set of components and have the majority of methods and data in

common. Some vendors then find it more efficient to combine the SPP's into one Properties Package; each SPP being referred to as an Option Set. The Option Sets are identified by simple titles, such as "high pressure" or "low pressure".

**Physical Property.** In this document "physical property" includes all relevant properties. It thus includes both transport and thermodynamic properties of pure components 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 Data.** Physical Properties Data includes both Physical Properties Parameters and Raw Physical Properties Data. When used without further description, the term will generally refer just to parameters.

**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 components 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 Simple Properties Package (SPP) is a complete, consistent, reusable, ready-to-use collection of methods, chemical components and model parameters for calculating any of a set of known properties for the phases of a multiphase system. It includes all the pure component methods and data, together with the relevant mixing rules and interaction parameters. A package normally covers only a small subset of the chemical components 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 components. 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). A Properties Package may be a Simple Properties Package, or at a vendors discretion, made up from Option Sets (see definition of Option Set).

**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 component.

**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 components. 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. Properties Systems implement two kinds of interfaces, external interfaces to the Simulator Executive and unit clients, and internal interfaces to calculation routines and neutral files. Since only the Properties System understands the interrelations between its constituent properties routines, a UNIT cannot communicate to a calculation routine directly, but only through a properties package set up within a specific Properties System.

**Raw Physical Properties Data.** A data set containing physical property values for a specified list of chemical species (or mixture of chemical species) at specified conditions. It may be used for correlating parameters of calculation routines. The 'raw' data may be obtained by experimental measurement (so that the methods chosen correlate the measurements as closely as possible) or may have been generated by a properties routine in one properties system so that the values calculated by a routine in a second system can be made to match the first as closely as possible.

**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 components, 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.