CAPE-OPEN

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

Thermodynamic and Physical Properties

v1.1

www.colan.org

ARCHIVAL INFORMATION

Filename	CO Thermo 1.1 Specification.doc
Authors	CO-LaN consortium
Status	Public
Date	10 May 2011
Version	version 3.11
Number of pages	124
Versioning	version 3 edited by Richard Szczepanski (Infochem) and Jasper van Baten (AmsterCHEM)
	Version 2.22 edited by Michel Pons (CO-LaN), Richard Szczepanski (Infochem) and Jasper van Baten (AmsterCHEM)
Additional material	COM IDL
Web location	
Implementation specifications version	1.1
Comments	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN documents and publications include software in the form of *sample code*. Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN 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 CAPE-OPEN Laboratories Network --- remains with you.

CO-LaN is a not for profit organization established under French law of 1901.

Trademark Usage

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN 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. These interfaces allow software components providing thermodynamic and physical property calculations to be used in a Process Modelling Environment (PME). The first part of the document describes a conceptual model that shows how these components are used in a PME. The second half contains reference material for each method in the interfaces.

ACKNOWLEDGEMENTS

Many individuals and their organizations have contributed to this document. The following were amongst the main ones:

Peter Banks	Peter Banks Associates, representing BP
Werner Drewitz	BASF AG
Michael Halloran	then with AspenTech
Daniel Piñol	then with Hyprotech S.L.
Michel Pons	CO-LaN and previously TOTAL
Richard Szczepanski	Infochem Computer Services Ltd.
Jasper van Baten	AmsterCHEM

The CO-LaN Thermodynamics Special Interest Group has been active in reviewing and improving the specification and has provided a great deal of valuable input.

CONTENTS

LIST OF FIGURES

1. Introduction

This interface specification is a complete revision of the previous CAPE-OPEN Thermodynamic and Physical Properties interface specification (version 1.0) described in reference [1]. The functionality covered by the interfaces is slightly extended, but the main motivations for the changes are simplification, increased flexibility and the removal of ambiguities. These are the issues that emerged most strongly from testing implementations of the original specification. This document is designed to make it easier to understand the interfaces, and therefore to implement CAPE-OPEN components, and to make inter-operability of CAPE-OPEN components more easily achievable.

Compared to previous versions of the specification, this one has more interfaces but fewer methods. The methods are logically grouped in interfaces to remove duplication and the result is a cleaner and more abstract design.

The new functionality covered by the specification falls into two areas. Firstly, the interfaces and methods related to equilibrium calculations are expanded to accommodate specialist equilibrium calculation tools, whose capabilities go beyond those of typical commercial simulators. Secondly, the description of phases is extended to provide better support for multi-phase calculations.

The latest version of this document (version 3) is a revision of the previously published version 1.1 specification [11]. There are no changes to the formal definition of the interfaces or methods. However, experience with the increasing number of implementations has shown the need for clarifications of the expected behaviour of the software components described. In response, the CO-LaN Thermodynamics Special Interest Group has produced an ‘Errata and clarifications’ document [12] which is now incorporated in this updated text.

The principal changes in the document are listed below.

- Addition of mole fraction derivatives.
- New property identifiers enthalpyF, enthalpyNF, entropyF, entropyNF.
- New universal constant: idealGasStateReferencePressure.
- Removed requirement for SetPresentPhases to make a phase ‘present’ before any properties are set.
- A Material Object must implement the ICapeThermoPhases interface.
- Clarification of intended use of SetPresentPhases and GetPresentPhases methods.
- CheckEquilibriumSpec requires a prior SetPresentPhases call.
- Clarification of chemicalFormula property format.
- Expanded definition of “Normal” and “Retrograde” in CalcEquilibrium method.
- Clarification of SetMaterial method usage.
- Clarification of meaning and usage of Overall phase properties (section 7.5.7).
- New section on Implementation of the Persistence Interface (section 8).

2. Audience

This document is intended primarily for software developers who want to build CAPE-OPEN Property Package, Equilibrium Calculator and Physical Property Calculator components. It is also intended for the developers of other software components, such as Unit Operations and Reaction Packages, which make use of these Thermodynamic and Physical Properties software components. Finally, it is intended for developers of CAPE-OPEN-compliant Process Modelling Environments because it describes how the interfaces are to be used to implement communication between the environment and the external software components.

Designers of other CAPE-OPEN interface specifications should read this document to ensure consistency across the various designs.

For any reader an understanding of UML diagrams is assumed.

This document is not intended for end-users of CAPE-OPEN software components or process simulation software.

3. Glossary

Material
A Material is a mixture of one or more Compounds occurring in one or more Phases. A Material is characterised by the values of Physical Properties, which can describe the overall Material or the Compounds within particular Phases. A Material often corresponds to a stream in conventional process simulation software.

Material Object
A Material Object is a software object that implements the interfaces and behaviour of a Material as defined in this interface specification.

Equilibrium Calculation
A calculation that determines the composition and amounts of each Phase of a Material subject to specified constraints such as, for example, a particular temperature and pressure.

Physical Property
A Physical Property is an attribute used to characterise a Material. Typical examples are Temperature and Pressure. A Material is described by the values of some subset of the set of possible Physical Properties defined within section 7.5 of this document. Note that at this revision of the interface specification, the Physical Properties used to describe the unique characteristics of polymers, and petroleum fractions are not included in the set of possible Physical Properties. These will be added later as the interface specification is expanded to cover these kinds of Material.

The subset of Physical Properties used to describe a Material is not fixed: it will depend on the aspect of the Material behaviour being studied.

Within simulation software, a Physical Property is usually calculated by a model selected from some set of available models for that Physical Property.

Physical Property Calculation
A Physical Property Calculation is a calculation that determines the value of a Physical Property in a Material, given temperature, pressure and composition.

Compound
A chemical substance as defined by a particular set of Physical Properties, calculation methods and data. Compounds can be identified in various ways: by a common name, by a CAS Registry number or by a chemical formula. Examples are water, hydrogen and oxygen.

Component
A piece of executable software whose functionality is accessed via specified interfaces, which can be deployed independently of other software and which can be used in the composition of other software systems without modification. For the sake of clarity it is often associated with the term software.

Phase
A Phase is a stable or metastable collection of Compounds with a defined amount of substance and a homogeneous composition. It has an associated State of Aggregation, *e.g.* liquid. A given Phase can be distinguished from others through the presence of physical interfaces that separate states of matter with different characteristics, such as density.

In a Material, each Phase is given a unique identifier. In order to distinguish multiple instances of similar Phases, such as liquids, the Phase can be associated with a number of attributes such as a 'key compound'.

Universal Constant
A Universal Constant is any well-known physical constant, such as the Avogadro constant.

State of Aggregation
The State of Aggregation of a Phase is the physical state in which the Compounds in that Phase occur. Possible values for State of Aggregation are Vapor, Liquid, Solid or Unknown. A PME is expected to use State of Aggregation to help distinguish one Phase of a Material from another.

COSE
COSE is an acronym for the term CAPE-OPEN Simulation Environment. It refers to any software that makes use of CAPE-OPEN components.

PMC
PMC is an acronym for Process Modelling Component. A PMC is a software component used to provide specific functionality within a PME. Examples are Unit Operations, Numerical Solvers, and Property Packages.

PME
PME is an acronym for Process Modelling environment. PME is synonymous with COSE.

Physical Properties System
A Physical Properties System is a proprietary software system that includes a Physical Properties Executive, a set of Physical Property routines and access to data for a number of chemical Compounds. A Physical Properties System is likely to include text information that the user can access to help select the most appropriate properties, methods and data for the particular application. It will often access a large Physical Properties data bank, using different interfaces from the ones described in this document. For example, it may use the CAPE-OPEN Physical Properties Data Base (PPDB) interfaces, which allow bulk access for Thermodynamic and Physical Properties databases.

Physical Properties Executive
A Physical Properties Executive is the part of a Physical Properties System that provides the user interface by which the methods, data and Compounds can be selected. It also organises the computation so that, in calculating Material properties, the correct methods are employed for the specific Material conditions. The Physical Properties Executive may provide access

to additional services, such as the ability to correlate raw data to generate parameters for selected methods.

Property Package
A Property Package is a complete, consistent, reusable, ready-to-use collection of methods, Compounds and model parameters for calculating any of a set of known Physical Properties for the Phases of a Material. It includes all the pure Compound methods and data, together with the relevant mixing rules and interaction parameters. A Property Package normally covers only a small subset of the Compounds and methods accessible through a Physical Properties System.

4. Scope

This section describes Thermodynamic and Physical Properties software components, the interfaces they support and the relationship between the different software components.

The standard defines the following primary CAPE-OPEN components:

- Physical Property Calculator – a software component that can calculate certain Physical Properties, possibly restricted to mixtures of particular Compounds existing in particular Phases.
- Equilibrium Calculator – a software component that can calculate the composition of non-reacting mixtures at equilibrium, possibly restricted to mixtures of particular Compounds existing in particular Phases and subject to certain constraints.
- Property Package – a software component that combines the functions of a Property Calculator and an Equilibrium Calculator for a fixed set of Compounds and Phases. A Property Package will provide compound constants and may also provide universal constants.
- Property Package Manager – a software component that manages a set of Property Packages. It is responsible for instantiating Property Packages on request and may allow Property Packages to be edited and/or created.

This document describes the following interfaces:

- ICapeThermoEquilibriumRoutine – methods implemented by components which can perform an Equilibrium Calculation.
- ICapeThermoPropertyRoutine – methods implemented by components that can calculate values for Physical Properties.
- ICapeThermoUniversalConstants – methods implemented by components that can supply the values of Universal Constants.
- ICapeThermoPhases – methods implemented by components that need to describe the Phases that are present or could be present in a Material.
- ICapeThermoCompounds – methods implemented by components that need to describe the Compounds that occur or can occur in a Material.
- ICapeThermoMaterial – methods implemented by components that need to provide access to the Physical Properties of a particular Material.
- ICapeThermoMaterialContext – methods implemented by components that need to be given a particular Material as the context for a calculation.

- ICapeThermoPropertyPackageManager – methods implemented by a component that can list and create instances of available Property Package components.

These interfaces are functional: they collect together the methods required to perform particular types of calculation and data access. The software components defined in this interface specification will support all the interfaces required to provide the functionality they implement. There is not a one-to-one correspondence between particular interfaces and particular software components. For example, a software component that implements the ICapeThermoEquilibriumRoutine interface may be an Equilibrium Calculator component or a Property Package component.

The interfaces allow clients of the four above defined types of components to communicate with them independently of the component implementations. Communication between a client and any type of thermodynamic software component requires the exchange of data describing the Material for which calculations are required. The concept of a Material Object is described for this purpose. A Material Object implements CAPE-OPEN interfaces but it is not a CAPE-OPEN component. A Material Object is a software object that is responsible for holding the data describing the state of a Material. Each client that uses CAPE-OPEN Thermodynamic and Physical Properties components must provide its own implementation of a Material Object because the configuration and data storage used in a Material Object will be different for each client.

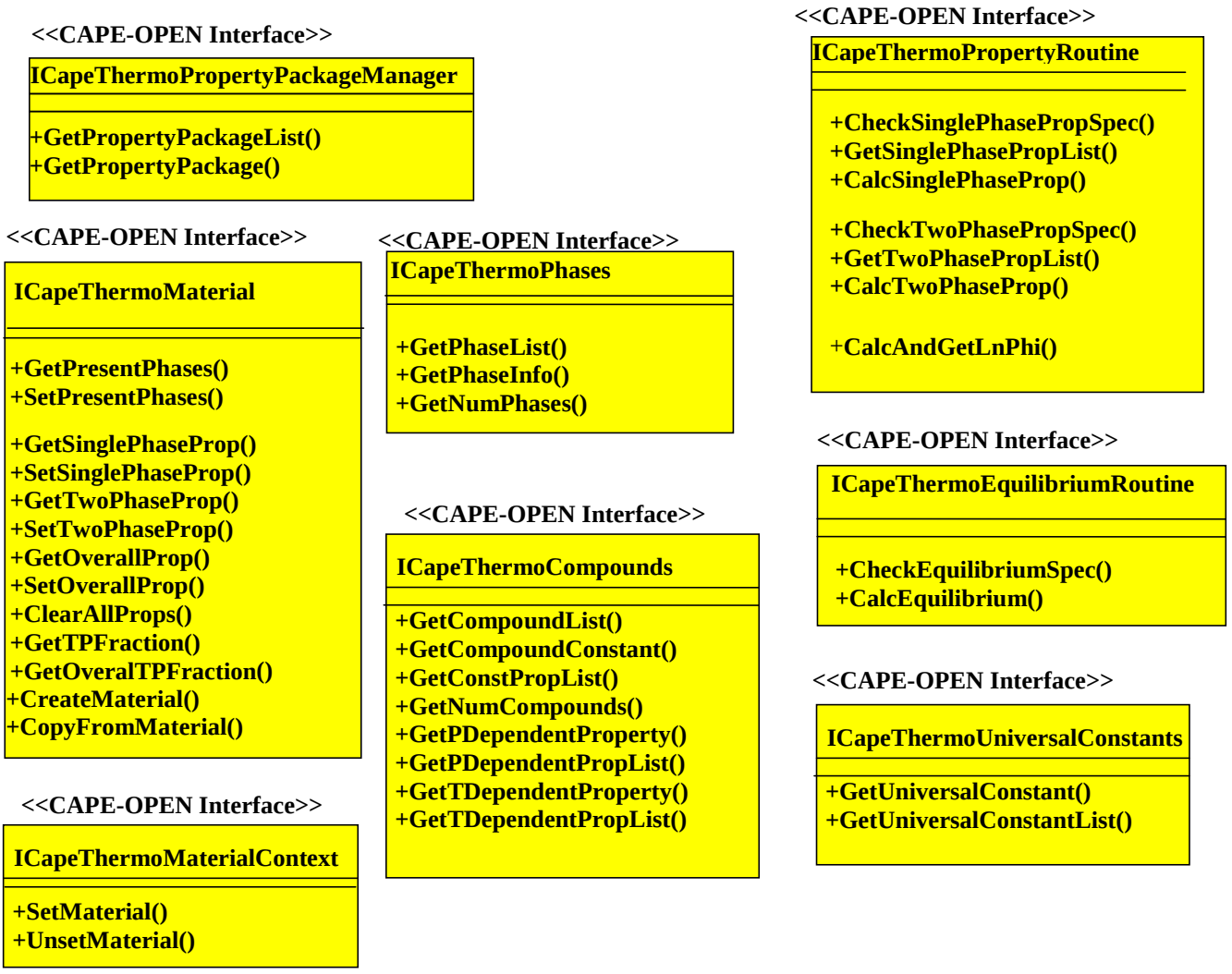


Figure 1: Interface diagram

This document does not cover the Physical Properties and interfaces required to implement support for complex materials such as solids, petroleum fractions, and polymers. Nor does it cover the interfaces required to describe a system of reactions within a Material. Each of these topics is covered in a separate document [3], [7].

This specification document does not provide detailed information on how to implement the software components it describes. Information and examples that support implementations will be available from the CO-LaN website: www.colan.org.

5. Conceptual Object Model

Before describing the detail of the interfaces, a Conceptual Object Model of the interaction between a PME and an external CAPE-OPEN Thermodynamic and Physical Properties component is required. This Conceptual Object Model is not concerned with how the components are created, configured, selected or associated with the PME, its purpose is to show what kinds of software components are required and to show how a PME and a component interact. Much of this material will already be familiar to anyone who knows the earlier versions of the CAPE-OPEN standard.

An important concept in the Conceptual Object Model is that of a Material. In earlier versions of this specification, Material was a central concept and, in fact, the way in which a Material is characterised, in terms of a set of Compounds and the values for Physical Properties, is still the same here. The kinds of Material that can be represented are also unchanged. However, in order to develop a more component-oriented model to justify the interface specifications described here, the role of Material is given less emphasis. Within this specification, the term Material Object refers to the software object that implements the representation of a Material.

Three types of software component are considered in the Conceptual Object Model. Each type of software component is meant to provide an aggregation of functionalities:

- Physical Property Calculator – a software component, which, given the temperature, pressure and composition of a Phase of a Material, is able to calculate an additional Physical Property or Physical Properties of the Material. Again, a Physical Property Calculator will be designed to work with certain kinds of Material. Note that a Physical Property Calculator is not called directly by a PME; rather, it is called via a CAPE-OPEN Property Package. The purpose of a Physical Property Calculator is to extend or to override the list of calculations that a CAPE-OPEN Physical Property Package can perform. A CAPE-OPEN Property Calculator can only be used with a Property Package which supports the use of Property Calculators.
- Equilibrium Calculator – a software component, which, given a description of a Material, and a specification of constraints on the calculations such as temperature and pressure, can calculate the composition of each Phase present in the Material. An Equilibrium Calculator will typically be designed to work with certain types of Material. That is, it will provide the Phase compositions for Materials containing a subset of the set of Compounds known to the Equilibrium Calculator, where those Compounds occur in Phases known to the Equilibrium Calculator. For example, a specific Equilibrium Calculator may not be able to deal with Materials with solid Phases, or those containing polymeric Compounds. Note that an Equilibrium Calculator is not called directly by a PME. Like a Physical Property Calculator, it is only called via a Property Package. Like a Property Calculator, the purpose of an Equilibrium Calculator is to extend or to override the list of Equilibrium Calculations that a CAPE-OPEN Property Package can perform. A CAPE-OPEN Equilibrium Calculator can only be used with a Property Package which supports the use of Equilibrium Calculators.
- Property Package – a software component that is both a Physical Property Calculator and an Equilibrium Calculator for Materials containing a specific set of Compounds occurring in a specific number of physical states. A Property Package will make use of certain models to perform these calculations. A Property Package can be configured to make use of external Physical Property Calculators and/or Equilibrium Calculators. Alternatively, it can provide the functionality of these two components internally without making use of external components. Configuring the Compounds, Phases, models and external components used in a Property Package is outside the scope of this CAPE-OPEN interface specification.

For the purposes of this Conceptual Object Model, a PME is any piece of software that needs to use the functionality of the immediately above listed three kinds of component. For example the PME could be Microsoft Excel, or it could be a Simulation Environment, or it could be a Unit Operation executing within a Simulation Environment.

To put the use of these components in context, the object diagram below shows a simplified representation of the kinds of objects found in proprietary Simulation Environments and some of the relationships that exist between them. A flowsheet object is associated with a number of streams and with a number of Unit Operations. Each Unit Operation and stream has access to a Thermodynamics sub-system that provides Physical Property Calculations using various Physical Property models, equilibrium, or flash, calculations; and constant Physical Property data. The flowsheet object also has access to the Thermodynamics sub-system. There are various relationships between the streams and the streams are used to connect blocks, but blocks may also contain streams.

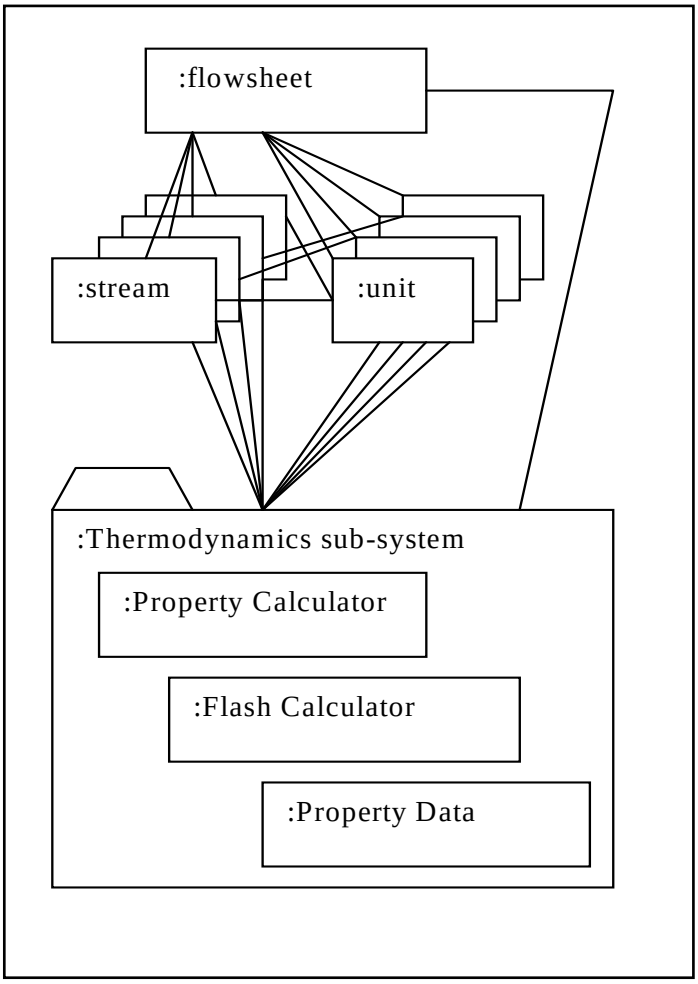


Figure 2: Object diagram on proprietary implementations

The connection link between a stream and a Unit Operation exists because a Unit Operation must be given a description of the Material at its inlets so that it can calculate a description of the Material at its outlets.

The ownership link (which is not distinguished from a connection link on the diagram) between a Unit Operation and a stream exists because a Unit Operation may need to create a representation of a Material in order to perform an internal calculation.

The Thermodynamics sub-system provides three main services: a way of calculating the value of a non-constant Physical Property, a way of calculating equilibrium and, thirdly, access to a database of constant Physical Property data. The Thermodynamics sub-system does not know about Unit Operations and streams; when asked to perform a calculation it is given a description of a Material as input. Equally, the Thermodynamics subsystem is not

responsible for maintaining the description of any Material in a problem; it simply performs the requested calculation given the description of a particular Material and returns the result.

The CAPE-OPEN standard is intended to allow proprietary objects used within a Simulation or other Process Modelling Environment to be replaced with external CAPE-OPEN components that implement the same functionality. The next diagram shows how proprietary "socket" objects, which allow communication with the external CAPE-OPEN components, can replace parts of the proprietary Thermodynamics sub-system and Unit Operations.

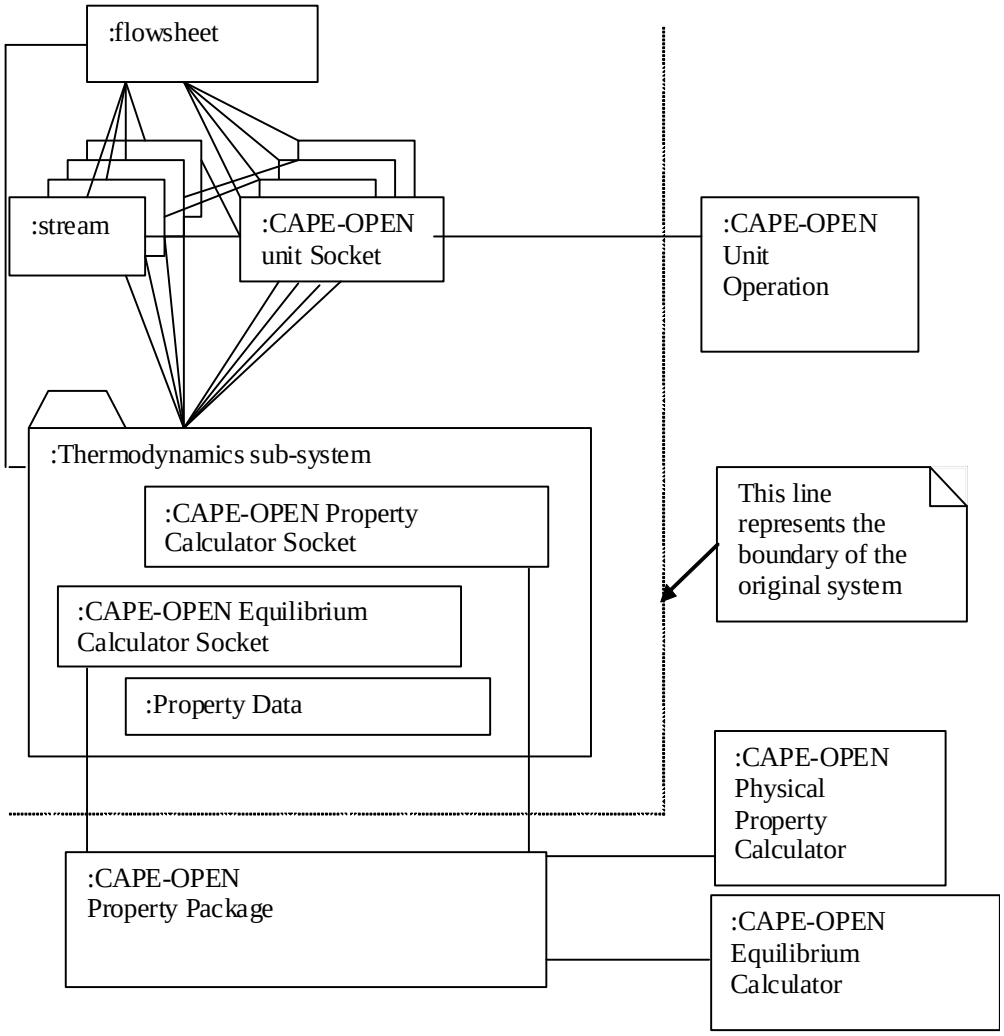


Figure 3: Communication between sockets and plugs

In order to make use of external software components, standard interfaces to each of the three types of CAPE-OPEN Thermodynamic and Physical Properties components are required. When such components are being used in a PME, their functionality must be available to proprietary objects within the PME, but also to any other external components being used by the PME. One example of the importance of this requirement is the need for consistent Physical Property Calculation routines to be used for both built-in Unit Operations and CAPE-OPEN Unit Operations, whether the PME is using external Physical Property Calculations, or its own internal calculation routines.

In this diagram, the proprietary Thermodynamics subsystem contains proprietary sockets, which are software objects that allow communication with external CAPE-OPEN components. This is an example of the "Adapter" design pattern [6]. Existing proprietary

objects such as streams and Unit Operations still use the internal proprietary interfaces implemented by the Thermodynamics subsystem. The stream and Unit Operation objects therefore do not need to be changed. Internally, the Property Calculator and Equilibrium Calculator sockets translate any in-coming call to a call to an external CAPE-OPEN object using the standard interfaces that CAPE-OPEN defines. The diagram also shows how a Property Package, can itself make use of the CAPE-OPEN interfaces to call CAPE-OPEN components implementing the Physical Property Calculator and Equilibrium Calculator interfaces.

Note that in this diagram the Thermodynamics sub-system still provides the functionality of a Property database, that is, it is responsible for the list of compounds that the PME can use, the Phases that can be present in Materials and the values for constant Physical Properties. The Property database may be internal to the Thermo sub-system or provided in some other way for example through a CAPE-OPEN PPDB socket.

The same pattern is used for Unit Operations: a proprietary adapter, called “CAPE-OPEN Unit Socket” in the diagram, is used to map calls made by the PME through its proprietary unit interface to objects implementing the CAPE-OPEN unit interfaces.

5.1 The Description of Material

The description of a Material is both an input to and an output of all the components shown in the diagrams. In a sequential modular simulator, a Unit Operation requires a description of the Material at each of its material inlet ports so that it can calculate a description of the Material at each of its material outlet ports. An Equilibrium Calculator component, regardless of whether it is a Property Package or an independent Equilibrium Calculator, takes the description of a Material and updates it with the composition and amounts of each Phase subject to specified constraints. Similarly, a Physical Property Calculator component takes the composition of a Material and its temperature and pressure and calculates the value of additional Physical Properties.

All PMEs will have their own proprietary representation of Material. For example, a Simulation Environment might use an instance of a C++ class to hold the data describing the state of a Material, or, in a spreadsheet, a particular arrangement of cells might be used. Whatever representation is used, this data has to be presented to CAPE-OPEN Thermodynamic and Physical Properties components and Unit Operation components in a standard way. As well as using the description of Material as input, both Thermodynamic and Physical Properties components and Unit Operation components need to be able to update the description as an output of their calculations. Providing access to the data describing a Material via CAPE-OPEN interfaces is the responsibility of the Material Object.

The CAPE-OPEN standard defines a list of Physical Properties that can be used to describe a Material. The list is presented in section 7.5 of this document. In addition to the values of Physical Properties, a set of Compounds and a list of the Phases present also characterise a Material. Some Physical Properties describe the overall state of a Material; others characterise individual Compounds or Phases within it.

Earlier, the adapter design pattern was used to show how a PME could access external software components. The same design pattern can be used to implement a Material Object. As in the other examples of adapters, the Material Object is implemented by the PME. It presents the CAPE-OPEN interfaces for accessing the description of a Material to the

external components. Internally, it does what ever is required to access the data held in the PME's proprietary data structures.

5.2 Material Object responsibilities

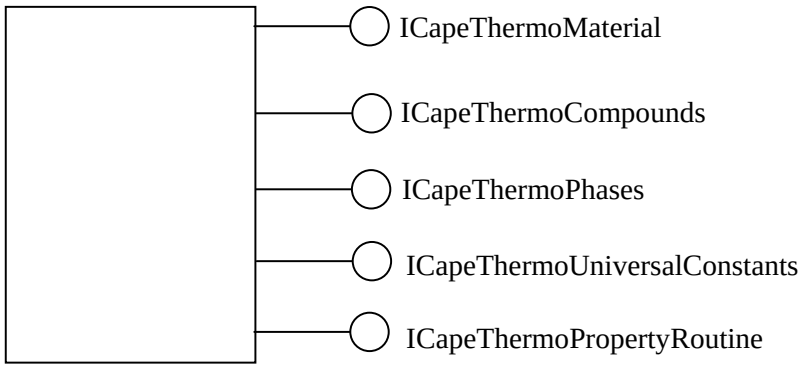
The primary interface implemented by a Material Object is the ICapeThermoMaterial interface. Typically, a Material Object will be designed to access a proprietary data structure belonging to a PME, and it will itself be proprietary to a PME. The interfaces that a Material Object needs to support vary depending on which type of CAPE-OPEN Thermodynamic and Physical Properties component is using the Material Object. The following sections describe which interfaces a Material Object implementation must support for each of the different types of component. A single Material Object implementation can be used in all these circumstances as long as it implements all the interfaces that could be used.

Note that in the following object and component diagrams, objects and components are shown as supporting multiple interfaces. The implication is that if a client is passed one of the supported interfaces, it also has access to all the others since each Material Object is passed as a CapeInterface (IDispatch in COM). The diagrams do not show the ICapeIdentification [4] and Error Interfaces [5] that should be supported by all CAPE-OPEN components.

5.2.1 Interfaces used by a Physical Property Calculator to access Material Objects

A CAPE-OPEN Physical Property Calculator needs to be able to get the values of the Physical Properties from a Material Object and to set the values that it calculates. To a Physical Property Calculator, a Material Object must behave as a store of data, but must also provide access to Compound constants and Physical Property Calculations. The reason for these requirements is to support Physical Property Calculator components that need supplementary data from a Physical Property Package in order to complete their own calculations.

Therefore, to satisfy the requirements of CAPE-OPEN Physical Property Calculators, a Material Object must implement the following interfaces:



The ICapeThermoMaterial interface is required to provide access to the values of the Physical Properties that describe the Material.

The ICapeThermoCompounds interface is required so that the caller can: find out what Compounds are present in the Material; check that they can be recognised; and, request the values of Compound constants.

The ICapeThermoPhases interface is required so that the caller can: find out the Phases that are present in the Material in order to set properties and perform property calculations.

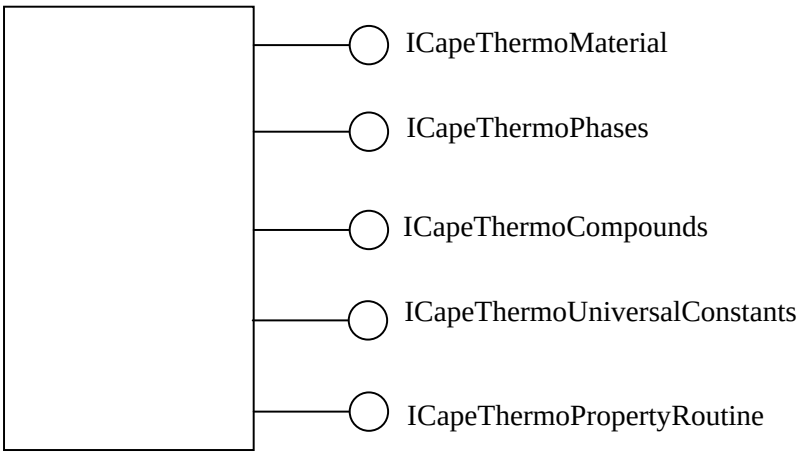
The ICapeThermoUniversalConstants interface is optional but may be implemented to provide the caller with the values of Universal Constants. Note that a Physical Property Calculator that requires the values of Universal Constants will not work in a Property Package that provides a Material Object that does not implement this interface. The Material Object may simply forward a request for universal constants to a Property Package or provide constants through some proprietary mechanism, *e.g.* from a PME.

The ICapeThermoPropertyRoutine interface is optional but may be implemented to provide the caller with access to Physical Property Calculations. As with the ICapeThermo-UniversalConstants interface, if the ICapeThermoPropertyRoutine interface is not supported, a Physical Property Calculator component may not work.

Note that when a Property Package calls a Physical Property Calculator it can provide its own Material Object or it can pass the Material Object passed to it by the PME. However, a Material Object provided by a PME may not support all the properties required by the Property calculator.

5.2.2 Interfaces used by a Equilibrium Calculator to access Material Objects

A CAPE-OPEN Equilibrium Calculator component requires the same Material interfaces as a Physical Property Calculator component because it needs to get values of Physical Properties from a Material and it needs to set the values of the Physical Properties that it calculates. Unlike a Physical Property Calculator, it will determine which Phases are present in the Material and may therefore update the Material's list of Phases. As part of performing this calculation, it may need to call Physical Property Calculations. Performing Physical Property Calculations requires the interfaces that define a Physical Property Calculator component:



For an Equilibrium Calculator component, a Material has to behave both as a store of data and as a Physical Property Calculator. As described earlier, a Material Object is expected to implement the ICapeThermoPropertyRoutine methods by forwarding all the calculation calls back to its creator, which could be the PME or the Property Package that is calling the Equilibrium Calculator component.

Note that when a Property Package calls an Equilibrium Calculator it can provide its own Material Object or it can pass the Material Object passed to it by the PME.

5.2.3 Interfaces used by a Unit Operation to access Material Objects

To satisfy the requirements of CAPE-OPEN Unit Operation components, a more complex Material Object implementation is required. This is because a Unit Operation needs to get and set the values of Physical Properties in a Material and it also needs to perform both Physical Property and Equilibrium Calculations. Therefore, in order to be used by Unit Operation components, a Material Object must implement the interfaces defining a Property Package.

The diagram below shows a software object, rather than a software component, that presents the interfaces required to describe a Material and the interfaces required to perform Physical Property and Equilibrium Calculations. The object is shown as requiring the same Physical Property and Equilibrium calculation interfaces as it presents on the assumption that it simply forwards the calculation calls to another component, passing itself as the context for the calculation. This is only one of several possible arrangements. In practice, the software object may call a proprietary interface to perform the actual calculation. The implementation of the proprietary interface may then construct a second Material description to pass to an external component for a calculation.

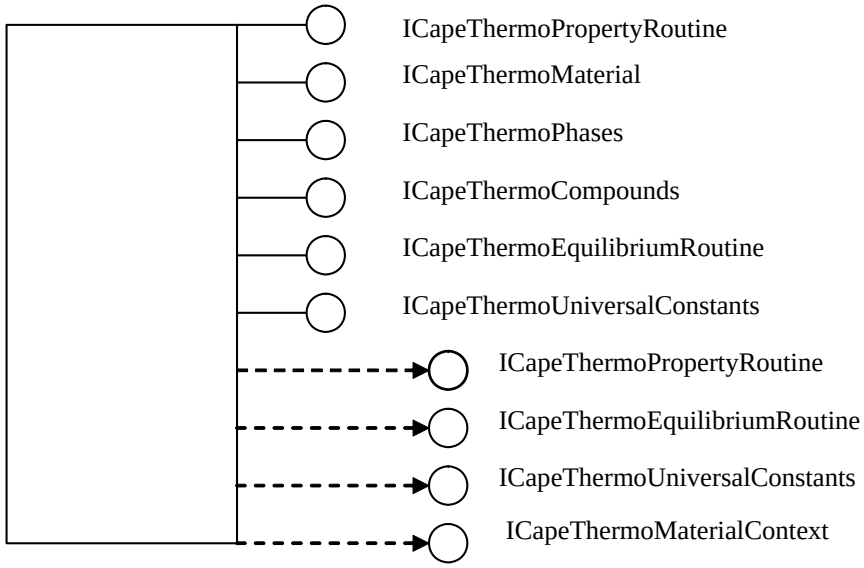


Figure 4: Interfaces required on a software object

This design is required by the CAPE-OPEN Unit Operation specification because it does not provide a mechanism for a PME to tell a Unit Operation which Property Calculator and which Equilibrium Calculator to use for a specific inlet stream. Instead, it only allows the PME to associate an object implementing the ICapeThermoMaterial interface with a Port. As a result, an object that aggregates the Material interfaces and the calculation interfaces is required.

Similarly, for a Property Package component, the only context that a PME can provide is an object implementing the ICapeThermoMaterial interface. Since a Physical Property Calculator needs to use other interfaces, the object passed to it must implement those interfaces as well.

Note that a Material Object that meets the requirements of a CAPE-OPEN Unit Operation will meet the requirements of all Physical Property components described in this document.

This design has advantages and disadvantages. It complicates the implementation of objects that support the ICapeThermoMaterial interface. But the extra complication has to be balanced against the fact that the design is extensible. In future versions of the standard, Property Packages and Unit Operations will be able to access other data and functionality supplied by the PME, simply by being passed Material Objects which implement the new interfaces.

5.2.4 Material Object behaviour

So far, this section has described the interfaces that Material Objects need to implement to meet the requirements of CAPE-OPEN components. As well as supporting the required interfaces, a Material Object has to exhibit specific behaviour if it is to be CAPE-OPEN compliant. These behavioural specifications are intended to ensure the consistency of the data available to a client through the CAPE-OPEN interfaces.

A Material Object may optionally support (ie. allow a client to set/retrieve property values) many Physical Properties but it is a requirement that the following properties are supported: temperature, pressure, fraction and phaseFraction. In addition a Material Object that is passed to a Unit Operation component must support the flow and totalFlow properties. The property identifiers are defined in section 7.5.

A Material Object client may get or set any basis-dependent Physical Property of a Material using any basis. It is the Material Object's responsibility to ensure that the client sees consistent values whatever basis is used. This means that the Material Object must:

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

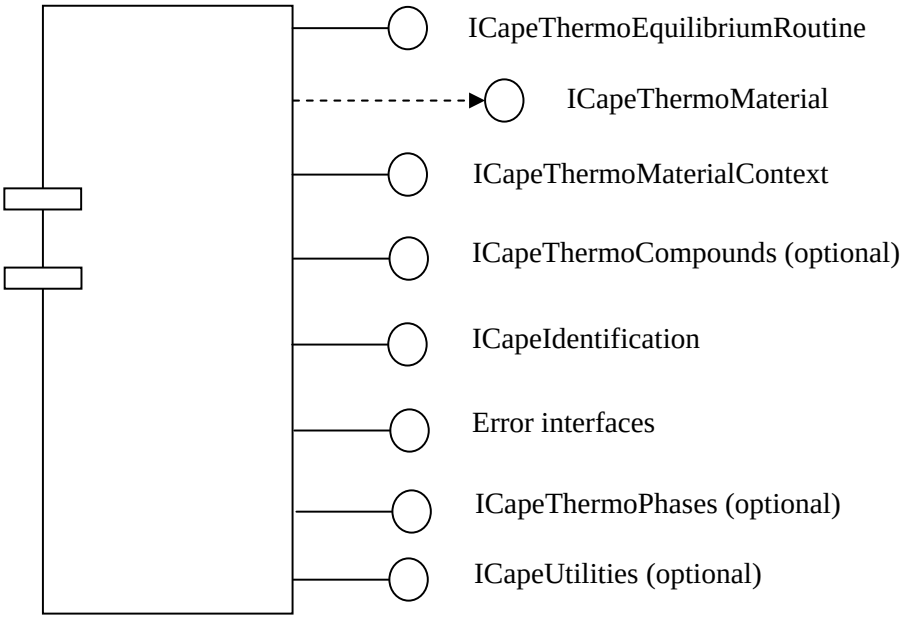
As discussed in earlier sections of this document, a software object implementing the CAPE-OPEN Material interfaces can be implemented in a number of ways. These are some of the alternatives:

- Implement the object as a store, or cache of data and exchange that data with both the PME and external CAPE-OPEN components.
- Implement the object using the façade design pattern [6]. The resulting object does not store any data, but it maintains a reference to a PME data structure that does store the data.

The choice of implementation depends on the requirements of a particular PME, so this document does not make a recommendation.

5.3 Equilibrium Calculator component responsibilities

An Equilibrium Calculator component must implement the ICapeThermoEquilibrium-Routine and ICapeThermoMaterialContext interfaces. In turn it uses the ICapeThermo-Material interface passed to it via the ICapeThermoMaterialContext::SetMaterial interface to access the description of the Material being worked on from a Material Object and to update the Material's properties. The ICapeIdentification [4] and Error Interfaces [5] must be implemented by all CAPE-OPEN components.



An Equilibrium Calculator whose calculations are restricted to particular Compounds must implement support for the ICapeThermoCompounds interface so that any Property Package that uses it can check consistency. An Equilibrium Calculator that does not implement the ICapeThermoCompounds interface is assumed to be able to perform its calculations for any Compound.

Similarly, an Equilibrium Calculator whose calculations are restricted to particular Phases must implement support for the ICapeThermoPhases interface for the same reason of consistency. An Equilibrium Calculator that does not implement the ICapeThermoPhases interface is assumed to be able to perform its calculations for any Phase. Typically, an Equilibrium Calculator will need to implement ICapeThermoPhases.

The ICapeUtilities interface [17] may be implemented if the Equilibrium Calculator requires to make use of facilities provided by this interface, for example, the Initialize method.

The following conceptual sequence diagram shows the expected interaction between a PME, a Property Package, an Equilibrium Calculator component and a Material Object when an Equilibrium Calculation is performed using an Equilibrium Calculator component. This sequence diagram is a sketch, so some operations are abbreviated for simplicity. Note that the diagram does not use actual method names. The names used are intended to indicate the nature of the call being made.

The diagram shows how an implementation of a Material Object is required in order to pass data from the PME to the Property Package and from the Property Package to the external Equilibrium Calculator component. The Equilibrium Calculator stores its results using the Material Object. The diagram does not show the interaction between the Equilibrium

Calculator and the Material Object during the CalcEquilibrium call, but typically the Equilibrium Calculator will call back to the Material Object for Property data, Property Calculations and compound data.

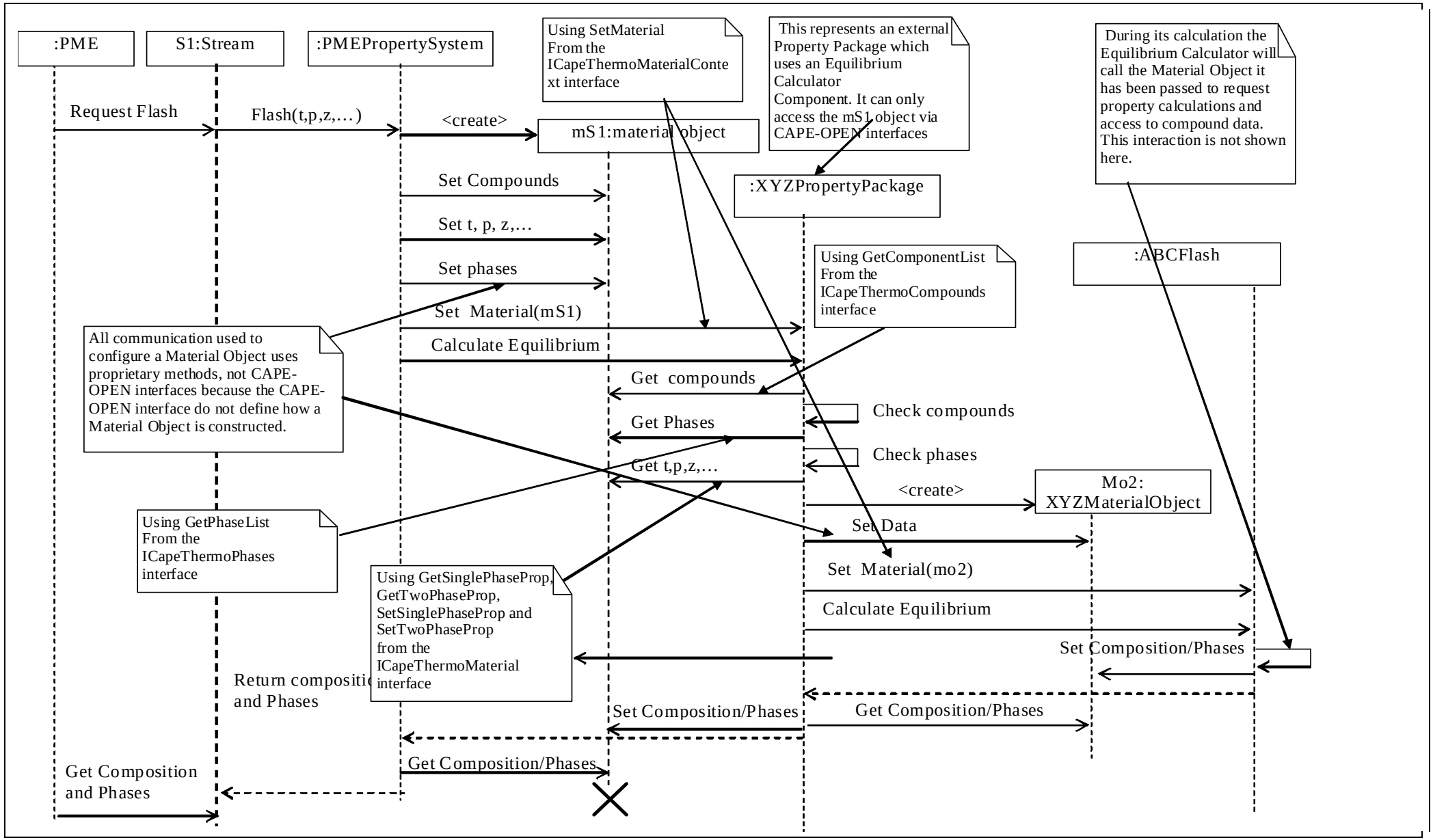


Figure 5: Conceptual Sequence Diagram

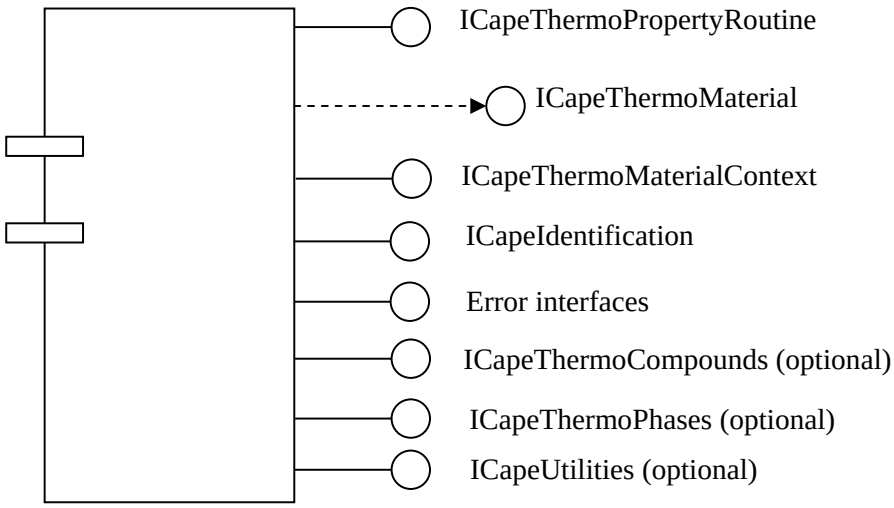
5.4 Equilibrium Calculator behaviour

An Equilibrium Calculator component is expected to compute the amounts (phase fractions) and compositions of all Phases at equilibrium and it can be used to calculate overall Temperature and Pressure if these are not specified. It must set Temperature and Pressure for all Phases present at equilibrium as well as for "Overall" if not already specified. A phase may be present in zero amount, for example the liquid phase in a dew point calculation. The calculation should not update or set any other Physical Property in the Material Object it has been passed. In particular it must not set any properties for phases that are not present. If it requests a Physical Property Calculation as part of executing the Equilibrium Calculation, the resulting value must not be stored in the Material Object.

The reason for insisting on no side effects is that clients may come to rely on them, even though they are not part of the CAPE-OPEN standards.

5.5 Physical Property Calculator component responsibilities

A Physical Property Calculator component must implement the ICapeThermo-PropertyRoutine and ICapeThermoMaterialContext interfaces. In turn it uses the ICapeThermoMaterial interface passed to it via its ICapeThermoMaterialContext SetMaterial method to access the data describing the Material being worked on from a Material Object and to update the Material's properties. The ICapeIdentification [4] and Error Interfaces [5] must be implemented by all CAPE-OPEN components.



A Physical Property Calculator whose calculations are restricted to particular Compounds must implement support for the ICapeThermoCompounds interface so that any Property Package that uses it can check consistency. A Property Calculator that does not implement the ICapeThermoCompounds interface is assumed to be able to perform its calculations for any Compound.

Similarly, a Property Calculator whose calculations are restricted to particular Phases must implement support for the ICapeThermoPhases interface for the same reason of consistency. A Property Calculator that does not implement the ICapeThermoPhases interface is assumed to be able to perform its calculations for any Phase. Typically, a Property Calculator will need to implement support for ICapeThermoPhases.

The ICapeUtilities interface [17] may be implemented if the Property Calculator requires to make use of facilities provided by this interface, for example, the Initialize method.

The following conceptual sequence diagram shows the expected interaction between a PME, a Property Package, a Physical Property Calculator component and a Material Object when a Physical Property Calculation is performed. As before, this sequence diagram is a sketch, so some operations are abbreviated for simplicity. As before, note that the diagram does not use actual method names. The names used are intended to indicate the nature of the call being made.

The pattern of calls used for the Equilibrium Calculation is used again here. The PME Material Object acts as an intermediary between the PME and the Physical Property Calculator component. A second Material Object constructed by the Physical Property Package act as intermediary between the Property Package and the Property Calculator. According to this diagram, the lifetime of a Material Objects is very short; it only exists for as long as the object that creates it needs to communicate with the external component. In general, the creator of a Material Object will determine how long it needs to exist.

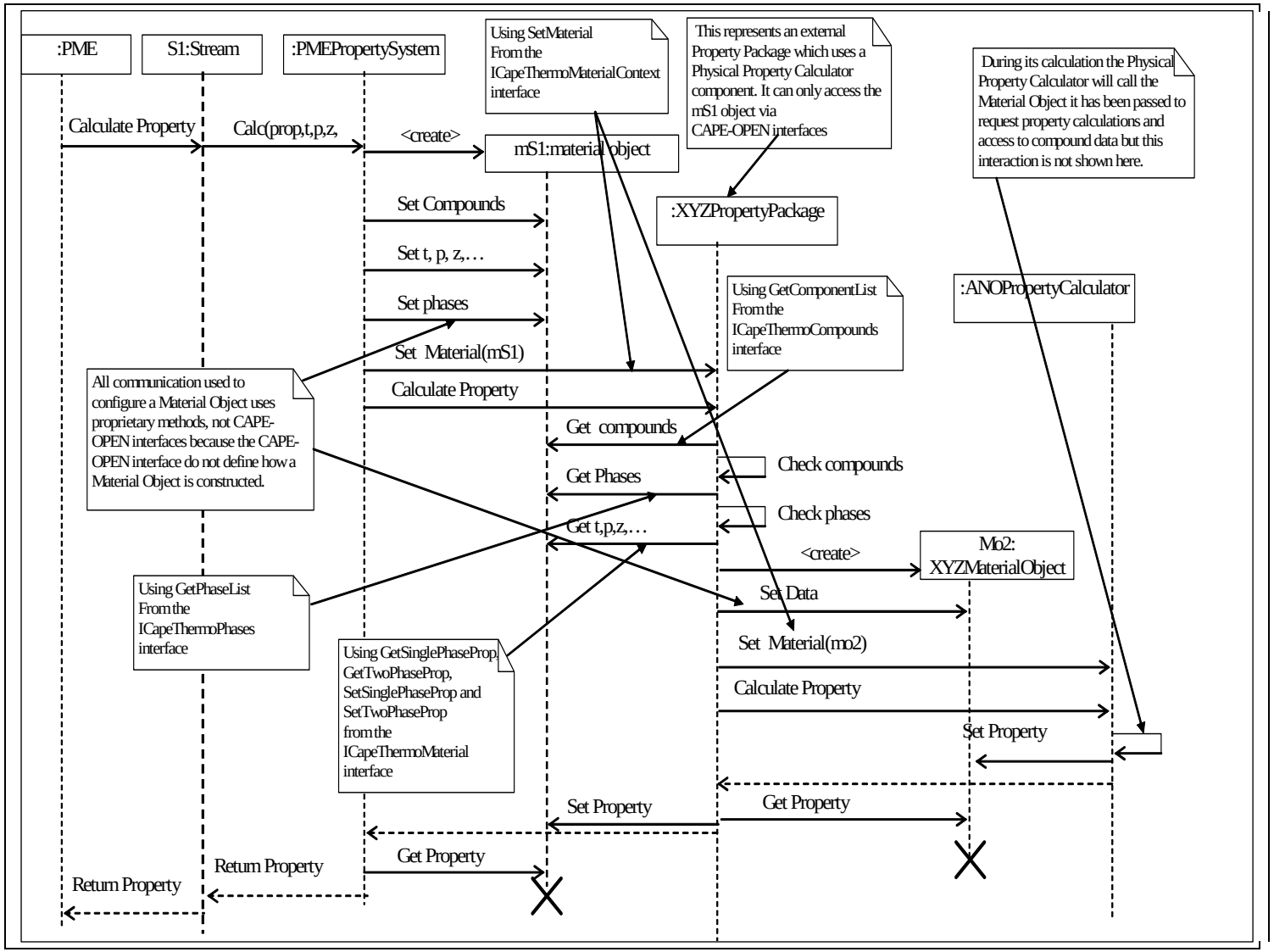


Figure 6: Conceptual Sequence Diagram

5.6 Physical Property Calculator behaviour

A Physical Property Calculator must calculate the requested Physical Property without any visible side effects on the Material Object it is passed. If the calculation it performs requires other Physical Property calculations, the results of these calculations must not be stored in the Material Object.

The reason for insisting on no side effects is that clients may come to rely on them even though they are not part of the standard.

A Property Calculator would typically obtain the values of any compound properties it requires using the Material Object supplied via the ICapeThermoMaterialContext interface. However, this behaviour cannot be used to change the values of physical properties of compounds (for example pseudocomponents) dynamically. This is because there is no way to ensure that a property calculator will reset its internal data structures. The properties of petroleum fractions can however be changed using the Petroleum Fractions Interface [7].

Each Physical Property Calculator or Property Package may use a different base reference for Enthalpy (H) or Entropy (S) calculation. It is impossible to anticipate all the different possible conventions that might be used by a PME and a Physical Property Calculator/ Property Package, so an automatic procedure is preferable. A PME, or other client of a property package, can adjust for any difference in h and s zeros by the following simple procedure:

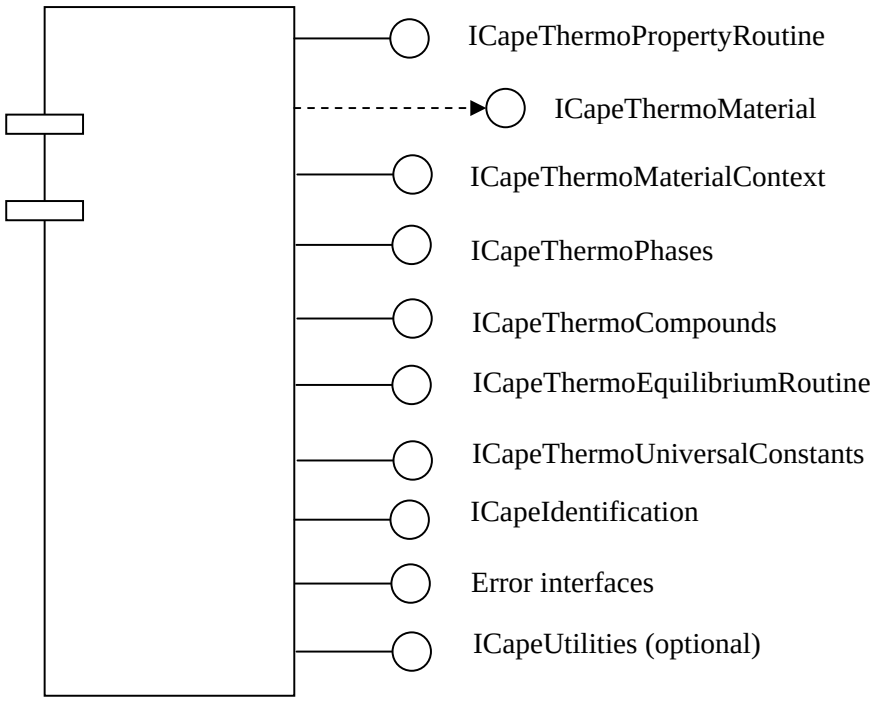
1. Evaluate h and s for each pure compound in the Physical Property Calculator/Property Package at the T, P and other conditions (*e.g.* perfect gas state) corresponding to the PME's zeros.

2. Store the values for each compound i as h_{0i} and s_{0i}.

3. Each time h or s is requested from the PP subtract $\sum n_i h_{0i}$ or $\sum n_i s_{0i}$ from the value returned by the PP (where the n_i are amounts of substance expressed in the appropriate units).

5.7 Property Package Component responsibilities

A Property Package component can describe a set of Compounds and their constant properties, it can describe the Phases that it can deal with and it can behave as a Physical Property Calculator and/or an Equilibrium Calculator. In order to support the ICapeThermoPropertyRoutine and ICapeThermoEquilibriumRoutine interfaces, it must support the ICapeThermoMaterialContext interface. For Physical Property and Equilibrium Calculation functions, it uses the ICapeThermoMaterial interface passed to it via its ICapeThermoMaterialContext::SetMaterial method to access the description of the Material being worked on from a Material Object and to update the Material's properties. The ICapeIdentification [4] and Error interfaces [5] must be implemented by all CAPE-OPEN components. The ICapeUtilities interface [17] may be implemented if the Property Package requires to make use of facilities provided by this interface, for example, the Initialize method.



The next two conceptual sequence diagrams show the expected interaction between a PME, a Property Package component and a Unit Operation component when the Unit Operation is asked to calculate its outlets. The diagrams showing the interactions with Equilibrium Calculators and Property Calculators already show the interaction between a PME and Property Package for Equilibrium and Physical Property Calculations.

In the first diagram, a PME is asked to import a Property Package, which is to be used by a CAPE-OPEN Unit Operation. The PME has to check that there is compatibility between the Compounds and Phases that the Property Package contains and that the Physical Properties that it needs can be calculated.

In the second diagram, streams are connected to a CAPE-OPEN Unit Operation and the Unit Operation is asked to perform a calculation. A Material Object that implements the Material interfaces, the Equilibrium Calculator interfaces and the Physical Property Calculator interfaces is required as an intermediary between the PME and the Unit Operation, so that the Unit Operation can request Equilibrium and Physical Property Calculations for the Materials connected to its Ports. Here, the Material Object is just a wrapper for the PME's representation of a stream, so the only data it holds is a stream name. Any requests made to it are forwarded to the PME, for stream data, or to the PME ThermoSubsystem for calculations.

Once again, these sequence diagrams are sketches, so some operations are abbreviated for simplicity and the method names used are intended to describe the intent of the call.

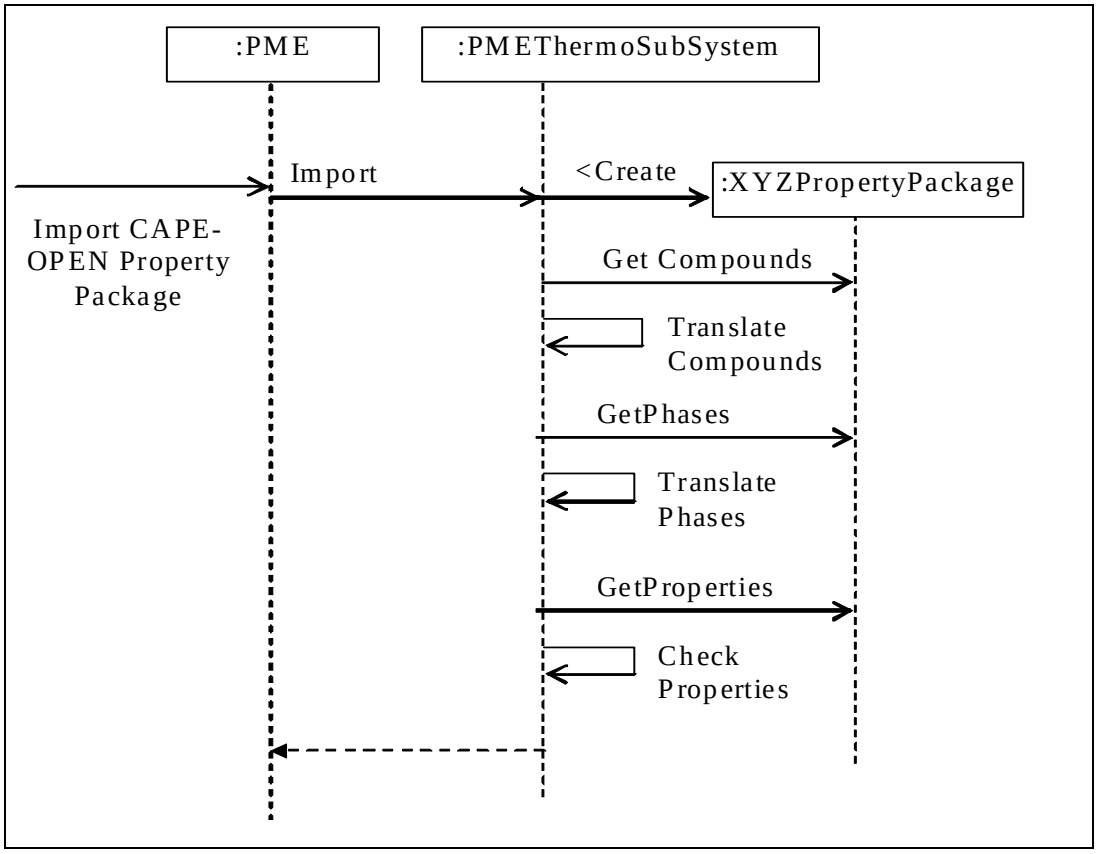


Figure 7: PME is asked to import a PP

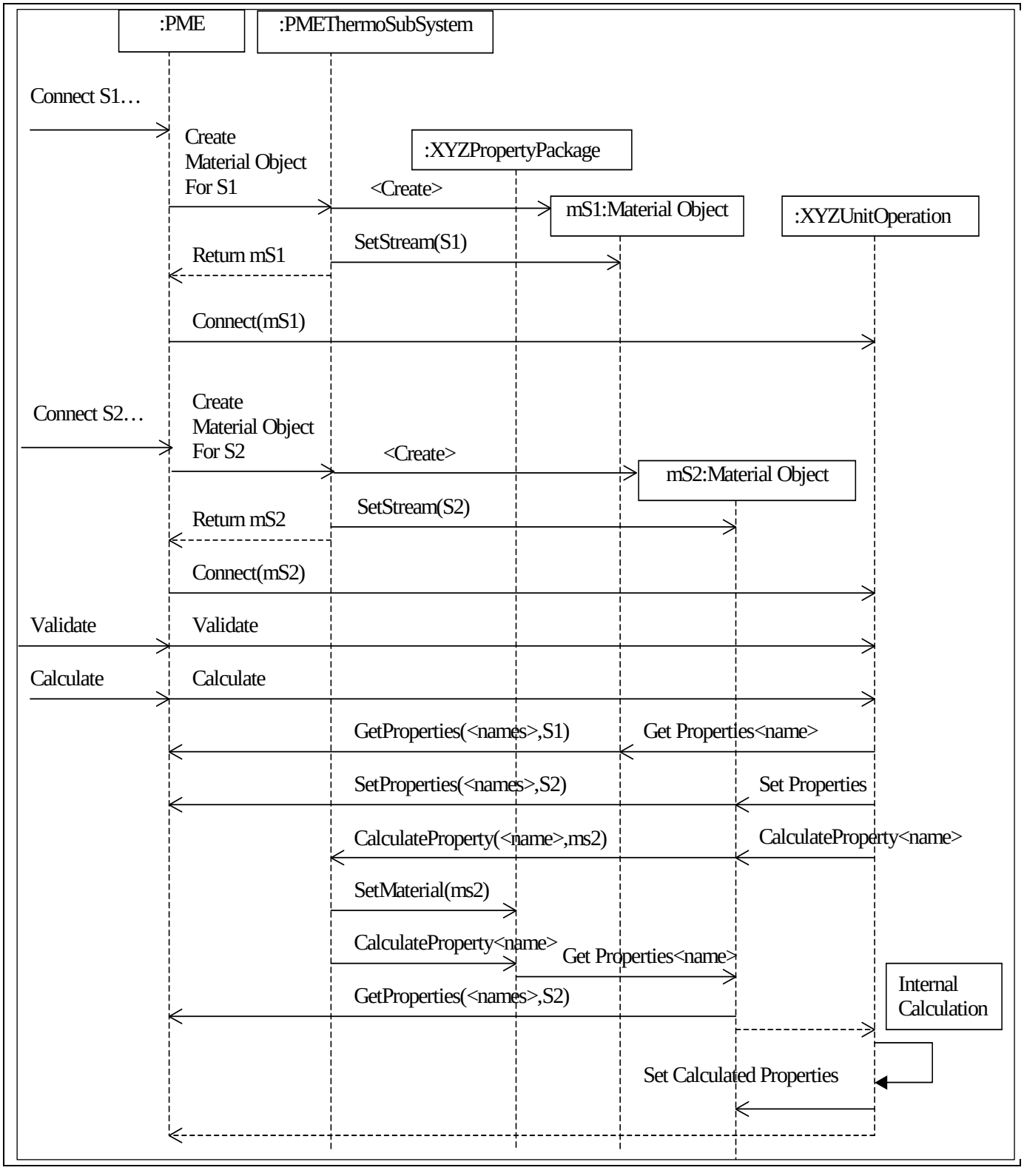


Figure 8: Streams connected to a UNIT and calculation

5.8 Property Package component behaviour

A Property Package is permitted to use the ICapeThermoCompounds, ICapeThermo UniversalConstants and ICapeThermoPropertyRoutine interfaces supported by the Material Object passed to the Property Package by the PME. The PME is required to support these call backs in order to make it possible to write Property Packages with minimum effort (for example just providing calculation routines and not pure compound physical properties). It allows the writer of a Property Package to take advantage of typical PME functionality: providing Universal Constants, Compound constants and Physical Property Calculations.

A Property Package that is configured to use external Property Calculator and Equilibrium Calculator components must be able to provide them with Material Objects in order to pass and receive data.

One implementation would be for the Property Package to provide its own Material Object implementation so that it can decide whether to handle the call itself or to pass it on to the PME via the PME's Material Object. The disadvantage with this solution is that implementing a Material Object is not simple.

The alternative implementation would be for the Property Package to pass the Material Object that it was given by its calling PME. The consequence of this implementation is that when the Property Calculator (or Equilibrium Calculator) calls a method of one of the ICapeThermoCompounds, ICapeThermoUniversalConstants or ICapeThermoPropertyRoutine interfaces implemented by the Material Object, the call will be handled by the PME, instead of the Property Package, because the Material Object is not aware of the Property Package. With this alternative, the PME should insure that there is no loop arising any call back.

The sequence diagrams showing Property Calculator and Equilibrium Calculator behaviour both show this approach.

When a Property Package implements the functionality of a Physical Property Calculator without using external Property Calculator components it must show the behaviour described for a Physical Property Calculator component in 5.6: no side effects resulting from calculations and support for different enthalpy and entropy references. A Property package component must only set properties on a passed Material Object if those properties have been requested explicitly. Unlike a Physical Property Calculator, a Property Package must implement support for the ICapeThermoCompounds and ICapeThermoPhases interfaces so that it can be used correctly by a PME.

When a Property Package implements the functionality of an Equilibrium Calculator without using an external Equilibrium Calculator component it must show the behaviour described for an Equilibrium Calculator component in 5.4: to set temperature pressure and composition for all Phases calculated to be present – with no side effects from internal calculations.

5.9 Property Package Manager responsibilities

A Property Package Manager component is responsible for managing a set of Property Packages. It implements the ICapeThermoPropertyPackageManager interface which allows a client to get a list of the names of the Property Packages managed by the component and to request that a Property Package be instantiated.

The main purpose of a Property Package Manager is to allow a component developer to implement the instantiation of a Property Package in a proprietary way, which is independent of the middleware being used.

As an example, assume that a Property Package Manager stores the description of its Property Packages in text files. When a client requests a Property Package to be instantiated, the Property Package Manager instantiates a generic object that supports the interfaces of a Property Package component. This component then reads the corresponding data file in order to configure itself.

As a primary CAPE-OPEN component, a Property Package Manager may allow a client to edit Property Packages and to create new ones.

5.10 COM Implementation details

In COM the mechanism used to access any other interface supported by the object or component is a call to the QueryInterface method, which is inherited by all COM interfaces.

To aid identification, the Property Packages managed by a COM Property Package Manager need to be described using the 'CapeDescription' registry entries specified in [8]. Since managed Property Packages do not have their own registry entries, their 'CapeDescription' entries are written as part of the registry entries for the Property Package Manager component using the following structure of keys:

HKEY_CLASSES_ROOT\CLSID

\{<Property Package Manager Class id>}

\CapeDescription

<values describing the Property Package Manager component>

\PropertyPackages

\<Name of first Property Package>

\CapeDescription

<values describing the first Property Package >

\<Name of second Property Package>

\CapeDescription

<values describing the second Property Package >

and so on for each Property Package.

Some users will not have the necessary privileges to write to "system" areas of the registry, in which case the Property Package entries should be written to the HKEY_CURRENT_USER area of the registry. The format of the entries is the same as described above except that they are located under:

HKEY_CURRENT_USER\Software\Classes\CLSID

PMEs that present lists of available Property Packages to the user must aggregate the contents of the HKEY_CURRENT_USER and HKEY_CLASSES_ROOT entries for each Property Package Manager to show a complete list of the available Property Packages. Where a Property Package with the same name occurs in both lists the HKEY_CURRENT_USER entry takes precedence.

Consistent with previous version of the CAPE-OPEN standard, COM Category Ids are defined for the Thermodynamic and Physical Property component types in order to provide a mechanism to allow a PME to identify installed components that are compliant with this version of the standard.

Description	Category ID
CAPE-OPEN 1.1 Property Package Manager	CF51E383-0110-4ed8-ACB7-B50CFDE6908E
CAPE-OPEN 1.1 Property Package	CF51E384-0110-4ed8-ACB7-B50CFDE6908E
CAPE-OPEN 1.1 Physical Property Calculator	CF51E385-0110-4ed8-ACB7-B50CFDE6908E
CAPE-OPEN 1.1 Equilibrium Calculator	CF51E386-0110-4ed8-ACB7-B50CFDE6908E

Developers of CAPE-OPEN 1.1 Thermodynamic and Physical Property components must, as part of the installation procedures for their components, ensure that these Category Ids with these descriptions exist in the registry of the machine where the components are being installed. If these category ids do not exist, then the installation procedure should create them.

For COM implementations, CO-LaN provides an installation kit that installs the CAPE-OPEN type libraries and creates all the necessary Category ids for registering components. For COM developers, CO-LaN also provides an install merge module that can be integrated with the installation kit for other components so that when the component is installed the type libraries will also be installed.

5.11 CORBA Implementation details

In CORBA implementations, a casting mechanism is used to select between the interfaces inherited by an implementation class.

There are no other CORBA-specific implementation details.

6. Interface Reference

In the following descriptions of the methods of each interface, some arguments are designated as 'ACTUALLYout'. The IDL definition of an ACTUALLYout argument is identical to [in, out] but the intent is that it is actually an output argument. ACTUALLYout is used for two reasons:

- Performance optimisation. An [in,out] declaration allows an array structure to be created only once by the client application and then reused in each call of the method.
- Language limitations. Some programming languages such as Visual Basic 6 do not allow an [out] designation for an argument.

It should be noted that the special value denoted by UNDEFINED is used in the argument descriptions. For more information on the interpretation of this value see section 7.3.

CAPE-OPEN data types are described in a CAPE-OPEN document (see [8] in the Bibliography). In particular the definition of the CapeArray types should be carefully noted.

6.1 ICapeThermoMaterial

A Material Object is a container of information that describes a Material stream. Calculations of thermophysical and thermodynamic properties are performed by a Property Package using information stored in a Material Object. Results of such calculations may be stored in the Material Object for further usage. The ICapeThermoMaterial interface provides the methods to gather information and perform checks in preparation for a calculation, to request a calculation and to retrieve results and information stored in the Material Object.

The following methods are exposed by this interface:

- **ClearAllProps**
- **CreateMaterial**
- **CopyFromMaterial**
- **GetPresentPhases**
- **GetOverallProp**
- **GetOverallTPFraction**
- **GetSinglePhaseProp**
- **GetTPFraction**
- **GetTwoPhaseProp**
- **SetOverallProp**
- **SetPresentPhases**
- **SetSinglePhaseProp**
- **SetTwoPhaseProp**

ClearAllProps

Interface Name **ICapeThermoMaterial**

Method Name ClearAllProps

Returns -

Description

Remove all stored Physical Property values.

Notes

ClearAllProps removes all stored Physical Properties that have been set using the SetSinglePhaseProp, SetTwoPhaseProp or SetOverallProp methods. This means that any subsequent call to retrieve Physical Properties will result in an exception until new values have been stored using one of the Set methods. ClearAllProps does not remove the configuration information for a Material, *i.e.* the list of Compounds and Phases.

Using the ClearAllProps method results in a Material Object that is in the same state as when it was first created. It is an alternative to using the CreateMaterial method but it is expected to have a smaller overhead in operating system resources.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeUnknown - The error to be raised when other error(s), specified for this operation, are not suitable.

CreateMaterial

Interface Name **ICapeThermoMaterial**

Method Name CreateMaterial

Returns CapeInterface

Description

Creates a Material Object with the same configuration as the current Material Object.

Arguments

Name	Type	Description
[out, retval] *materialObject*	CapeInterface	The interface for the Material Object.

Notes

The Material Object created does not contain any non-constant Physical Property value but has the same configuration (Compounds and Phases) as the current Material Object. These Physical Property values must be set using SetSinglePhaseProp, SetTwoPhaseProp or SetOverallProp. Any attempt to retrieve Physical Property values before they have been set will result in an exception.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeFailedInitialisation – The pre-requisites for the creation of the Material Object are not valid. The necessary initialisation has not been performed or has failed.

ECapeOutOfResources - The physical resources necessary to the creation of the Material Object are out of limits.

ECapeNoMemory - The physical memory necessary to the creation of the Material Object is out of limit.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

CopyFromMaterial

Interface Name **ICapeThermoMaterial**

Method Name CopyFromMaterial

Returns -

Description

Copies all the stored non-constant Physical Properties (which have been set using the SetSinglePhaseProp, SetTwoPhaseProp or SetOverallProp) from the *source* Material Object to the current instance of the Material Object.

Arguments

Name	Type	Description
[in] *source*	CapeInterface	Source Material Object from which stored properties will be copied.

Notes

Before using this method, the Material Object must have been configured with the same exact list of Compounds and Phases as the source one. Otherwise, calling the method will raise an exception. There are two ways to perform the configuration: through the PME proprietary mechanisms and with CreateMaterial. Calling CreateMaterial on a Material Object S and subsequently calling CopyFromMaterial(S) on the newly created Material Object N is equivalent to the deprecated method ICapeMaterialObject.Duplicate.

The method is intended to be used by a client, for example a Unit Operation that needs a Material Object to have the same state as one of the Material Objects it has been connected to. One example is the representation of an internal stream in a distillation column.

If the Material Object supports the Petroleum Fractions Interface [7] the petroleum fraction properties are also copied from the source Material Object to the current instance of the Material Object.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeFailedInitialisation – The pre-requisites for copying the non-constant Physical Properties of the Material Object are not valid. The necessary initialisation, such as configuring the current Material with the same Compounds and Phases as the *source*, has not been performed or has failed.

ECapeOutOfResources - The physical resources necessary to copy the non-constant Physical Properties are out of limits.

ECapeNoMemory - The physical memory necessary to copy the non-constant Physical Properties is out of limit.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetOverallProp

Interface Name **ICapeThermoMaterial**

Method Name GetOverallProp

Returns -

Description

Retrieves non-constant Physical Property values for the overall mixture.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the Physical Property for which values are requested. This must be one of the single-phase Physical Properties or derivatives that can be stored for the overall mixture. The standard identifiers are listed in sections 7.5.5 and 7.6.
[in] *basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[ACTUALLYout]*results*	CapeArrayDouble	Results vector containing Physical Property value(s) in SI units.

Notes

The Physical Property values returned by GetOverallProp refer to the overall mixture. These values are set by calling the SetOverallProp method. Overall mixture Physical Properties are not *calculated* by components that implement the ICapeThermoMaterial interface. The property values are only used as input specifications for the CalcEquilibrium method of a component that implements the ICapeThermoEquilibriumRoutine interface.

It is expected that this method will normally be able to provide Physical Property values on any basis, i.e. it should be able to convert values from the basis on which they are stored to the basis requested. This operation will not always be possible. For example, if the molecular weight is not known for one or more Compounds, it is not possible to convert between a mass basis and a molar basis.

Although the result of some calls to GetOverallProp will be a single value, the return type is CapeArrayDouble and the method must always return an array even if it contains only a single element.

Exceptions

ECapeNoImpl – The operation GetOverallProp is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeThrmPropertyNotAvailable – The Physical Property required is not available from the Material Object, possibly for the basis requested. This exception is raised when a Physical

Property value has not been set following a call to the CreateMaterial or ClearAllProps methods.

ECapeInvalidArgument – To be used when an invalid argument value was passed, for example UNDEFINED for *property*.

EcapeFailedInitialisation - The pre-requisites are not valid. The necessary initialisation has not been performed or has failed.

ECapeUnknown – The error to be raised when other error(s), specified for this operation are not suitable.

GetOverallTPFraction

Interface Name **ICapeThermoMaterial**

Method Name GetOverallTPFraction

Returns -

Description

Retrieves temperature, pressure and composition for the overall mixture.

Arguments

Name	Type	Description
[ACTUALLYout] *temperature*	CapeDouble	Temperature (in K)
[ACTUALLYout] *pressure*	CapeDouble	Pressure (in Pa)
[ACTUALLYout] *composition*	CapeArrayDouble	Composition (mole fractions)

Notes

This method is provided to make it easier for developers to make efficient use of the CAPE-OPEN interfaces. It returns the most frequently requested information from a Material Object in a single call.

There is no choice of basis in this method. The composition is always returned as mole fractions.

Exceptions

ECapeNoImpl – The operation GetOverallTPFraction is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeThrmPropertyNotAvailable – One of the Physical Properties is not available from the Material Object. This exception is raised when a Physical Property value has not been set following a call to the CreateMaterial method or the value has been erased by a call to the ClearAllProps methods.

EcapeFailedInitialisation - The pre-requisites are not valid. The necessary initialization has not been performed, or has failed.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetPresentPhases

Interface Name **ICapeThermoMaterial**
Method Name GetPresentPhases
Returns -

Description

Returns Phase labels for the Phases that are currently present in the Material Object.

Arguments

Name	Type	Description
[ACTUALLYout] *phaseLabels*	CapeArrayString	The list of Phase labels (identifiers – names) for the Phases present in the Material Object. The Phase labels in the Material Object must be a subset of the labels returned by the GetPhaseList method of the ICapeThermoPhases interface.
[ACTUALLYout] *phaseStatus*	CapeArrayEnumeration	Array of Phase status flags corresponding to each of the Phase labels. See description below.

Notes

A Phase is 'present' in a Material Object (or other component that implements the ICapeThermoMaterial interface) if it has been explicitly made present by calling the SetPresentPhases method or if any properties have been set by calling the SetSinglePhaseProp or SetTwoPhaseProp methods. Even if a Phase is present, it does not necessarily imply that any Physical Properties are actually set unless the *phaseStatus* is Cape_AtEquilibrium or Cape_Estimates (see below). Note that calling the SetPresentPhases method of the ICapeThermoMaterial interface will cause any phases not specified in its *phaseLabels* list to not present, even if previously present as a result of a SetSingle-PhaseProp or SetTwoPhaseProp call.

If no Phases are present, UNDEFINED should be returned for both the *phaseLabels* and *phaseStatus* arguments.

The *phaseStatus* argument contains as many entries as there are Phase labels. The valid settings are listed in the following table:

Identifier	Meaning
Cape_UnknownPhaseStatus	This is the normal setting when a Phase is specified as being available for an Equilibrium Calculation.
Cape_AtEquilibrium	The Phase has been set as present as a result of an Equilibrium Calculation.
Cape_Estimates	Estimates of the equilibrium state have been set in the Material Object.

All the Phases with a status of Cape_AtEquilibrium have values of temperature, pressure, composition and Phase fraction set that correspond to an equilibrium state, *i.e.* equal temperature, pressure and fugacities of each Compound. Phases with a Cape_Estimates status have values of temperature, pressure, composition and Phase fraction set in the Material Object. These values are available for use by an Equilibrium Calculator component to initialise an Equilibrium Calculation. The stored values are available but there is no guarantee that they will be used.

GetPresentPhases is intended to be used in several contexts.

- A Property Package, Property Calculator or other PMC may use this method to check whether a phase is present in the Material Object prior to requesting and/or calculating some properties.
- An Equilibrium Calculator component will use this method to obtain the list of phases to consider in an equilibrium calculation or when checking an equilibrium specification (see below for more details).
- The method will be used by the PME or PMC to obtain the list of phases present as the result of an equilibrium calculation (see below for more details).
- A Unit Operation (or other PMC) will use this method to get the list of phases present at an inlet port or during its calculations.

In the context of Equilibrium Calculations the GetPresentPhases method is intended to work in conjunction with the SetPresentPhases method. Together these methods provide a means of communication between a PME (or another client) and an Equilibrium Calculator (or other component that implements the ICapeThermoEquilibriumRoutine interface). The following sequence of operations is envisaged.

1. Prior to requesting an Equilibrium Calculation, a PME will use the SetPresentPhases method to define a list of Phases that may be considered in the Equilibrium Calculation. Typically, this is necessary because an Equilibrium Calculator may be capable of handling a large number of Phases but for a particular application, it may be known that only certain Phases will be involved. For example, if the complete Phase list contains Phases with the following labels (with the obvious interpretation): vapour, hydrocarbonLiquid and aqueousLiquid and it is required to model a liquid decanter, the present Phases might be set to hydrocarbonLiquid and aqueousLiquid.
2. The GetPresentPhases method is then used by the CalcEquilibrium method of the ICapeThermoEquilibriumRoutine interface to obtain the list of Phase labels corresponding to the Phases that may be present at equilibrium.
3. The Equilibrium Calculation determines which Phases actually co-exist at equilibrium. This list of Phases may be a sub-set of the Phases considered because some Phases may not be present at the prevailing conditions. For example, if the amount of water is sufficiently small the aqueousLiquid Phase in the above example may not exist because all the water dissolves in the hydrocarbonLiquid Phase.
4. The CalcEquilibrium method uses the SetPresentPhases method to indicate the Phases present following the equilibrium calculation (and sets the phase properties).
5. The PME uses the GetPresentPhases method to find out the Phases present following the calculation and it can then use the GetSinglePhaseProp or GetTPFraction methods to get the Phase properties.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetSinglePhaseProp

Interface Name **ICapeThermoMaterial**

Method Name GetSinglePhaseProp

Returns -

Description

Retrieves single-phase non-constant Physical Property values for a mixture.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the Physical Property for which values are requested. This must be one of the single-phase Physical Properties or derivatives. The standard identifiers are listed in sections 7.5.5 and 7.6.
[in] *phaseLabel*	CapeString	Phase label of the Phase for which the Physical Property is required. The Phase label must be one of the identifiers returned by the GetPresentPhases method of this interface.
[in] *basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[ACTUALLYout] *results*	CapeVariant	Results vector (CapeArrayDouble) containing Physical Property value(s) in SI units or CapeInterface (see notes).

Notes

The *results* argument returned by GetSinglePhaseProp is either a CapeArrayDouble that contains one or more numerical values, *e.g.* temperature, or a CapeInterface that may be used to retrieve single-phase Physical Properties described by a more complex data structure, *e.g.* distributed properties.

It is required that a component that implements the ICapeThermoMaterial interface will always support the following properties: temperature, pressure, fraction, phaseFraction, flow, totalFlow.

Although the result of some calls to GetSinglePhaseProp may be a single numerical value, the return type for numerical values is CapeArrayDouble and in such a case the method must return an array even if it contains only a single element.

A Phase is ‘present’ in a Material if its identifier is returned by the GetPresentPhases method. An exception is raised by the GetSinglePhaseProp method if the Phase specified is not present. Even if a Phase is present, this does not necessarily mean that any Physical Properties are available.

The Physical Property values returned by GetSinglePhaseProp refer to a single Phase. These values may be set by the SetSinglePhaseProp method, which may be called directly, or by

other methods such as the CalcSinglePhaseProp method of the ICapeThermoPropertyRoutine interface or the CalcEquilibrium method of the ICapeThermoEquilibriumRoutine interface. Note: Physical Properties that depend on more than one Phase, for example surface tension or K-values, are returned by the GetTwoPhaseProp method.

It is expected that this method will normally be able to provide Physical Property values on any basis, i.e. it should be able to convert values from the basis on which they are stored to the basis requested. This operation will not always be possible. For example, if the molecular weight is not known for one or more Compounds, it is not possible to convert from mass fractions or mass flows to mole fractions or molar flows.

Exceptions

ECapeNoImpl – The operation GetSinglePhaseProp is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeThrmPropertyNotAvailable – The property required is not available from the Material Object possibly for the Phase label or basis requested. This exception is raised when a property value has not been set following a call to the CreateMaterial or the value has been erased by a call to the ClearAllProps methods.

ECapeInvalidArgument – To be used when an invalid argument value was passed: for example UNDEFINED for *property*, or an unrecognised identifier for *phaseLabel*.

EcapeFailedInitialisation - The pre-requisites are not valid. The necessary initialisation has not been performed, or has failed. This exception is returned if the Phase specified does not exist.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetTPFraction

Interface Name **ICapeThermoMaterial**

Method Name GetTPFraction

Returns -

Description

Retrieves temperature, pressure and composition for a Phase.

Arguments

Name	Type	Description
[in] *phaseLabel*	CapeString	Phase label of the Phase for which the property is required. The Phase label must be one of the identifiers returned by the GetPresentPhases method of this interface.
[ACTUALLYout] *temperature*	CapeDouble	Temperature (in K)
[ACTUALLYout] *pressure*	CapeDouble	Pressure (in Pa)
[ACTUALLYout] *composition*	CapeArrayDouble	Composition (mole fractions)

Notes

This method is provided to make it easier for developers to make efficient use of the CAPE-OPEN interfaces. It returns the most frequently requested information from a Material Object in a single call.

There is no choice of basis in this method. The composition is always returned as mole fractions.

To get the equivalent information for the overall mixture the GetOverallTPFraction method of the ICapeThermoMaterial interface should be used.

Exceptions

ECapeNoImpl – The operation GetTPFraction is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeThrmPropertyNotAvailable – One of the properties is not available from the Material Object. This exception is raised when a property value has not been set following a call to the CreateMaterial or the value has been erased by a call to the ClearAllProps methods.

ECapeInvalidArgument – To be used when an invalid argument value was passed: for example an unrecognized identifier for *phaseLabel*.

EcapeFailedInitialisation - The pre-requisites are not valid. The necessary initialization has not been performed, or has failed. This exception is returned if the phase specified does not exist.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetTwoPhaseProp

Interface Name **ICapeThermoMaterial**

Method Name GetTwoPhaseProp

Returns -

Description

Retrieves two-phase non-constant Physical Property values for a mixture.

Arguments

Name	Type	Description
[in]*property*	CapeString	The identifier of the property for which values are requested. This must be one of the two-phase Physical Properties or Physical Property derivatives listed in sections 7.5.6 and 7.6.
[in]*phaseLabels*	CapeArrayString	List of Phase labels of the Phases for which the property is required. The Phase labels must be two of the identifiers returned by the GetPhaseList method of the Material Object.
[in]*basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[ACTUALLYout] *results*	CapeVariant	Results vector (CapeArrayDouble) containing property value(s) in SI units or CapeInterface (see notes).

Notes

The results argument returned by GetTwoPhaseProp is either a CapeArrayDouble that contains one or more numerical values, *e.g.* kvalues, or a CapeInterface that may be used to retrieve 2-phase Physical Properties described by a more complex data structure, *e.g.* distributed Physical Properties.

Although the result of some calls to GetTwoPhaseProp may be a single numerical value, the return type for numerical values is CapeArrayDouble and in such a case the method must return an array even if it contains only a single element.

A Phase is ‘present’ in a Material if its identifier is returned by the GetPresentPhases method. An exception is raised by the GetTwoPhaseProp method if any of the Phases specified is not present. Even if all Phases are present, this does not necessarily mean that any Physical Properties are available.

The Physical Property values returned by GetTwoPhaseProp depend on two Phases, for example surface tension or K-values. These values may be set by the SetTwoPhaseProp method that may be called directly, or by other methods such as the CalcTwoPhaseProp method of the ICapeThermoPropertyRoutine interface, or the CalcEquilibrium method of the ICapeThermoEquilibriumRoutine interface. Note: Physical Properties that depend on a single Phase are returned by the GetSinglePhaseProp method.

It is expected that this method will normally be able to provide Physical Property values on any basis, i.e. it should be able to convert values from the basis on which they are stored to the basis requested. This operation will not always be possible. For example, if the molecular weight is not known for one or more Compounds, it is not possible to convert between a mass basis and a molar basis.

If a composition derivative is requested this means that the derivatives are returned for *both* Phases in the order in which the Phase labels are specified. The number of values returned for a composition derivative will depend on the dimensionality of the property. For example, if there are *N* Compounds then the *results* vector for the surface tension derivative will contain *N* composition derivative values for the first Phase, followed by *N* composition derivative values for the second Phase. For K-value derivative there will be N^2 derivative values for the first phase followed by N^2 values for the second phase in the order defined in 7.6.2.

Exceptions

ECapeNoImpl – The operation is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This could be the case if two-phase non-constant Physical Properties are not required by the PME and so there is no particular need to implement this method.

ECapeThrmPropertyNotAvailable – the property required is not available from the Material Object possibly for the Phases or basis requested.

ECapeFailedInitialisation - The pre-requisites are not valid. This exception is raised when a call to the SetTwoPhaseProp method has not been performed, or has failed, or when one or more of the Phases referenced does not exist.

ECapeInvalidArgument – To be used when an invalid argument value was passed: for example, UNDEFINED for *property*, or an unrecognised identifier in *phaseLabels*.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

SetOverallProp

Interface Name **ICapeThermoMaterial**

Method Name SetOverallProp

Returns -

Description

Sets non-constant property values for the overall mixture.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the property for which values are set. This must be one of the single-phase properties or derivatives that can be stored for the overall mixture. The standard identifiers are listed in sections 7.5.5 and 7.6.
[in] *basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[in] *values*	CapeArrayDouble	Values to set for the property.

Notes

The property values set by SetOverallProp refer to the overall mixture. These values are retrieved by calling the GetOverallProp method. Overall mixture properties are not *calculated* by components that implement the ICapeThermoMaterial interface. The property values are only used as input specifications for the CalcEquilibrium method of a component that implements the ICapeThermoEquilibriumRoutine interface.

Although some properties set by calls to SetOverallProp will have a single value, the type of argument *values* is CapeArrayDouble and the method must always be called with *values* as an array even if it contains only a single element.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This method may not be required if the PME does not deal with any single-phase property.

ECapeInvalidArgument - To be used when an invalid argument value was passed, that is a value that does not belong to the valid list described above, for example UNDEFINED for *property*.

ECapeOutOfBounds – one or more of the entries in the *values* argument is outside of the range of values accepted by the Material Object.

ECapeUnknown – The error to be raised when other error(s), specified for the SetSinglePhaseProp operation, are not suitable.

SetPresentPhases

Interface Name **ICapeThermoMaterial**

Method Name SetPresentPhases

Returns CapeError

Description

Allows the PME or the Property Package to specify the list of Phases that are currently present.

Arguments

Name	Type	Description
[in] *phaseLabels*	CapeArrayString	The list of Phase labels for the Phases present. The Phase labels in the Material Object must be a subset of the labels returned by the GetPhaseList method of the ICapeThermoPhases interface.
[in] *phaseStatus*	CapeArrayEnumeration	Array of Phase status flags corresponding to each of the Phase labels. See description below.

Notes

SetPresentPhases is intended to be used in the following ways:

- To restrict an Equilibrium Calculation (using the CalcEquilibrium method of a component that implements the ICapeThermoEquilibriumRoutine interface) to a subset of the Phases supported by the Property Package component;
- When the component that implements the ICapeThermoEquilibriumRoutine interface needs to specify which Phases are present in a Material Object after an Equilibrium Calculation has been performed.
- In the context of dynamic simulations to specify the state of a Material Object that is an output of a unit operation. This is the equivalent of calculating equilibrium in steady-state simulations.

If a Phase in the list is already present, its Physical Properties are unchanged by the action of this method. Any Phases not in the list when SetPresentPhases is called are removed from the Material Object. This means that any Physical Property values that may have been stored on the removed Phases are no longer available (i.e. a call to GetSinglePhaseProp or GetTwoPhaseProp including this Phase will return an exception). A call to the GetPresentPhases method of the Material Object will return the same list as specified by SetPresentPhases.

The *phaseStatus* argument must contain as many entries as there are Phase labels. The valid settings are listed in the following table:

Identifier	Meaning
Cape_UnknownPhaseStatus	This is the normal setting when a Phase is specified as being available for an Equilibrium Calculation.
Cape_AtEquilibrium	The Phase has been set as present as a result of an Equilibrium Calculation.
Cape_Estimates	Estimates of the equilibrium state have been set in the Material Object.

All the Phases with a status of Cape_AtEquilibrium must have properties that correspond to an equilibrium state, *i.e.* equal temperature, pressure and fugacities of each Compound (this does not imply that the fugacities are set as a result of the Equilibrium Calculation). The Cape_AtEquilibrium status should be set by the CalcEquilibrium method of a component that implements the ICapeThermoEquilibriumRoutine interface following a successful Equilibrium Calculation. If the temperature, pressure or composition of an equilibrium Phase is changed, the Material Object implementation is responsible for resetting the status of the Phase to Cape_UnknownPhaseStatus. Other property values stored for that Phase should not be affected.

Phases with an Estimates status must have values of temperature, pressure, composition and phase fraction set in the Material Object. These values are available for use by an Equilibrium Calculator component to initialise an Equilibrium Calculation. The stored values are available but there is no guarantee that they will be used.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeInvalidArgument - To be used when an invalid argument value was passed, that is a value that does not belong to the valid list described above, for example if *phaseLabels* contains UNDEFINED or *phaseStatus* contains a value that is not in the above table.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

SetSinglePhaseProp

Interface Name **ICapeThermoMaterial**

Method Name SetSinglePhaseProp

Returns -

Description

Sets single-phase non-constant property values for a mixture.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the property for which values are set. This must be one of the single-phase properties or derivatives. The standard identifiers are listed in sections 7.5.5 and 7.6.
[in] *phaseLabel*	CapeString	Phase label of the Phase for which the property is set. The phase label must be one of the strings returned by the GetPhaseList method of the ICapeThermoPhases interface.
[in] *basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[in] *values*	CapeVariant	Values to set for the property (CapeArrayDouble) or CapeInterface (see notes).

Notes

The *values* argument of SetSinglePhaseProp is either a CapeArrayDouble that contains one or more numerical values to be set for a property, *e.g.* temperature, or a CapeInterface that may be used to set single-phase properties described by a more complex data structure, *e.g.* distributed properties.

It is required that a component that implements the ICapeThermoMaterial interface will always support the following properties: temperature, pressure, fraction, phaseFraction, flow, totalFlow.

Although some properties set by calls to SetSinglePhaseProp will have a single numerical value, the type of the *values* argument for numerical values is CapeArrayDouble and in such a case the method must be called with *values* containing an array even if it contains only a single element.

The property values set by SetSinglePhaseProp refer to a single Phase. Properties that depend on more than one Phase, for example surface tension or K-values, are set by the SetTwoPhaseProp method of the ICapeThermoMaterial Interface.

To set a property using SetSinglePhaseProp, a *phaseLabel* identifier should be passed that is supported by the Property Package or Material Object, i.e. one that appears in the list returned by the GetPhaseList method of the ICapeThermoPhases interface. Setting such a

property should cause the phase to be present on the Material Object, as if it were specified in a call to SetPresentPhases with status Cape_UnknownPhaseStatus. The SetPresentPhases method of this interface does not need to be called before calling SetSinglePhaseProp.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation. This method may not be required if the PME does not deal with any single-phase properties.

ECapeInvalidArgument - To be used when an invalid argument value was passed, that is a value that does not belong to the valid list described above, for example UNDEFINED for *property* or *phaseLabel* is not in the list returned by GetPhaseList.

ECapeOutOfBounds – one or more of the entries in the *values* argument is outside of the range of values accepted by the Material Object.

ECapeUnknown – The error to be raised when other error(s), specified for the SetSinglePhaseProp operation, are not suitable.

SetTwoPhaseProp

Interface Name **ICapeThermoMaterial**

Method Name SetTwoPhaseProp

Returns -

Description

Sets two-phase non-constant property values for a mixture.

Arguments

Name	Type	Description
[in] *property*	CapeString	The property for which values are set in the Material Object. This must be one of the two-phase properties or derivatives included in sections 7.5.6 and 7.6.
[in] *phaseLabels*	CapeArrayString	Phase labels of the Phases for which the property is set. The Phase labels must be two of the identifiers returned by the GetPhaseList method of the ICapeThermoPhases interface.
[in] *basis*	CapeString	Basis of the results. Valid settings are: “Mass” for Physical Properties per unit mass or “Mole” for molar properties. Use UNDEFINED as a place holder for a Physical Property for which basis does not apply. See section 7.5.5 for details.
[in] *values*	CapeVariant	Value(s) to set for the property (CapeArrayDouble) or CapeInterface (see notes).

Notes

The *values* argument of SetTwoPhaseProp is either a CapeArrayDouble that contains one or more numerical values to be set for a property, *e.g.* kvalues, or a CapeInterface that may be used to set two-phase properties described by a more complex data structure, *e.g.* distributed properties.

Although some properties set by calls to SetTwoPhaseProp will have a single numerical value, the type of the *values* argument for numerical values is CapeArrayDouble and in such a case the method must be called with the *values* argument containing an array even if it contains only a single element.

The Physical Property values set by SetTwoPhaseProp depend on two Phases, for example surface tension or K-values. Properties that depend on a single Phase are set by the SetSinglePhaseProp method.

If a Physical Property with composition derivative is specified, the derivative values will be set for *both* Phases in the order in which the Phase labels are specified. The number of values returned for a composition derivative will depend on the property. For example, if there are *N* Compounds then the *values* vector for the surface tension derivative will contain *N* composition derivative values for the first Phase, followed by *N* composition derivative values for the second Phase. For K-values there will be N^2 derivative values for the first phase followed by N^2 values for the second phase in the order defined in 7.6.2.

To set a property using SetTwoPhaseProp, *phaseLabels* identifiers should be passed that are supported by the Property Package or Material Object, i.e. one that appears in the list returned by the GetPhaseList method of the ICapeThermoPhases interface. Setting such a property should cause the phases to be present on the Material Object, as if it were present in a call to SetPresentPhases with status Cape_UnknownPhaseStatus. The SetPresentPhases method of this interface does not need to be called before calling SetTwoPhaseProp.

Exceptions

ECapeNoImpl – The operation is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This method may not be required if the PME does not deal with any two-phase properties.

ECapeInvalidArgument – To be used when an invalid argument value was passed, that is a value that does not belong to the valid lists described above, for example if UNDEFINED is used for identifying the property, or the calculation type, or the *phaseLabels* argument contains only one item or the *phaseLabels* are not in the list returned by GetPhaseList.

ECapeOutOfBounds – One or more of the entries in the *values* argument is outside of the range of values accepted by the Material Object, for example, negative K-values.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

6.2 ICapeThermoMaterialContext

This interface should be implemented by all Thermodynamic and Physical Properties components that need an ICapeThermoMaterial interface in order to set and get a Material's property values. The following methods are described in this section:

- **SetMaterial**
- **UnsetMaterial**

SetMaterial

Interface Name **ICapeThermoMaterialContext**

Method Name SetMaterial

Returns -

Description

Allows the client of a component that implements this interface to pass an ICapeThermoMaterial interface to the component, so that it can access the properties of a Material.

Arguments

Name	Type	Description
[in] *material*	CapeInterface	The Material interface.

Notes

The SetMaterial method allows a Thermodynamic and Physical Properties component, such as a Property Package, to be given the ICapeThermoMaterial interface of a Material Object. This interface gives the component access to the description of the Material for which Property Calculations or Equilibrium Calculations are required. The component can access property values directly using this interface. A client can also use the ICapeThermoMaterial interface to query a Material Object for its ICapeThermoCompounds and ICapeThermo-Phases interfaces, which provide access to Compound and Phase information, respectively.

It is envisaged that the SetMaterial method will be used to check that the Material Interface supplied is valid and useable. For example, a Property Package may check that there are some Compounds in a Material Object and that those Compounds can be identified by the Property Package. In addition a Property Package may perform any initialisation that depends on the configuration of a Material Object. A Property Calculator component might typically use this method to query the Material Object for any required information concerning the Compounds.

Calling the UnsetMaterial method of the ICapeThermoMaterialContext interface has the effect of removing the interface set by the SetMaterial method.

After a call to SetMaterial() has been received, the object implementing the ICapeThermo-MaterialContext interface can assume that the number, name and order of compounds for that Material Object will remain fixed until the next call to SetMaterial() or UnsetMaterial().

A PME must not call SetMaterial on a Property Package while the Property Package is in the process of performing a calculation. A scenario in which this behavior was encountered is described below for clarification.

Example: the Material Object performs a reference state correction, as outlined in section 5.6 “Physical Property Calculator behaviour”. The PME requires a PH equilibrium calculation. The PME has a Material Object for this purpose, that is configured with the overall temperature, pressure and composition. The PME sets the Material Object on the Property Package and calls CalcEquilibrium with the request for enthalpy calculation. The Property Package requests overall enthalpy from the Material Object. The Material Object determines that it has not yet calculated the compound reference enthalpies (see section 5.6), and creates a duplicate of itself to ask the Property Package to calculate the pure compound reference

rty Package, and
er of the Equilibrium calculation will fail because it references the wrong Material Object.

This example illustrates that if the Material Object requires reference values for entropies and enthalpies, it cannot postpone their calculations until the Property Package is calling the Material Object to provide enthalpy or entropy values. The PME should calculate such reference value before it asks the Property Package to perform an equilibrium calculation.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeInvalidArgument – The input argument is not a valid CapeInterface.

ECapeFailedInitialisation – The pre-requisites for the property calculation are not valid. For example:

- There are no Compounds in the object that implements the ICapeThermoMaterial interface.
- The Compounds cannot be identified by the client (*e.g.* a Property Package). This case is a possibility if the way a Material Object has been configured by a PME is not consistent with the Property Package being used.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

UnsetMaterial

Interface Name	**ICapeThermoMaterialContext**
Method Name	UnsetMaterial
Returns	CapeError

Description

Removes any previously set Material interface.

Notes

The UnsetMaterial method removes any Material interface previously set by a call to the SetMaterial method of the ICapeThermoMaterialContext interface. This means that any methods of other interfaces that depend on having a valid Material Interface, for example methods of the ICapeThermoPropertyRoutine or ICapeThermoEquilibriumRoutine interfaces, should behave in the same way as if the SetMaterial method had never been called.

If UnsetMaterial is called before a call to SetMaterial it has no effect and no exception should be raised.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable

.

6.3 ICapeThermoCompounds

Any component or object that maintains a list of Compounds must implement the ICapeThermoCompounds interface. Within the scope of this specification this means that it must be implemented by Property Package components and Material Objects. When implemented by a Property Package, this interface is used to access the list of Compounds that the Property Package can deal with, as well as the Compounds Physical Properties. When implemented by a Material Object, the interface is used for the same purpose but is applied to the Compounds present in the Material.

It is recommended for the SetMaterial method of the ICapeThermoMaterialContext interface to be called prior to calling any of the methods described below. A Property Package may contain Physical Property values for all the Compounds that it supports or it may rely on the PME to provide these data through the Material Object.

The following methods are described in this section:

- **GetCompoundConstant**
- **GetCompoundList**
- **GetConstPropList**
- **GetNumCompounds**
- **GetPDependentProperty**
- **GetPDependentPropList**
- **GetTDependentProperty**
- **GetTDependentPropList**

GetCompoundConstant

Interface Name **ICapeThermoCompounds**

Method Name GetCompoundConstant

Returns CapeArrayVariant

Description

Returns the values of constant Physical Properties for the specified Compounds.

Arguments

Name	Type	Description
[in] *props*	CapeArrayString	The list of Physical Property identifiers. Valid identifiers for constant Physical Properties are listed in section 7.5.2.
[in] *compIds*	CapeArrayString	List of Compound identifiers for which constants are to be retrieved. Set *compIds* to UNDEFINED to denote all Compounds in the component that implements the ICapeThermoCompounds interface.
[out, retval] *propvals*	CapeArrayVariant	Values of constants for the specified Compounds.

Notes

The GetConstPropList method can be used in order to check which constant Physical Properties are available.

If the number of requested Physical Properties is *P* and the number of Compounds is *C*, the *propvals* array will contain *C***P* variants. The first *C* variants will be the values for the first requested Physical Property (one variant for each Compound) followed by *C* values of constants for the second Physical Property, and so on. The actual type of values returned (Double, String, etc.) depends on the Physical Property as specified in section 7.5.2.

Physical Properties are returned in a fixed set of units as specified in section 7.5.2.

If the compIds argument is set to UNDEFINED this is a request to return property values for all compounds in the component that implements the ICapeThermoCompounds interface with the compound order the same as that returned by the GetCompoundList method. For example, if the interface is implemented by a Property Package component the property request with compIds set to UNDEFINED means *all compounds in the Property Package* rather than all compounds in the Material Object passed to the Property package.

If any Physical Property is not available for one or more Compounds, then undefined values must be returned for those combinations and an ECapeThrmPropertyNotAvailable exception must be raised. If the exception is raised, the client should check all the values returned to determine which is undefined.

Exceptions

ECapeNoImpl – The operation GetCompoundConstant is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is

to say that the operation exists, but it is not supported by the current implementation. This exception should be raised if no compounds or no properties are supported.

ECapeThrmPropertyNotAvailable – At least one item in the list of Physical Properties is not available for a particular Compound. This exception is meant to be treated as a warning rather than as an error.

ECapeLimitedImpl – One or more Physical Properties are not supported by the component that implements this interface. This exception should also be raised if any element of the *props* argument is not recognised since the list of Physical Properties in section 7.5.2 is not intended to be exhaustive and an unrecognised Physical Property identifier may be valid. If no Physical Properties at all are supported ECapeNoImpl should be raised (see above).

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example, an unrecognised Compound identifier or UNDEFINED for the *props* argument.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetCompoundConstant method. The error would not be raised when the GetCompoundConstant method is implemented by a Material Object.

GetCompoundList

Interface Name **ICapeThermoCompounds**

Method Name GetCompoundList

Returns -

Description

Returns the list of all Compounds. This includes the Compound identifiers recognised and extra information that can be used to further identify the Compounds.

Arguments

Name	Type	Description
[ACTUALLYout] *compIds*	CapeArrayString	List of Compound identifiers
[ACTUALLYout] *formulae*	CapeArrayString	List of Compound formulae
[ACTUALLYout] *names*	CapeArrayString	List of Compound names.
[ACTUALLYout] *boilTemps*	CapeArrayDouble	List of boiling point temperatures.
[ACTUALLYout] *molwts*	CapeArrayDouble	List of molecular weights.
[ACTUALLYout] *casnos*	CapeArrayString	List of Chemical Abstract Service (CAS) Registry numbers.

Notes

If any item cannot be returned then the value should be set to UNDEFINED. The same information can also be extracted using the GetCompoundConstant method. The equivalences between GetCompoundList arguments and Compound constant Physical Properties, as specified in section 7.5.2, is given in the table below.

When the ICapeThermoCompounds interface is implemented by a Material Object, the list of Compounds returned is fixed when the Material Object is configured.

For a Property Package component, the Property Package will normally contain a limited set of Compounds selected for a particular application, rather than all possible Compounds that *could* be available to a proprietary Properties System.

The *compIds* returned by the GetCompoundList method must be unique within the component that implements the ICapeThermoCompounds interface. There is no restriction on the length of the strings returned in *compIds*. However, it should be recognised that a PME may restrict the length of Compound identifiers internally. In such a case the PME's CAPE-OPEN socket must maintain a method of mapping the, potentially long, identifiers used by a CAPE-OPEN Property package component to the identifiers used within the PME.

In order to identify the Compounds of a Property Package, the PME, or other client, will use the *casnos* argument rather than the *compIds*. This is because different PMEs and different Property Packages may give different names to the same Compounds and the *casnos* is

(almost always) unique. If the *casnos* is not available (*e.g.* for petroleum fractions), or not unique, the other pieces of information returned by GetCompoundList can be used to distinguish the Compounds. It should be noted, however, that for communication with a Property Package a client must use the Compound identifiers returned in the *compIds* argument. It is the responsibility of the client to maintain appropriate data structures that allow it to reconcile the different Compound identifiers used by different Property Packages and any native property system.

GetCompoundList arguments	**Compound constant property**
compIds	No equivalence. *compIds* is an artefact, which is assigned by the component that implements the GetCompoundList method. This string must contain a unique Compound identifier such as "benzene". It must be used in all the arguments which are named "*compIds*" in the methods of the ICapeThermoCompounds and ICapeThermoMaterial interfaces.
Formulae	chemicalFormula
names	iupacName
boilTemps	normalBoilingPoint
molwts	molecularWeight
casnos	casRegistryNumber

Exceptions

ECapeNoImpl –The operation GetCompoundList is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown –The error to be raised when other error(s), specified for the GetCompoundList operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetCompoundList method. The error would not be raised when the GetCompoundList method is implemented by a Material Object.

GetConstPropList

Interface Name **ICapeThermoCompounds**

Method Name GetConstPropList

Returns CapeArrayString

Description

Returns the list of supported constant Physical Properties.

Arguments

Name	Type	Description
[out, retval] *props*	CapeArrayString	List of identifiers for all supported constant Physical Properties. The standard constant property identifiers are listed in section 7.5.2.

Notes

GetConstPropList returns identifiers for all the constant Physical Properties that can be retrieved by the GetCompoundConstant method. If no properties are supported, UNDEFINED should be returned. The CAPE-OPEN standards do not define a minimum list of Physical Properties to be made available by a software component that implements the ICapeThermoCompounds interface.

A component that implements the ICapeThermoCompounds interface may return constant Physical Property identifiers which do not belong to the list defined in section 7.5.2. However, these proprietary identifiers may not be understood by most of the clients of this component.

Exceptions

ECapeNoImpl –The operation GetConstPropList is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the Get-ConstPropList operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetConstPropList method. The error would not be raised when the GetConstPropList method is implemented by a Material Object.

GetNumCompounds

Interface Name **ICapeThermoCompounds**

Method Name GetNumCompounds

Returns CapeLong

Description

Returns the number of Compounds supported.

Arguments

Name	Type	Description
[out, retval] *num*	CapeLong	Number of Compounds supported.

Notes

The number of Compounds returned by this method must be equal to the number of Compound identifiers that are returned by the GetCompoundList method of this interface. It must be zero or a positive number.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetNumCompounds method. The error would not be raised when the GetNumCompounds method is implemented by a Material Object.

GetPDependentProperty

Interface Name **ICapeThermoCompounds**

Method Name GetPDependentProperty

Returns -

Description

Returns the values of pressure-dependent Physical Properties for the specified pure Compounds.

Arguments

Name	Type	Description
[in] *props*	CapeArrayString	The list of Physical Property identifiers. Valid identifiers for pressure-dependent Physical Properties are listed in section 7.5.4
[in] *pressure*	CapeDouble	Pressure (in Pa) at which Physical Properties are evaluated
[in] *compIds*	CapeArrayString	List of Compound identifiers for which Physical Properties are to be retrieved. Set *compIds* to UNDEFINED to denote all Compounds in the component that implements the ICapeThermoCompounds interface.
[ACTUALLYout] *propvals*	CapeArrayDouble	Property values for the Compounds specified.

Notes

The GetPDependentPropList method can be used in order to check which Physical Properties are available.

If the number of requested Physical Properties is *P* and the number Compounds is *C*, the *propvals* array will contain *C***P* values. The first *C* will be the values for the first requested Physical Property followed by *C* values for the second Physical Property, and so on.

Physical Properties are returned in a fixed set of units as specified in section 7.5.4.

If the compIds argument is set to UNDEFINED this is a request to return property values for all compounds in the component that implements the ICapeThermoCompounds interface with the compound order the same as that returned by the GetCompoundList method. For example, if the interface is implemented by a Property Package component the property request with compIds set to UNDEFINED means *all compounds in the Property Package* rather than all compounds in the Material Object passed to the Property package.

If any Physical Property is not available for one or more Compounds, then undefined values must be returned for those combinations and an ECapeThrmPropertyNotAvailable exception must be raised. If the exception is raised, the client should check all the values returned to determine which is undefined.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This exception should be raised if no Compounds or no Physical Properties are supported.

ECapeLimitedImpl – One or more Physical Properties are not supported by the component that implements this interface. This exception should also be raised (rather than ECapeInvalidArgument) if any element of the *props* argument is not recognised since the list of Physical Properties in section 7.5.4 is not intended to be exhaustive and an unrecognised Physical Property identifier may be valid. If no Physical Properties at all are supported, ECapeNoImpl should be raised (see above).

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example UNDEFINED for argument *props*.

ECapeOutOfBounds – The value of the pressure is outside of the range of values accepted by the Property Package.

ECapeThrmPropertyNotAvailable – at least one item in the properties list is not available for a particular compound.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetPDependentProperty method. The error would not be raised when the GetPDependentProperty method is implemented by a Material Object.

GetPDependentPropList

Interface Name **ICapeThermoCompounds**

Method Name GetPDependentPropList

Returns CapeArrayString

Description

Returns the list of supported pressure-dependent properties.

Arguments

Name	Type	Description
[out, retval] *props*	CapeArrayString	The list of Physical Property identifiers for all supported pressure-dependent properties. The standard identifiers are listed in section 7.5.4

Notes

GetPDependentPropList returns identifiers for all the pressure-dependent properties that can be retrieved by the GetPDependentProperty method. If no properties are supported UNDEFINED should be returned. The CAPE-OPEN standards do not define a minimum list of Physical Properties to be made available by a software component that implements the ICapeThermoCompounds interface.

A component that implements the ICapeThermoCompounds interface may return identifiers which do not belong to the list defined in section 7.5.4. However, these proprietary identifiers may not be understood by most of the clients of this component.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetPDependentPropList method. The error would not be raised when the GetPDependentPropList method is implemented by a Material Object.

GetTDependentProperty

Interface Name **ICapeThermoCompounds**

Method Name GetTDependentProperty

Returns -

Description

Returns the values of temperature-dependent Physical Properties for the specified pure Compounds.

Arguments

Name	Type	Description
[in] *props*	CapeArrayString	The list of Physical Property identifiers. Valid identifiers for temperature-dependent Physical Properties are listed in section 7.5.3
[in] *temperature*	CapeDouble	Temperature (in K) at which properties are evaluated
[in] *compIds*	CapeArrayString	List of Compound identifiers for which Physical Properties are to be retrieved. Set *compIds* to UNDEFINED to denote all Compounds in the component that implements the ICapeThermoCompounds interface.
[ACTUALLYout] *propvals*	CapeArrayDouble	Physical Property values for the Compounds specified.

Notes

The GetTDependentPropList method can be used in order to check which Physical Properties are available.

If the number of requested Physical Properties is *P* and the number of Compounds is *C*, the *propvals* array will contain *C***P* values. The first *C* will be the values for the first requested Physical Property followed by *C* values for the second Physical Property, and so on.

Properties are returned in a fixed set of units as specified in section 7.5.3.

If the compIds argument is set to UNDEFINED this is a request to return property values for all compounds in the component that implements the ICapeThermoCompounds interface with the compound order the same as that returned by the GetCompoundList method. For example, if the interface is implemented by a Property Package component the property request with compIds set to UNDEFINED means *all compounds in the Property Package* rather than all compounds in the Material Object passed to the Property package.

If any Physical Property is not available for one or more Compounds, then undefined values must be returned for those combinations and an ECapeThrmPropertyNotAvailable exception must be raised. If the exception is raised, the client should check all the values returned to determine which is undefined.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation

exists, but it is not supported by the current implementation. This exception should be raised if no Compounds or no Physical Properties are supported.

ECapeLimitedImpl – One or more Physical Properties are not supported by the component that implements this interface. This exception should also be raised (rather than ECapeInvalidArgument) if any element of the *props* argument is not recognised since the list of properties in section 7.5.3 is not intended to be exhaustive and an unrecognised Physical Property identifier may be valid. If no properties at all are supported ECapeNoImpl should be raised (see above).

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example UNDEFINED for argument *props*.

ECapeOutOfBounds – The value of the temperature is outside of the range of values accepted by the Property Package.

ECapeThrmPropertyNotAvailable – at least one item in the properties list is not available for a particular compound.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetTDependentProperty method. The error would not be raised when the GetTDependentProperty method is implemented by a Material Object.

GetTDependentPropList

Interface Name **ICapeThermoCompounds**

Method Name GetTDependentPropList

Returns CapeArrayString

Description

Returns the list of supported temperature-dependent Physical Properties.

Arguments

Name	Type	Description
[out, retval] *props*	CapeArrayString	The list of Physical Property identifiers for all supported temperature-dependent properties. The standard identifiers are listed in section 7.5.3

Notes

GetTDependentPropList returns identifiers for all the temperature-dependent Physical Properties that can be retrieved by the GetTDependentProperty method. If no properties are supported UNDEFINED should be returned. The CAPE-OPEN standards do not define a minimum list of properties to be made available by a software component that implements the ICapeThermoCompounds interface.

A component that implements the ICapeThermoCompounds interface may return identifiers which do not belong to the list defined in section 7.5.3. However, these proprietary identifiers may not be understood by most of the clients of this component.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the operation, are not suitable.

ECapeBadInvOrder – The error to be raised if the Property Package required the SetMaterial method to be called before calling the GetTDependentPropList method. The error would not be raised when the GetTDependentPropList method is implemented by a Material Object.

6.4 ICapeThermoPhases

This interface is designed to provide information about the number and types of Phases supported by the component that implements it. It defines all the Phases that a component such as a Physical Property Calculator can handle. It does not provide information about the Phases that are actually present in a Material Object. This function is provided by the GetPresentPhases method of the ICapeThermoMaterial interface.

The following methods are described in this section:

- **GetNumPhases**
- **GetPhaseInfo**
- **GetPhaseList**

GetNumPhases

Interface Name **ICapeThermoPhases**

Method Name GetNumPhases

Returns CapeLong

Description

Returns the number of Phases.

Arguments

Name	Type	Description
[out, retval] *num*	CapeLong	The number of Phases supported.

Notes

The number of Phases returned by this method must be equal to the number of Phase labels that are returned by the GetPhaseList method of this interface. It must be zero, or a positive number.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetPhaseInfo

Interface Name **ICapeThermoPhases**

Method Name GetPhaseInfo

Returns CapeVariant

Description

Returns information on an attribute associated with a Phase for the purpose of understanding what lies behind a Phase label.

Arguments

Name	Type	Description
[in] *phaseLabel*	CapeString	A (single) Phase label. This must be one of the values returned by GetPhaseList method.
[in] *phaseAttribute*	CapeString	One of the Phase attribute identifiers from the table below.
[out, retval] *value*	CapeVariant	The value corresponding to the Phase attribute identifier – see table below.

Notes

GetPhaseInfo is intended to allow a PME, or other client, to identify a Phase with an arbitrary label. A PME, or other client, will need to do this to map stream data into a Material Object, or when importing a Property Package. If the client cannot identify the Phase, it can ask the user to provide a mapping based on the values of these properties.

The list of supported Phase attributes is defined in the following table:

Phase attribute identifier	Supported values
StateOfAggregation	One of the following strings: Vapor Liquid Solid Unknown
KeyCompoundId	The identifier of the Compound (*compId* as returned by GetCompoundList) that is expected to be present in highest concentration in the Phase. May be undefined in which case UNDEFINED should be returned.
ExcludedCompoundId	The identifier of the Compound (*compId* as returned by GetCompoundList) that is expected to be present in low or zero concentration in the Phase. May not be defined in which case UNDEFINED should be returned.
DensityDescription	A description that indicates the density range expected for the Phase. One of the following strings or UNDEFINED: Heavy Light

UserDescription	A description that helps the user or PME to identify the Phase. It can be any string or UNDEFINED.
TypeOfSolid	A description that provides more information about a solid Phase. For Phases with a "Solid" state of aggregation it may be one of the following standard strings or UNDEFINED: PureSolid SolidSolution HydrateI HydrateII HydrateH Other values may be returned for solid Phases but these may not be understood by most clients. For Phases with any other state of aggregation it must be UNDEFINED.

For example, the following information might be returned by a Property Package component that supports a vapour Phase, an organic liquid Phase and an aqueous liquid Phase:

Phase label	**Gas**	**Organic**	**Aqueous**
StateOfAggregation	Vapor	Liquid	Liquid
KeyCompoundId	UNDEFINED	UNDEFINED	Water
ExcludedCompoundId	UNDEFINED	Water	UNDEFINED
DensityDescription	UNDEFINED	Light	Heavy
UserDescription	The gas Phase	The organic liquid Phase	The aqueous liquid Phase
TypeOfSolid	UNDEFINED	UNDEFINED	UNDEFINED

Exceptions

ECapeNoImpl – The operation is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists but it is not supported by the current implementation.

ECapeInvalidArgument – *phaseLabel* is not recognised, or UNDEFINED, or *phaseAttribute* is not recognised.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

GetPhaseList

Interface Name **ICapeThermoPhases**

Method Name GetPhaseList

Returns -

Description

Returns Phase labels and other important descriptive information for all the Phases supported.

Arguments

Name	Type	Description
[ACTUALLYout] *phaseLabels*	CapeArrayString	The list of Phase labels for the Phases supported. A Phase label can be any string but each Phase must have a unique label. If, for some reason, no Phases are supported an UNDEFINED value should be returned for the *phaseLabels*. The number of Phase labels must also be equal to the number of Phases returned by the GetNumPhases method.
[ACTUALLYout] *stateOfAggregation*	CapeArrayString	The physical State of Aggregation associated with each of the Phases. This must be one of the following strings: ”Vapor”, “Liquid”, “Solid” or “Unknown”. Each Phase must have a single State of Aggregation. The value must not be left undefined, but may be set to “Unknown”.
[ACTUALLYout] *keyCompoundId*	CapeArrayString	The key Compound for the Phase. This must be the Compound identifier (as returned by GetCompoundList), or it may be undefined in which case a UNDEFINED value is returned. The key Compound is an indication of the Compound that is expected to be present in high concentration in the Phase, *e.g.* water for an aqueous liquid phase. Each Phase can have a single key Compound.

Notes

The Phase label allows the phase to be uniquely identified in methods of the ICapeThermo-Phases interface and other CAPE-OPEN interfaces. The State of Aggregation and key Compound provide a way for the PME, or other client, to interpret the meaning of a Phase label in terms of the physical characteristics of the Phase.

All arrays returned by this method must be of the same length, *i.e.* equal to the number of Phase labels.

To get further information about a Phase, use the GetPhaseInfo method.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

6.5 ICapeThermoPropertyRoutine

Any Component or object that can calculate a Physical Property must implement the ICapeThermoPropertyRoutine interface. Within the scope of this specification this means that it must be implemented by Calculation Routine components, Property Package components and Material Object implementations that will be passed to clients which may need to perform Property Calculations, such as Unit Operations [2] and Reaction Package components [3].

When the ICapeThermoPropertyRoutine interface is implemented by a Material Object, it is expected that the actual Calculate, Check and Get functions will be delegated either to proprietary methods within a PME or to methods in an associated CAPE-OPEN Property Package or Calculation Routine component.

The following methods are described in this section:

- **CalcAndGetLnPhi**
- **CalcSinglePhaseProp**
- **CalcTwoPhaseProp**
- **CheckSinglePhasePropSpec**
- **CheckTwoPhasePropSpec**
- **GetSinglePhasePropList**
- **GetTwoPhasePropList**

CalcAndGetLnPhi

Interface Name **ICapeThermoPropertyRoutine**

Method Name CalcAndGetLnPhi

Returns -

Description

This method is used to calculate the natural logarithm of the fugacity coefficients (and optionally their derivatives) in a single Phase mixture. The values of temperature, pressure and composition are specified in the argument list and the results are also returned through the argument list.

Arguments

Name	Type	Description
[in] *phaseLabel*	CapeString	Phase label of the Phase for which the properties are to be calculated. The Phase label must be one of the strings returned by the GetPhaseList method on the ICapeThermoPhases interface.
[in] *temperature*	CapeDouble	The temperature (K) for the calculation.
[in] *pressure*	CapeDouble	The pressure (Pa) for the calculation.
[in] *moleNumbers*	CapeArrayDouble	Mole fractions of Compounds in the mixture.
[in] *fFlags*	CapeInteger	Code indicating whether natural logarithm of the fugacity coefficients and/or derivatives should be calculated (see notes).
[ACTUALLYout] *lnPhi*	CapeArrayDouble	Natural logarithm of the fugacity coefficients (if requested).
[ACTUALLYout] *lnPhiDT*	CapeArrayDouble	Derivatives of natural logarithm of the fugacity coefficients w.r.t. temperature (if requested).
[ACTUALLYout] *lnPhiDP*	CapeArrayDouble	Derivatives of natural logarithm of the fugacity coefficients w.r.t. pressure (if requested).
[ACTUALLYout] *lnPhiDn*	CapeArrayDouble	Derivatives of natural logarithm of the fugacity coefficients w.r.t. mole numbers (if requested).

Notes

This method is provided to allow the natural logarithm of the fugacity coefficient, which is the most commonly used thermodynamic property, to be calculated and returned in a highly efficient manner.

The temperature, pressure and composition (mole fractions) for the calculation are specified by the arguments and are not obtained from the Material Object by a separate request. Note that the *moleNumbers* argument should actually contain mole fractions. This inconsistency in the argument name arises because the specification has been revised but it is desired to keep the COM IDL unchanged.

Likewise, any quantities calculated are returned through the arguments and are not stored in the Material Object. The state of the Material Object is not affected by calling this method. It

should be noted however, that prior to calling CalcAndGetLnPhi a valid Material Object must have been defined by calling the SetMaterial method on the ICapeThermoMaterialContext interface of the component that implements the ICapeThermoPropertyRoutine interface. The compounds in the Material Object must have been identified and the number of mole fraction values supplied in the *moleNumbers* argument must be equal to the number of Compounds in the Material Object. It should not be assumed that the mole fractions are normalized and values may also lie outside the range 0 to 1. If fractions are not normalized, or are outside the expected range, it is the responsibility of the Property Package to decide how to deal with the situation.

The fugacity coefficient information is returned as the natural logarithm of the fugacity coefficient. This is because thermodynamic models naturally provide the natural logarithm of this quantity and also a wider range of values may be safely returned.

The quantities actually calculated and returned by this method are controlled by an integer code *fFlags*. The code is formed by summing contributions for the property and each derivative required using the enumerated constants eCapeCalculationCode (defined in the Thermo version 1.1 IDL) shown in the following table. For example, to calculate log fugacity coefficients and their T-derivatives the *fFlags* argument would be set to CAPE_LOG_FUGACITY_COEFFICIENTS + CAPE_T_DERIVATIVE.

	code	**numerical value**
no calculation	CAPE_NO_CALCULATION	0
log fugacity coefficients	CAPE_LOG_FUGACITY_COEFFICIENTS	1
T-derivative	CAPE_T_DERIVATIVE	2
P-derivative	CAPE_P_DERIVATIVE	4
mole number derivatives	CAPE_MOLE_NUMBERS_DERIVATIVES	8

If CalcAndGetLnPhi is called with *fFlags* set to CAPE_NO_CALCULATION no property values are returned; the CAPE_NO_CALCULATION is provided for completeness and is generally not used.

The values returned by this method should be identical to the equivalent quantities returned by the CalcSinglePhaseProp method with the same input information set on the Material Object. In particular, this means that the mole number derivatives are evaluated for a total of one mole of substance.

A typical sequence of operations for this method when implemented by a Property Package component would be:

- Check that the *phaseLabel* specified is valid.
- Check that the *moleNumbers* array contains the number of values expected (should be consistent with the last call to the SetMaterial method).
- Calculate the requested properties/derivatives at the T/P/composition specified in the argument list.
- Store values for the properties/derivatives in the corresponding arguments.

Note that this calculation can be carried out irrespective of whether the Phase actually exists in the Material Object.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeLimitedImpl – Would be raised if the one or more of the properties requested cannot be returned because the calculation is not implemented.

ECapeBadInvOrder - The necessary pre-requisite operation has not been called prior to the operation request. For example, the ICapeThermoMaterial interface has not been passed via a SetMaterial call prior to calling this method.

ECapeFailedInitialisation - The pre-requisites for the Property Calculation are not valid. For example, the composition of the phase is not defined, the number of Compounds in the Material Object is zero or not consistent with the *moleNumbers* argument or any other necessary input information is not available.

ECapeThrmPropertyNotAvailable – At least one item in the requested properties cannot be returned. This could be because the property cannot be calculated at the specified conditions or for the specified Phase. If the property calculation is not implemented then ECapeLimitedImpl should be returned.

ECapeSolvingError – One of the property calculations has failed. For example if one of the iterative solution procedures in the model has run out of iterations, or has converged to a wrong solution.

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example an unrecognised value, or UNDEFINED for the *phaseLabel* argument.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

CalcSinglePhaseProp

Interface Name **ICapeThermoPropertyRoutine**

Method Name CalcSinglePhaseProp

Returns -

Description

CalcSinglePhaseProp is used to calculate properties and property derivatives of a mixture in a single Phase at the current values of temperature, pressure and composition set in the Material Object. CalcSinglePhaseProp does not perform phase Equilibrium Calculations.

Arguments

Name	Type	Description
[in] *props*	CapeArrayString	The list of identifiers for the single-phase properties or derivatives to be calculated. See sections 7.5.5 and 7.6 for the standard identifiers.
[in] *phaseLabel*	CapeString	Phase label of the Phase for which the properties are to be calculated. The Phase label must be one of the strings returned by the GetPhaseList method on the ICapeThermoPhases interface and the phase must be present in the Material Object.

Notes

CalcSinglePhaseProp calculates properties, such as enthalpy or viscosity that are defined for a single Phase. Physical Properties that depend on more than one Phase, for example surface tension or K-values, are handled by CalcTwoPhaseProp method.

Components that implement this method must get the input specification for the calculation (temperature, pressure and composition) from the associated Material Object and set the results in the Material Object.

Thermodynamic and Physical Properties Components, such as a Property Package or Property Calculator, must implement the ICapeThermoMaterialContext interface so that an ICapeThermoMaterial interface can be passed via the SetMaterial method.

The component that implements the ICapeThermoPropertyRoutine interface (*e.g.* a Property Package or Property Calculator) must also implement the ICapeThermoPhases interface so that it is possible to get a list of supported phases. The *phaseLabel* passed to this method must be one of the phase labels returned by the GetPhaseList method of the ICapeThermoPhases interface and it must also be present in the Material Object, ie. one of the phase labels returned by the GetPresentPhases method of the ICapeThermoMaterial interface. This latter condition will be satisfied if the phase is made present explicitly by calling the SetPresentPhases method or if any phase properties have been set by calling the SetSinglePhaseProp or SetTwoPhaseProp methods.

A typical sequence of operations for CalcSinglePhaseProp when implemented by a Property Package component would be:

- Check that the *phaseLabel* specified is valid.

- Use the GetTPFraction method (of the Material Object specified in the last call to the SetMaterial method) to get the temperature, pressure and composition of the specified Phase.
- Calculate the properties.
- Store values for the properties of the Phase in the Material Object using the SetSinglePhaseProp method of the ICapeThermoMaterial interface.

CalcSinglePhaseProp will request the input Property values it requires from the Material Object through GetSinglePhaseProp calls. If a requested property is not available, the exception raised will be ECapeThrmPropertyNotAvailable. If this error occurs then the Property Package can return it to the client, or request a different property. Material Object implementations must be able to supply property values using the client's choice of basis by implementing conversion from one basis to another.

Clients should not assume that Phase fractions and Compound fractions in a Material Object are normalised. Fraction values may also lie outside the range 0 to 1. If fractions are not normalised, or are outside the expected range, it is the responsibility of the Property Package to decide how to deal with the situation.

It is recommended that properties are requested one at a time in order to simplify error handling. However, it is recognised that there are cases where the potential efficiency gains of requesting several properties simultaneously are more important. One such example might be when a property and its derivatives are required.

If a client uses multiple properties in a call and one of them fails then the whole call should be considered to have failed. This implies that 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.

It is likely that a PME might request values of properties for a Phase at conditions of temperature, pressure and composition where the Phase does not exist (according to the mathematical/physical models used to represent properties). The exception ECapeThrmPropertyNotAvailable may be raised or an extrapolated value may be returned. It is responsibility of the implementer to decide how to handle this circumstance.

Exceptions

ECapeNoImpl – The operation is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeLimitedImpl – Would be raised if the one or more of the properties requested cannot be returned because the calculation (of the particular property) is not implemented. This exception should also be raised (rather than ECapeInvalidArgument) if the *props* argument is not recognised because the list of properties in section 7.5.5 is not intended to be exhaustive and an unrecognised property identifier may be valid. If no properties at all are supported ECapeNoImpl should be raised (see above).

ECapeBadInvOrder - The necessary pre-requisite operation has not been called prior to the operation request. For example, the ICapeThermoMaterial interface has not been passed via a SetMaterial call prior to calling this method.

ECapeFailedInitialisation - The pre-requisites for the property calculation are not valid. For example, the composition of the phases is not defined or any other necessary input information is not available.

ECapeThrmPropertyNotAvailable – At least one item in the requested properties cannot be returned. This could be because the property cannot be calculated at the specified conditions or for the specified phase. If the property calculation is not implemented then ECapeLimitedImpl should be returned.

ECapeSolvingError – One of the property calculations has failed. For example if one of the iterative solution procedures in the model has run out of iterations, or has converged to a wrong solution.

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example an unrecognised value or UNDEFINED for the *phaseLabel* argument or UNDEFINED for the *props* argument.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

CalcTwoPhaseProp

Interface Name **ICapeThermoPropertyRoutine**

Method Name CalcTwoPhaseProp

Returns -

Description

CalcTwoPhaseProp is used to calculate mixture properties and property derivatives that depend on two Phases at the current values of temperature, pressure and composition set in the Material Object. It does not perform Equilibrium Calculations.

Arguments

Name	Type	Description
[in] *props*	CapeArrayString	The list of identifiers for properties to be calculated. This must be one or more of the supported two-phase properties and derivatives (as given by the GetTwoPhasePropList method). The standard identifiers for two-phase properties are given in section 7.5.6 and 7.6.
[in] *phaseLabels*	CapeArrayString	Phase labels of the phases for which the properties are to be calculated. The phase labels must be two of the strings returned by the GetPhaseList method on the ICapeThermoPhases interface and the phases must also be present in the Material Object.

Notes

CalcTwoPhaseProp calculates the values of properties such as surface tension or K-values. Properties that pertain to a single Phase are handled by the CalcSinglePhaseProp method of the ICapeThermoPropertyRoutine interface.Components that implement this method must get the input specification for the calculation (temperature, pressure and composition) from the associated Material Object and set the results in the Material Object.

Components such as a Property Package or Property Calculator must implement the ICapeThermoMaterialContext interface so that an ICapeThermoMaterial interface can be passed via the SetMaterial method.

The component that implements the ICapeThermoPropertyRoutine interface (*e.g.* a Property Package or Property Calculator) must also implement the ICapeThermoPhases interface so that it is possible to get a list of supported phases. The *phaseLabels* passed to this method must be in the list of phase labels returned by the GetPhaseList method of the ICapeThermoPhases interface and they must also be present in the Material Object, ie. in the list of phase labels returned by the GetPresentPhases method of the ICapeThermoMaterial interface. This latter condition will be satisfied if the phases are made present explicitly by calling the SetPresentPhases method or if any phase properties have been set by calling the SetSinglePhaseProp or SetTwoPhaseProp methods.

A typical sequence of operations for CalcTwoPhaseProp when implemented by a Property Package component would be:

- Check that the *phaseLabels* specified are valid.
- Use the GetTPFraction method (of the Material Object specified in the last call to the SetMaterial method) to get the temperature, pressure and composition of the specified Phases.
- Calculate the properties.
- Store values for the properties in the Material Object using the SetTwoPhaseProp method of the ICapeThermoMaterial interface.

CalcTwoPhaseProp will request the values it requires from the Material Object through GetTPFraction or GetSinglePhaseProp calls. If a requested property is not available, the exception raised will be ECapeThrmPropertyNotAvailable. If this error occurs, then the Property Package can return it to the client, or request a different property. Material Object implementations must be able to supply property values using the client choice of basis by implementing conversion from one basis to another.

Clients should not assume that Phase fractions and Compound fractions in a Material Object are normalised. Fraction values may also lie outside the range 0 to 1. If fractions are not normalised, or are outside the expected range, it is the responsibility of the Property Package to decide how to deal with the situation.

It is recommended that properties are requested one at a time in order to simplify error handling. However, it is recognised that there are cases where the potential efficiency gains of requesting several properties simultaneously are more important. One such example might be when a property and its derivatives are required.

If a client uses multiple properties in a call and one of them fails, then the whole call should be considered to have failed. This implies that 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.

CalcTwoPhaseProp must be called separately for each combination of Phase groupings. For example, vapour-liquid K-values have to be calculated in a separate call from liquid-liquid K-values.

Two-phase properties may not be meaningful unless the temperatures and pressures of all Phases are identical. It is the responsibility of the Property Package to check such conditions and to raise an exception if appropriate.

It is likely that a PME might request values of properties for Phases at conditions of temperature, pressure and composition where one or both of the Phases do not exist (according to the mathematical/physical models used to represent properties). The exception ECapeThrmPropertyNotAvailable may be raised or an extrapolated value may be returned. It is responsibility of the implementer to decide how to handle this circumstance.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeLimitedImpl – Would be raised if the one or more of the properties requested cannot be returned because the calculation (of the particular property) is not implemented. This exception should also be raised (rather than ECapeInvalidArgument) if the *props* argument is not recognised because the list of properties in section 7.5.6 is not intended to be

dentifier may be valid. If no properties at all are supported ECapeNoImpl should be raised (see above).

ECapeBadInvOrder - The necessary pre-requisite operation has not been called prior to the operation request. For example, the ICapeThermoMaterial interface has not been passed via a SetMaterial call prior to calling this method.

ECapeFailedInitialisation - The pre-requisites for the property calculation are not valid. For example, the composition of one of the Phases is not defined, or any other necessary input information is not available.

ECapeThrmPropertyNotAvailable – At least one item in the requested properties cannot be returned. This could be because the property cannot be calculated at the specified conditions or for the specified Phase. If the property calculation is not implemented then ECapeLimitedImpl should be returned.

ECapeSolvingError – One of the property calculations has failed. For example if one of the iterative solution procedures in the model has run out of iterations, or has converged to a wrong solution.

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example an unrecognised value or UNDEFINED for the *phaseLabels* argument or UNDEFINED for the *props* argument.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

CheckSinglePhasePropSpec

Interface Name **ICapeThermoPropertyRoutine**

Method Name CheckSinglePhasePropSpec

Returns CapeBoolean

Description

Checks whether it is possible to calculate a property with the CalcSinglePhaseProp method for a given Phase.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the property to check. To be valid this must be one of the supported single-phase properties or derivatives (as given by the GetSinglePhasePropList method).
[in] *phaseLabel*	CapeString	The Phase label for the calculation check. This must be one of the labels returned by the GetPhaseList method on the ICapeThermoPhases interface.
[out, retval] *valid*	CapeBoolean	Set to *True* if the combination of property and phaseLabel is supported or *False* if not supported.

Notes

The result of the check should only depend on the capabilities and configuration (Compounds and Phases supported) of the component that implements the ICapeThermoPropertyRoutine interface (*e.g.* a Property Package). It should not depend on whether a Material Object has been set nor on the state (temperature, pressure, composition etc.), or configuration of a Material Object that might be set.

It is expected that the PME, or other client, will use this method to check whether the properties it requires are supported by the Property Package when the package is imported. If any essential properties are not available, the import process should be aborted.

If either the *property* or the *phaseLabel* arguments are not recognised by the component that implements the ICapeThermoPropertyRoutine interface this method should return *False*.

Exceptions

ECapeNoImpl –The operation CheckSinglePhasePropSpec is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeInvalidArgument – One or more of the input arguments is not valid: for example, UNDEFINED value for the *property* argument or the *phaseLabel* argument.

ECapeUnknown – The error to be raised when other error(s), specified for the CheckSinglePhasePropSpec operation, are not suitable.

CheckTwoPhasePropSpec

Interface Name **ICapeThermoPropertyRoutine**

Method Name CheckTwoPhasePropSpec

Returns CapeBoolean

Description

Checks whether it is possible to calculate a property with the CalcTwoPhaseProp method for a given set of Phases.

Arguments

Name	Type	Description
[in] *property*	CapeString	The identifier of the property to check. To be valid this must be one of the supported two-phase properties (including derivatives), as given by the GetTwoPhasePropList method.
[in] *phaseLabels*	CapeArrayString	Phase labels of the Phases for which the properties are to be calculated. The Phase labels must be two of the identifiers returned by the GetPhaseList method on the ICapeThermoPhases interface.
[out, retval] *valid*	CapeBoolean	Set to *True* if the combination of property and phaseLabels is supported, or *False* if not supported.

Notes

The result of the check should only depend on the capabilities and configuration (Compounds and Phases supported) of the component that implements the ICapeThermoPropertyRoutine interface (*e.g.* a Property Package). It should not depend on whether a Material Object has been set nor on the state (temperature, pressure, composition etc.), or configuration of a Material Object that might be set.

It is expected that the PME, or other client, will use this method to check whether the properties it requires are supported by the Property Package when the Property Package is imported. If any essential properties are not available, the import process should be aborted.

If either the *property* argument or the values in the *phaseLabels* arguments are not recognised by the component that implements the ICapeThermoPropertyRoutine interface this method should return *False*.

Exceptions

ECapeNoImpl – The operation CheckTwoPhasePropSpec is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This may be the case if no two-phase property is supported.

ECapeInvalidArgument – One or more of the input arguments is not valid. For example, UNDEFINED value for the *property* argument or the *phaseLabels* argument or number of elements in *phaseLabels* array not equal to two.

ECapeUnknown – The error to be raised when other error(s), specified for the CheckTwoPhasePropSpec operation, are not suitable.

GetSinglePhasePropList

Interface Name **ICapeThermoPropertyRoutine**

Method Name GetSinglePhasePropList

Returns CapeArrayString

Description

Returns the list of supported non-constant single-phase Physical Properties.

Arguments

Name	Type	Description
[out, retval] *props*	CapeArrayString	List of all supported non-constant single-phase property identifiers. The standard single-phase property identifiers are listed in section 7.5.5.

Notes

A non-constant property depends on the state of the Material Object.

Single-phase properties, *e.g.* enthalpy, only depend on the state of one phase. GetSinglePhasePropList must return all the single-phase properties that can be calculated by CalcSinglePhaseProp. If derivatives can be calculated these must also be returned. The list of standard property identifiers in section 7.5.5 also contains properties such as temperature, pressure, fraction, phaseFraction, flow and totalFlow that are not usually calculated by the CalcSinglePhaseProp method and hence these property identifiers would not be returned by GetSinglePhasePropList. These properties would normally be used in calls to the Set/GetSinglePhaseProp methods of the ICapeThermoMaterial interface.

If no single-phase properties are supported this method should return UNDEFINED.

To get the list of supported two-phase properties, use GetTwoPhasePropList.

A component that implements this method may return non-constant single-phase property identifiers which do not belong to the list defined in section 7.5.5. However, these proprietary identifiers may not be understood by most of the clients of this component.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the GetSinglePhasePropList operation, are not suitable.

GetTwoPhasePropList

Interface Name **ICapeThermoPropertyRoutine**

Method Name GetTwoPhasePropList

Returns CapeArrayString

Description

Returns the list of supported non-constant two-phase properties.

Arguments

Name	Type	Description
[out, retval] *props*	CapeArrayString	List of all supported non-constant two-phase property identifiers. The standard two-phase property identifiers are listed in section 7.5.6.

Notes

A non-constant property depends on the state of the Material Object. Two-phase properties are those that depend on more than one co-existing phase, *e.g.* K-values.

GetTwoPhasePropList must return all the properties that can be calculated by CalcTwoPhaseProp. If derivatives can be calculated, these must also be returned.

If no two-phase properties are supported this method should return UNDEFINED.

To check whether a property can be evaluated for a particular set of phase labels use the CheckTwoPhasePropSpec method.

A component that implements this method may return non-constant two-phase property identifiers which do not belong to the list defined in section 7.5.6. However, these proprietary identifiers may not be understood by most of the clients of this component.

To get the list of supported single-phase properties, use GetSinglePhasePropList.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the GetTwoPhasePropList operation, are not suitable.

6.6 ICapeThermoEquilibriumRoutine

Any component or object that can perform an Equilibrium Calculation must implement the ICapeThermoEquilibriumRoutine interface. Within the scope of this specification, this means that it must be implemented by Equilibrium Calculator components, Property Package components and by Material Object implementations that will be passed to clients which may need to perform Equilibrium Calculations, such as Unit Operations [2].

When a Material Object implements the ICapeThermoEquilibriumRoutine interface, it is expected that the methods will be delegated either to proprietary methods within a PME, or to methods in an associated CAPE-OPEN Property Package or Equilibrium Calculator component.

- **CalcEquilibrium**
- **CheckEquilibriumSpec**

CalcEquilibrium

Interface Name **ICapeThermoEquilibriumRoutine**

Method Name CalcEquilibrium

Returns -

Description

CalcEquilibrium is used to calculate the amounts and compositions of Phases at equilibrium. CalcEquilibrium will calculate temperature and/or pressure if these are not among the two specifications that are mandatory for each Equilibrium Calculation considered.

Arguments

Name	Type	Description
[in]*specification1*	CapeArrayString	First specification for the Equilibrium Calculation. The specification information is used to retrieve the *value* of the specification from the Material Object. See below for details.
[in]*specification2*	CapeArrayString	Second specification for the Equilibrium Calculation in the same format as *specification1*.
[in]*solutionType*	CapeString	The identifier for the required solution type. The standard identifiers are given in the following list: Unspecified Normal Retrograde The meaning of these terms is defined below in the notes. Other identifiers may be supported but their interpretation is not part of the CO standard.

Notes

The *specification1* and *specification2* arguments provide the information necessary to retrieve the values of two specifications, for example the pressure and temperature, for the Equilibrium Calculation. The CheckEquilibriumSpec method can be used to check for supported specifications. Each specification variable contains a sequence of strings in the order defined in the following table (hence, the specification arguments may have 3 or 4 items):

item	meaning
property identifier	The property identifier can be any of the identifiers listed in section 7.5.5 but only certain property specifications will normally be supported by any Equilibrium Routine.
basis	The basis for the property value. Valid settings for basis are given in section 7.4. Use UNDEFINED as a placeholder for a property for which basis does not apply. For most Equilibrium Specifications, the result of the calculation is not dependent on the basis, but, for example, for phase fraction specifications the basis (Mole or Mass) does make a difference.

phase label	The phase label denotes the Phase to which the specification applies. It must either be one of the labels returned by GetPresentPhases, or the special value "Overall".
compound identifier (optional)	The compound identifier allows for specifications that depend on a particular Compound. This item of the specification array is optional and may be omitted. In case of a specification without compound identifier, the array element may be present and empty, or may be absent.

Some examples of typical phase equilibrium specifications are given in the table below.

Type of phase equilibrium calculation	**specification1**	**specification2**	**Comments**
Fixed temperature and pressure	temperature UNDEFINED Overall	pressure UNDEFINED Overall	
Fixed pressure and enthalpy	pressure UNDEFINED Overall	enthalpy UNDEFINED Overall	UNDEFINED is used as the basis for the enthalpy specification because the result of the calculation does not depend on the basis (*i.e.* CalcEquilibrium can request the enthalpy from a Material Object on any basis using the GetOverallProp method)
Fixed temperature and enthalpy	temperature UNDEFINED Overall	enthalpy UNDEFINED Overall	
Fixed temperature and molar fraction of vapour phase	temperature UNDEFINED Overall	phaseFraction Mole gas	1. Assumes Phase label of the vapour Phase is "gas" 2. The value of the Phase fraction must have been set in the Material Object. For a dew point this would be 1.0, for a bubble point it would be 0.0 and it could be set to any value in between. The basis setting indicates that the specification is for phase fraction on a *molar* basis.
Fixed pressure and activity of water	pressure UNDEFINED Overall	activityCoefficient UNDEFINED AqueousLiquid water	1. Assumes Phase label of the aqueous Phase is "AqueousLiquid" 2. The value of the activity coefficient for water must have been set in the Material Object.

The values corresponding to the specifications in the argument list and the overall composition of the mixture must be set in the associated Material Object before a call to CalcEquilibrium.

Components such as a Property Package or an Equilibrium Calculator must implement the ICapeThermoMaterialContext interface, so that an ICapeThermoMaterial interface can be

passed via the SetMaterial method. It is the responsibility of the implementation of CalcEquilibrium to validate the Material Object before attempting a calculation.

The Phases that will be considered in the Equilibrium Calculation are those that exist in the Material Object, *i.e.* the list of phases specified in a SetPresentPhases call. This provides a way for a client to specify whether, for example, a vapour-liquid, liquid-liquid, or vapour-liquid-liquid calculation is required. CalcEquilibrium must use the GetPresentPhases method to retrieve the list of Phases and the associated Phase status flags. The Phase status flags may be used by the client to provide information about the Phases, for example whether estimates of the equilibrium state are provided. See the description of the GetPresentPhases and SetPresentPhases methods of the ICapeThermoMaterial interface for details. When the Equilibrium Calculation has been completed successfully, the SetPresentPhases method must be used to specify which Phases are present at equilibrium and the Phase status flags for the phases should be set to Cape_AtEquilibrium. This must include any Phases that are present in zero amount such as the liquid Phase in a dew point calculation.

Some types of Phase equilibrium specifications may result in more than one solution. A common example of this is the case of a dew point calculation. However, CalcEquilibrium can provide only one solution through the Material Object. The solutionType argument allows the "Normal" or "Retrograde" solution to be explicitly requested. The following definitions are intended for use when one of the specifications includes a phase fraction and the other is temperature or pressure.

Specification	Normal	Retrograde
T and *Fv*	$\left(\frac{\partial F_V}{\partial p}\right)_T \leq 0$	$\left(\frac{\partial F_V}{\partial p}\right)_T > 0$
T and F_L	$\left(\frac{\partial F_L}{\partial p}\right)_T \geq 0$	$\left(\frac{\partial F_L}{\partial p}\right)_T < 0$
T and F_s	$\left(\frac{\partial F_S}{\partial p}\right)_T \geq 0$	$\left(\frac{\partial F_S}{\partial p}\right)_T < 0$
P and *Fv*	$\left(\frac{\partial F_V}{\partial T}\right)_p \geq 0$	$\left(\frac{\partial F_V}{\partial T}\right)_p < 0$
P and F_L	$\left(\frac{\partial F_L}{\partial T}\right)_p \leq 0$	$\left(\frac{\partial F_L}{\partial T}\right)_p > 0$
P and F_S	$\left(\frac{\partial F_S}{\partial T}\right)_p \leq 0$	$\left(\frac{\partial F_S}{\partial T}\right)_p > 0$

where F_V is a vapor phase fraction, F_L is a liquid phase fraction and F_S is a solid phase fraction. The derivatives are at equilibrium states. When none of the specifications includes a phase fraction and P or T, the solutionType argument should be set to "Unspecified".

CalcEquilibrium must set the amounts (phase fractions), 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. It must not set any other values – in particular it must not set any values for phases that are not present.

As an example, the following sequence of operations might be performed by CalcEquilibrium in the case of an Equilibrium Calculation at fixed pressure and temperature:

- With the ICapeThermoMaterial interface of the supplied Material Object:
    - Use the GetPresentPhases method to find the list of Phases that the Equilibrium Calculation should consider.
    - With the ICapeThermoCompounds interface of the Material Object use the GetCompoundList method to find which Compounds are present.
    - Use the GetOverallProp method to get the temperature, pressure and composition for the overall mixture.
- Perform the Equilibrium Calculation.
    - Use SetPresentPhases to specify the Phases present at equilibrium and set the Phase status flags to Cape_AtEquilibrium.
    - Use SetSinglePhaseProp to set pressure, temperature, Phase amount (or Phase fraction) and composition for all Phases present.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeBadInvOrder - The necessary pre-requisite operation has not been called prior to the operation request. The ICapeThermoMaterial interface has not been passed via a SetMaterial call prior to calling this method.

ECapeSolvingError – The Equilibrium Calculation could not be solved. For example if the solver has run out of iterations, or has converged to a trivial solution.

ECapeLimitedImpl – Would be raised if the Equilibrium Routine is not able to perform the flash it has been asked to perform. For example, the values given to the input specifications are valid, but the routine is not able to perform a flash given a temperature and a Compound fraction. That would imply a bad usage or no usage of CheckEquilibriumSpec method, which is there to prevent calling CalcEquilibrium for a calculation which cannot be performed.

ECapeInvalidArgument – To be used when an invalid argument value is passed. It would be raised, for example, if a specification identifier does not belong to the list of recognised identifiers. It would also be raised if the value given to argument *solutionType* is not among the three defined, or if UNDEFINED was used instead of a specification identifier.

ECapeFailedInitialisation - The pre-requisites for the Equilibrium Calculation are not valid. For example:

- The overall composition of the mixture is not defined.
- The Material Object (set by a previous call to the SetMaterial method of the ICapeThermoMaterialContext interface) is not valid. This could be because no

Phases are present or because the Phases present are not recognised by the component that implements the ICapeThermoEquilibriumRoutine interface.

- Any other necessary input information is not available.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

CheckEquilibriumSpec

Interface Name **ICapeThermoEquilibriumRoutine**
Method Name CheckEquilibriumSpec
Returns CapeBoolean

Description

Checks whether the Property Package can support a particular type of Equilibrium Calculation.

Arguments

Name	Type	Description
[in]*specification1*	CapeArrayString	First specification for the Equilibrium Calculation.
[in]*specification2*	CapeArrayString	Second specification for the Equilibrium Calculation.
[in]*solutionType*	CapeString	The required solution type.
[out, retval] *isSupported*	CapeBoolean	Set to *True* if the combination of specifications and solutionType is supported for a particular combination of present phases or *False* if not supported.

Notes

The meaning of the *specification1*, *specification2* and *solutionType* arguments is the same as for the CalcEquilibrium method. If *solutionType*, *specification1* and *specification2* arguments appear valid but the actual specifications are not supported or not recognised a *False* value should be returned.

The result of the check should depend primarily on the capabilities and configuration (compounds and phases supported) of the component that implements the ICapeThermo-EquilibriumRoutine interface (egg. a Property package). A component that supports calculation specifications for any combination of supported phases is capable of checking the specification without any reference to a Material Object. However, it is possible that there may be restrictions on the combinations of phases supported in an equilibrium calculation. For example a component may support vapor-liquid and liquid-liquid calculations but not vapor-liquid-liquid calculations. In general it is therefore a necessary prerequisite that a Material Object has been set (using the SetMaterial method of the ICapeThermoMaterialContext interface) and that the SetPresentPhases method of the ICapeThermoMaterial interface has been called to specify the combination of phases for the equilibrium calculation. The result of the check should not depend on the state (temperature, pressure, composition etc.) of the Material Object.

Exceptions

ECapeNoImpl – The operation is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeInvalidArgument – To be used when an invalid argument value is passed, for example UNDEFINED for *solutionType*, *specification1* or *specification2* argument.

ECapeBadInvOrder - The necessary pre-requisite operation has not been called prior to the operation request. *E.g.* the ICapeThermoMaterial interface has not been passed via a SetMaterial call prior to calling this method.

ECapeUnknown – The error to be raised when other error(s), specified for this operation, are not suitable.

6.7 ICapeThermoUniversalConstants

Any component that can return the value of a Universal Constant can implement the ICapeThermoUniversalConstants interface in order that clients can access these values. This interface is optional for all components. It is recommended that it is implemented by Property Package components and Material Objects being used by CAPE-OPEN Unit Operations.

The following methods are described in this section:

- **GetUniversalConstant**
- **GetUniversalConstantList**

GetUniversalConstant

Interface Name **ICapeThermoUniversalConstants**

Method Name GetUniversalConstant

Returns CapeVariant

Description

Retrieves the value of a Universal Constant.

Arguments

Name	Type	Description
[in] *constantId*	CapeString	Identifier of Universal Constant. The list of constants supported should be obtained by using the GetUniversalConstantList method.
[out, retval] *constantValue*	CapeVariant	Value of Universal Constant. This could be a numeric or a string value. For numeric values the units of measurement are specified in section 7.5.1.

Notes

Universal Constants (often called fundamental constants) are quantities like the gas constant, or the Avogadro constant.

Exceptions

ECapeNoImpl –The operation GetUniversalConstant is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeInvalidArgument –for example, UNDEFINED for *constantId* argument is used, or value for *constantId* argument does not belong to the list of recognised values.

ECapeUnknown – The error to be raised when other error(s), specified for the GetUniversalConstant operation, are not suitable.

GetUniversalConstantList

Interface Name **ICapeThermoUniversalConstants**

Method Name GetUniversalConstantList

Returns CapeArrayString

Description

Returns the identifiers of the supported Universal Constants.

Arguments

Name	Type	Description
[out, retval] constantIds	CapeArrayString	List of identifiers of Universal Constants. The list of standard identifiers is given in section 7.5.1.

Notes

A component may return Universal Constant identifiers that do not belong to the list defined in section 7.5.1. However, these proprietary identifiers may not be understood by most of the clients of this component.

Exceptions

ECapeNoImpl –The operation GetUniversalConstantList is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation. This may occur when the Property Package does not support any Universal Constants, or if it does not want to provide values for any Universal Constants which may be used within the Property Package.

ECapeUnknown – The error to be raised when other error(s), specified for the GetUniversalConstantList operation, are not suitable.

6.8 ICapeThermoPropertyPackageManager

The ICapeThermoPropertyPackageManager interface should only be implemented by a Property Package Manager component. This interface is used to access the Property Packages managed by such a component.

The following methods are described in this section:

- **GetPropertyPackageList**
- **GetPropertyPackage**

GetPropertyPackageList

Interface Name **ICapeThermoPropertyPackageManager**

Method Name GetPropertyPackageList

Returns CapeArrayString

Description

Retrieves the names of the Property Packages being managed by a Property Package Manager component.

Arguments

Name	Type	Description
[out, retval] *PackageNames*	CapeArrayString	The names of the managed Property Packages

Notes

If no packages are managed by the Property Package Manager UNDEFINED should be returned.

Exceptions

ECapeNoImpl –The operation GetPropertyPackageList is “not” implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeUnknown – The error to be raised when other error(s), specified for the GetPropertyPackageList operation, are not suitable.

GetPropertyPackage

Interface Name **ICapeThermoPropertyPackageManager**

Method Name GetPropertyPackage

Returns CapeInterface

Description

Creates a new instance of a Property Package with the configuration specified by the *PackageName* argument.

Arguments

Name	Type	Description
[in] *PackageName*	CapeString	The name of one of the Property Packages managed by this Property Package Manager component.
[out, retval] *Package*	CapeInterface	The ICapeThermoPropertyRoutine interface of the named Property Package.

Notes

The Property Package Manager is only an indirect mechanism to create Property Packages. After the Property Package has been created, the Property Package Manager instance can be destroyed, and this will not affect the normal behaviour of the Property Packages.

Exceptions

ECapeNoImpl –The operation GetPropertyPackage is "not" implemented even if this method can be called for reasons of compatibility with the CAPE-OPEN standards. That is to say that the operation exists, but it is not supported by the current implementation.

ECapeFailedInitialisation – This error should be returned if the Property Package cannot be created for any reason.

ECapeInvalidArgument – This error will be returned if the name of the Property Package asked for does not belong to the list of recognised names. Comparison of names is not case sensitive.

ECapeUnknown – The error to be raised when other error(s), specified for the GetPropertyPackage operation, are not suitable.

7. Property Descriptions

7.1 Case-sensitivity of identifiers

Although the identifiers listed in this section are shown in a combination of lower- and upper-case characters, all identifiers must be treated as case-independent in any implementation of the interfaces described in this document.

7.2 Units of measurement

The units of measurement for all properties are base SI units. The unit to be used for each property is listed along with the property identifier in these appendices. For more information on units refer to the *Bureau International des Poids et Mesures* website http://www.bipm.fr/enus/3_SI/si.html.

7.3 UNDEFINED interpretation

Be aware that the UNDEFINED value depends on the type of the corresponding argument. This special value is described in [8]. Reference [8] does not include an interpretation of UNDEFINED for a CapeInterface. For COM this is NULL (Empty object for VB).

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).

7.4 Identifiers for Basis

The following strings or placeholder may be used for the basis argument of the methods described in this document.

Identifier	Meaning
Mole	molar basis
Mass	mass basis
UNDEFINED	basis does not apply to the property

7.5 Property Identifiers

7.5.1 Universal constants

The following constants are returned by the GetUniversalConstant method of the ICapeThermoUniversalConstants interface. The possible return types are Double or String. Note: only the units of measurement are specified in the CAPE-OPEN standards, not the values.

Identifier	typical value	units	return type
avogadroConstant	$6.022\ 141\ 99(47)\times10^{23}$	1/mol	Double
boltzmannConstant	$1.380\ 6503(24)\times10^{-23}$	J /K	Double
idealGasStateReferencePressure	101325	Pa	Double
molarGasConstant	8.314 472(15)	J /mol/ K	Double
speedOfLightInVacuum	$2.99792458(1)\times10^{8}$	m/s	Double
standardAccelerationOfGravity	9.806 65	m /s^2	Double

7.5.2 Pure compound constant properties

STRING-VALUED PROPERTIES

The following properties are returned as string values by the GetCompoundConstant method of the ICapeThermoCompounds interface:

Identifier	meaning
casRegistryNumber	Chemical Abstract Service Registry Number
chemicalFormula	Chemical formula
iupacName	Complete IUPAC Name
SMILESformula	SMILES chemical structure formula

CASRegistryNumber

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

CAS numbers and other properties are accessible at

http://webbook.nist.gov/chemistry

Compounds can also be accessed directly. For example, compounds with the formula C10H22 can be located with

http://webbook.nist.gov/cgi/cbook.cgi?Formula=c10h22&NoIon=on&Units=SI

or compounds can be located by name. For example

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.

ChemicalFormula

The formula is delivered in Hill nomenclature [10]: organic compounds: first C, then H, other atoms alphabetical; inorganic compounds: all atoms alphabetical.

It is not obligatory to specify unitary atomicities explicitly. For example, the formula for carbon dioxide may be returned as CO2 or C1O2. The formula string must be case-sensitive so that it is possible to distinguish between cases such as Co (cobalt) and CO (carbon monoxide).

The atomicity may be non-integer, eg. $Fe_{0.947}O$ should be represented as Fe0.947O. The decimal separator in the character string that represents the non-integer atomicity must be a period (decimal point).

iupacName

Name assigned in accordance with the recommendations of IUPAC Blue Book [15, 16] for organic compounds and the IUPAC Red Book [14] for inorganic compounds. The Blue book is available online at http://www.acdlabs.com/iupac/nomenclature/ and the acdlabs website also has software that can be used to derive the IUPAC name for an organic compound.

SMILESformula

The Simplified Molecular Input Line Entry Specification that represents the structure of the molecule [9].

PROPERTIES WITH NUMERICAL VALUES

The following properties are returned by the GetCompoundConstant method of the ICapeThermoCompounds interface as numerical (Double) values.

Identifier	**meaning**	**units**
acentricFactor	Pitzer acentric factor	
associationParameter	association-parameter (Hayden-O'Connell)	
bornRadius		m
charge		
criticalCompressibilityFactor	critical compressibility factor *Z*	
criticalDensity	critical density	mol/m^3
criticalPressure	critical pressure	Pa
criticalTemperature	critical temperature	K
criticalVolume	critical volume	m^3/mol
diffusionVolume	diffusion volume	m^3
dipoleMoment	dipole moment	C m
energyLennardJones	Lennard-Jones energy parameter (divided by Boltzmann constant)	K
gyrationRadius	radius of gyration	m
heatOfFusionAtNormalFreezingPoint	enthalpy change on melting at normal freezing point (101325 Pa)	J/mol
heatOfVaporizationAtNormalBoilingPoint	enthalpy change on vaporization at normal boiling point (101325 Pa)	J/mol
idealGasEnthalpyOfFormationAt25C		J/mol
idealGasGibbsFreeEnergyOfFormationAt25C		J/mol
liquidDensityAt25C	liquid density at 25 ºC	mol/m^3
liquidVolumeAt25C	liquid volume at 25 ºC	m^3/mol
lengthLennardJones	Lennard-Jones length parameter	m
molecularWeight	relative molar mass	
normalBoilingPoint	boiling point temperature at 101325 Pa	K
normalFreezingPoint	melting point temperature at 101325 Pa	K
parachor	Parachor	$m^3\ kg^{0.25}/(s^{0.5}\ mol)$

Identifier	meaning	units
standardEntropyGas	Standard entropy of gas	J/mol
standardEntropyLiquid	standard entropy of liquid	J/mol
standardEntropySolid	standard entropy of solid	J/mol
standardEnthalpyAqueousDilution	Standard aqueous infinite dilution enthalpy	J/mol
standardFormationEnthalpyGas	standard enthalpy change on formation of gas	J/mol
standardFormationEnthalpyLiquid	standard enthalpy change on formation of liquid	J/mol
standardFormationEnthalpySolid	standard enthalpy change on formation of solid	J/mol
standardFormationGibbsEnergyGas	standard Gibbs energy change on formation of gas	J/mol
standardFormationGibbsEnergyLiquid	standard Gibbs energy change on formation of liquid	J/mol
standardFormationGibbsEnergySolid	standard Gibbs energy change on formation of solid	J/mol
standardGibbsAqueousDilution	Standard aqueous infinite dilution Gibbs energy	J/mol
triplePointPressure	triple point pressure	Pa
triplePointTemperature	triple point temperature	K
vanderwaalsArea	van der Waals area	m^2/mol
vanderwaalsVolume	van der Waals volume	m^3/mol

Standard conditions are 298.15 K (25 ºC) and the pressure returned by the GetUniversalConstant method for the property idealGasStateReferencePressure. If this property is not available a reference pressure of 101325 Pa (1 atm) may be assumed.

Note that the ‘standardFormationGibbsEnergyGas’, ‘standardFormationGibbsEnergyLiquid’ and ‘standardFormationGibbsEnergySolid’ identifiers respectively replace the ‘standardFreeFormationEnthalpyGas’, ‘standardFreeFormationEnthalpyLiquid’ and ‘standardFreeFormationEnthalpySolid’ identifiers used in previous versions of this specification.

7.5.3 Temperature-dependent pure compound properties

The following properties are returned as numerical (Double) values by the GetTDependentProperty method of the ICapeThermoCompounds interface:

Identifier	Meaning	units
cpAqueousInfiniteDilution	Heat capacity of a solute in an infinitely dilute aqueous solution.	J/(mol K)
dielectricConstant	The ratio of the capacity of a condenser with a particular substance as dielectric to the capacity of the same condenser with a vacuum for dielectric.	
expansivity	Coefficient of linear expansion for a solid: $\frac{1}{L}\left\|\frac{\partial L}{\partial T}\right\|$ (where L is the length) at 1 atm	1/K
fugacityCoefficientOfVapor	Fugacity coefficient of vapour on the saturation line	
glassTransitionPressure	Glass transition pressure	Pa
heatCapacityOfLiquid	Heat capacity (Cp) of liquid on the saturation line	J/(mol K)
heatCapacityOfSolid	Solid heat capacity (Cp) at 1 atm	J/(mol K)
heatOfFusion	Enthalpy change on fusion for the solid on the melting line	J/mol
heatOfSublimation	Enthalpy change on evaporation of the solid on the sublimation line	J/mol
heatOfSolidSolidPhaseTransition	Enthalpy change on phase transition	J/mol
heatOfVaporization	Enthalpy change on evaporation of the liquid on the saturation line	J/mol
idealGasEnthalpy	Enthalpy of ideal gas	J/mol
idealGasEntropy	Temperature-dependent part of entropy of ideal gas	J/(mol K)
idealGasHeatCapacity	Heat capacity (Cp) of ideal gas	J/(mol K)
meltingPressure	Pressure on melting line	Pa
selfDiffusionCoefficientGas	Self-diffusion coefficient in gas phase at 1 atm	m^2/s
selfDiffusionCoefficientLiquid	self-diffusion coefficient in liquid phase on saturation line	m^2/s
solidSolidPhaseTransitionPressure	Pressure at phase transition	Pa
sublimationPressure	Vapour pressure of solid on the sublimation line	Pa
surfaceTensionSatLiquid	Surface tension of liquid on the saturation line	N/m
thermalConductivityOfLiquid	Thermal conductivity of liquid on saturation line	W/(m K)
thermalConductivityOfSolid	Thermal conductivity of solid at 1 atm	W/(m K)
thermalConductivityOfVapor	Thermal conductivity of dilute gas	W/(m K)

Identifier	Meaning	units
vaporPressure	Vapour pressure of saturated liquid	Pa
virialCoefficient	Second virial coefficient of gas	m^3/mol
viscosityOfLiquid	Viscosity of liquid on saturation line	Pas
viscosityOfVapor	Viscosity in dilute gas state	Pas
volumeChangeUponMelting	Volume change for the solid on the melting line	m^3/mol
volumeChangeUponSolidSolidPhaseTransition	Volume change upon solid-solid phase transition	m^3/mol
volumeChangeUponSublimation	Volume change for the solid on the sublimation line	m^3/mol
volumeChangeUponVaporization	Volume change for the liquid on the saturation line	m^3/mol
volumeOfLiquid	Volume of liquid on saturation line	m^3/mol
volumeOfSolid	Volume of solid at 1 atm	m^3/mol

The properties in the table above are independent of pressure. Properties like the perfect gas heat capacity or virial coefficient are independent of pressure by definition. For properties at a phase transition, the pressure is implied by the temperature. For example, the pressure for the heatOfVaporization property is given by the vaporPressure property at the same temperature.

If a software component requires the value of the pressure as part as the calculation of one of the properties listed above, the software component is responsible for calculating the pressure value.

7.5.4 Pressure-dependent pure compound properties

The following properties are returned as numerical (Double) values by the GetPDependentProperty method of the ICapeThermoCompounds interface:

Identifier	Meaning	units
boilingPointTemperature	Temperature at liquid-vapour transition	K
glassTransitionTemperature	Glass transition temperature	K
meltingTemperature	Temperature on melting line	K
solidSolidPhaseTransitionTemperature	Temperature at phase transition	K

The properties in the table above are independent of temperature. For properties at a phase transition, the temperature is fixed by the pressure.

If a software component requires the value of the temperature as part as the calculation of one of the properties listed above, the software component is responsible for calculating the temperature value.

7.5.5 Non-constant single-phase mixture properties

See section 7.5.7 for more information on entries in this table.

Identifier	Meaning	type of prop-erty	dimens ionality	units	basis	over-all
activity	Activity	I	1		U	
activityCoefficient	Activity coefficient	I	1		U	
compressibility	Isothermal compressibility $\frac{1}{V}\left(\frac{\partial V}{\partial P}\right)_T$	I		1/Pa	U	
compressibilityFactor	Compressibility factor $Z = \frac{PV}{NRT}$	I			U	Y
density	Density	I		mol/m^3	mole/mass	Y
diffusionCoefficient	Binary diffusion coefficients for all species in mixture relative to all other species	I	2	m^2/s	U	
dissociationConstant	Chemical equilibrium constant corresponding to a dissociation reaction.	I			U	
enthalpy	Enthalpy (may or may not include the enthalpy of formation)	E		J	mole/mass	Y
enthalpyF	Enthalpy, including the enthalpy of formation	E		J	mole/mass	Y
enthalpyNF	Enthalpy, not including the enthalpy of formation	E		J	mole/mass	Y
entropy	Entropy (may or may not include the entropy of formation)	E		J/K	mole/mass	Y
entropyF	Entropy, including the entropy of formation	E		J/K	mole/mass	Y
entropyNF	Entropy, not including the entropy of formation	E		J/K	mole/mass	Y
excessEnthalpy	Excess enthalpy	E		J	mole/mass	
excessEntropy	Excess entropy	E		J/K	mole/mass	
excessGibbsEnergy	Excess Gibbs energy	E		J	mole/mass	

Identifier	Meaning	type of property	dimensionality	units	basis	overall
excessHelmholtzEnergy	Excess Helmholtz energy	E		J	mole/mass	
excessInternalEnergy	Excess internal energy	E		J	mole/mass	
excessVolume	Excess volume	E		m^3	mole/mass	
flow	Flows of each Compound in a given Phase (or the overall mixture)	E	1	mol/s	mole/mass	Y
fraction	Molar (or mass) fractions of each Compound in a given Phase (or the overall mixture)	I	1		mole/mass	Y
fugacity	Fugacity	I	1	Pa	U	
fugacityCoefficient	Fugacity coefficient	I	1		U	
gibbsEnergy	Gibbs energy	E		J	mole/mass	Y
heatCapacityCp	Heat capacity at constant pressure (Cp)	E		J/K	mole/mass	Y
heatCapacityCv	Heat capacity at constant volume (Cv)	E		J/ K	mole/mass	Y
helmholtzEnergy	Helmholtz energy	E		J	mole/mass	Y
internalEnergy	Internal energy	E		J	mole/mass	Y
jouleThomsonCoefficient	$\left(\frac{\partial T}{\partial P}\right)_H$	I		K/Pa	U	
logFugacity	Natural logarithm of fugacity (expressed in Pa)	I	1		U	
logFugacityCoefficient	Natural logarithm of fugacity coefficient	I	1		U	
meanActivityCoefficient	The geometrical mean of the activity coefficients of the ions in an electrolyte solution.	I	1		U	
molecularWeight	Mixture average molecular weight (relative molar mass)	I			U	

Identifier	Meaning	type of property	dimensionality	units	basis	overall
osmoticCoefficient	A measure of water activities, defined as, $\phi = - n_W \ln (x_W f_W)/(n_S \sum \nu_i)$ where, n_W is the moles of water; n_S is the moles of solute; x_W is the mole fraction of water; f_W is the symmetric activity coefficient of water; ν_I is the stoichiometric coefficient of component i.	I			U	
pH	pH	I			U	
pOH	pOH	I			U	
phaseFraction	The molar (or mass) fraction of the fluid that is in the specified phase	I			mole/mass	
pressure	Pressure	I		Pa	U	Y
speedOfSound	Thermodynamic speed of sound *w*, where $w^2 = (C_p / C_v)(v / M\beta_T)$ see [13]	I		m/s	U	
temperature	Temperature	I		K	U	Y
thermalConductivity	Thermal conductivity	I		W/(m K)	U	
totalFlow	Matter flow of a Phase or the overall mixture	E		mol/s	mole/mass	Y
viscosity	Viscosity	I		Pa s	U	
volume	Volume	E		m^3	mole/mass	Y

7.5.6 Non-constant two-phase properties

See section 7.5.7 for more information on entries in this table.

Identifier	meaning	type of prop-erty	dimens ionality	units	basis
kvalue	Ratio of fugacity coefficients for a pair of phases defined as follows: $K_i = \phi_{i2} / \phi_{i1}$ where ϕ_{i1} is the fugacity coefficient of compound i in phase 1 and ϕ_{i2} is the fugacity coefficient in phase 2	I	1		U
logKvalue	Natural logarithm of K-value	I	1		U
surfaceTension	Interfacial tension for a pair of phases	I		N/m	U

7.5.7 Notes

IDENTIFIERS

The identifiers in sections 7.5.5 and 7.5.6 include most of those used in previous versions of this specification but there are some changes. Partial molar properties no longer have their own identifiers because they can be identified as mole number derivatives of another property. The Gibbs free energy and Helmholtz free energy are now identified by their recommended names of Gibbs energy and Helmholtz energy. The 'energy' identifier has been replaced by 'internalEnergy'. The previous identifier of heatCapacity is changed to heatCapacityCp and the corresponding heatCapacityCv has been added. The identifiers 'dewPointPressure/Temperature' and 'boilingPointTemperature' have been removed because these are the results of Equilibrium Calculations rather than phase properties. They should be evaluated using the CalcEquilibrium method of the ICapeThermoEquilibriumRoutine interface.

New property identifiers enthalpyF, entropyF, enthalpyNF and entropyNF have been added. The enthalpy and entropy identifiers are still supported but it is not specified whether or not the properties include the enthalpy/entropy of formation. However the properties must always correspond to the 'NF' variant or always correspond to the 'F' variant. It is not permissible to switch between the two.

TYPE OF PROPERTY

This column shows whether the property is extensive: **E**, or intensive: **I**. An extensive property, like the volume, depends directly on the quantity of material involved. By contrast, an intensive property, such as the temperature or viscosity, is independent of the quantity of matter making up the system

DIMENSIONALITY

The dimensionality of the property indicates whether it has a scalar, vector or higher-rank value.

For properties that are scalar quantities no entry is made.

1 indicates a vector quantity (rank 1 tensor) where the number of values is equal to the number of compounds. Values are returned as a 1-dimensional array in the same order as the compounds.

2 indicates a rank 2 tensor quantity where the number of values is equal to the number of compounds squared. The components of some quantity **A** are returned as a sequence of values in the order: $\{a_{11}, a_{21, \ldots}\ a_{n1}, a_{12}, a_{22, \ldots}\ a_{n2}, \ldots, a_{1n}, a_{2n, \ldots}\ a_{nn}\}$. This convention is extendable to higher-rank quantities.

BASIS AND UNITS

The 'Basis' column shows the basis settings that may be used for each property. **U** means UNDEFINED which is used when a molar or mass basis is not applicable.

For properties where only one basis is applicable, the units are fixed.

Extensive thermodynamic properties (enthalpy, entropy, volume etc.) may be expressed on a molar or mass basis. The units shown must be changed to match the Basis setting: for a molar basis the unit is divided by 'mol', for a 'mass' basis the unit is divided by 'kg'. For example, the property 'enthalpy' with a basis of 'mole' has units of J/mol and with a basis of 'mass' it has units of J/kg.

For intensive properties where there is a choice of 'mole' or 'mass' basis, the unit shown corresponds to the molar basis and 'kg' must be substituted for 'mol' if a mass basis is used.

OVERALL

Properties which are allowed in the GetOverallProp or SetOverallProp methods of the ICapeThermoMaterial interface, are indicated by a **Y** entry in this column.

Overall properties are used as inputs to equilibrium calculations: the overall composition and the flash constraints must be set on a material object before requesting an equilibrium calculation. In addition, the equilibrium calculation must set overall pressure and temperature for the overall phase, if not part of the flash specification.

Overall properties (total flow and composition, or compound flows, temperature, pressure, …) are the ones that are typically transported in streams in a flowsheet. Unit operations can assume that feed streams are in equilibrium, and that the flow rate of the feeds are available. Upon calculation, unit operation must perform an equilibrium calculation on the product streams, as well as set the flow rate.

Calculation of overall properties cannot be requested on a Property Package by any of its clients. The client needs to obtain the necessary bits of information from a Property Package that enable the calculation overall properties, i.e. the client must iterate over all present phases, and ask for the calculation by the Property Package of the property for each phase. The overall is calculated from the contribution of each phase. For extensive properties, the overall value generally follows from summation of the phase fractions times the phase properties; such summation being performed by the client. Typically the overall property is desired at phase equilibrium; in this case, the client must request the phase equilibrium calculation before calculating the overall property.

PHASE ORDER FOR TWO-PHASE PROPERTIES

Some of the two-phase properties listed in section 7.5.6 and all the composition derivatives depend on the interpretation of phase order. When the definitions in section 7.5.6 refer to phase 1, this is the first phase in the *phaseLabels* argument of the CalcTwoPhaseProp method of the ICapeThermoPropertyRoutine interface. Phase 2 is the second phase in the *phaseLabels* argument.

FUGACITY

Section 7.5.5 includes four different identifiers that may be used to request information about the fugacity of a compound in a mixture (fugacity, fugacityCoefficient, logFugacity and logFugacityCoefficient). It is not the intention to imply that *all* of these properties should necessarily be supported by components that implement the ICapeThermoPropertyRoutine interface. The natural quantity that arises from most thermodynamic models of mixtures is the logFugacityCoefficient. Unlike the fugacity itself this quantity is also well-defined at zero concentration of a compound and the logarithmic form allows a wider range of numerical values to be represented. It is therefore recommended that when a component supports any of these properties at least the logFugcityCoefficient should be supported.

7.6 Derivatives

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

Derivative identifier	meaning	units
property.Dtemperature	derivative of property with respect to temperature with pressure and composition fixed	[property]/K
property.Dpressure	derivative of property with respect to pressure with temperature and composition fixed	[property]/Pa

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

Derivatives of two-phase properties are not equilibrium derivatives. That means that composition derivatives are evaluated independently for each phase keeping the temperature and pressure of both phases and the composition of the other phase fixed. Similarly a temperature (or pressure) derivative does not imply any change in the phase compositions or the pressure (or temperature).

For a two-phase property the derivatives with respect to temperature or pressure are the sum of the derivatives for each phase even if the temperatures or pressures in both phases are not the same.

7.6.1 Basis and Units

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

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

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

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

7.6.2 Number of values returned and order

The following rules apply.

- Dtemperature and Dpressure derivatives return the same number of values and in the same order as the corresponding property. For example, enthalpy.Dtemperature will return a single value, whereas fugacityCoefficient.Dtemperature will return a vector of values.
- Dmoles derivatives of scalar properties return a vector of values with the same number of elements as there are compounds in the mixture. For example, enthalpy.Dmoles with a basis of Mole will return a vector of the partial molar enthalpies containing as many values as there are compounds.
- Dmoles derivatives of vector-valued mixture properties are, conceptually, a sequence of vectors. The first vector returns the derivatives of all properties with respect to the mole number of the first compound. The second vector returns the derivative with respect to the mole number of the second compound, and so on. The actual representation of these values will be a single sequence that contains all the values of these vectors as a one-dimensional array.
  - For example, activity.Dmoles with respect to all compounds will return the following values:

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

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

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

- In general, Dmoles derivatives of a rank-m quantity are returned as a sequence of values containing the components of a rank-m+1 quantity.
- For two-phase properties the mole number derivatives are returned w.r.t. the composition of phase 1 followed by derivatives w.r.t. phase 2. The phase order is defined in the section Phase order for two-phase properties in 7.5.7.
- The remarks for Dmoles derivatives in this section also apply to DmoleFraction derivatives.

8. Implementation of the Persistence Interface

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

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

Therefore, it is recommended for thermodynamic software components (such as Property Packages) to implement the Persistence Interfaces as described in Persistence Common Interface document [19].

The following methods should be implemented: Load, Save, GetClassID, IsDirty, GetSizeMax and, optionally, InitNew. Implementation of persistence also allows modification of the thermodynamic software component's configuration while it is part of the simulation case, e.g. via the Edit method of ICapeUtilities. Note that Property Packages should implement ICapeUtilities; as such, it is possible for a Property Package to have Edit functionality as well as to expose parameters [18]. For Property Packages allowing Edit functionality or modification by means of changing parameter values, persistence is a prerequisite, as otherwise the modifications will be lost between simulation sessions.

Implementation tips:

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

8.1 Initialization and termination

A Property Package Manager software component is a top-level CAPE-OPEN object (or *PMC primary object* as outlined in the Methods & Tools integrated Guidelines [8] documentation) and should therefore implement the ICapeUtilities interface. For a stand-alone Property Package, this also holds. The simulation environment is expected to call Initialize and Terminate on such objects.

A Property Package that is created from a Property Package Manager is, in accordance with the definition of *PMC primary objects* and *PMC secondary object* in the also a top-level CAPE-OPEN object. Therefore, it is expected of the Property Package to implement ICapeUtilities, and it is expected from the simulation

environment that initialization and termination (and possibly persistence) are performed for such Property Packages as if it were a stand-alone Property Package.

9. Bibliography

1. "Open Interface Specification Thermodynamic and Physical Properties v1.0", CAPE-OPEN, 2002.
2. "Unit Operation Specification", CAPE-OPEN, 2001.
3. "Open Interface Specification: Reactions Interface", CAPE-OPEN, 2003.
4. "Open Interface Specification: Identification Common Interface", CAPE-OPEN, 2003.
5. "Open Interface Specification: Error Common Interface", CAPE-OPEN, 2003.
6. "Design Patterns", Gamma, Helm, Johnson, Vlissedes, Addison-Wesley, 1995.
7. "Open Interface Specification: Petroleum Fractions Interface", CAPE-OPEN, 2003.
8. "Methods & Tools Integrated Guidelines", CAPE-OPEN, 2003.
9. http://www.daylight.com/smiles/f_smiles.html , see also: D. Weininger, "SMILES 1.Introduction and Encoding Rules", J. Chem. Inf. Comput. Sci., 28, 31 (1988).
10. Hill, J. "On A System Of Indexing Chemical Literature; Adopted By The Classification Division Of The U. S. Patent Office", Am. Chem. Soc. 22(8), 478-494 (1900).
11. "Thermodynamics and Physical Properties v1.1", version 2.22, 23 October 2006.
12. "Errata and clarifications, Thermodynamics and Physical Properties v1.1", version 1.11, 12 July 2008.
13. Rowlinson, J.S. and Swinton, F. L., "Liquids and Liquid Mixtures", Butterworth Scientific (1982)
14. "Nomenclature of Organic Chemistry, Sections A, B, C, D, E, F, and H", Pergamon Press, 1979. Edited by J Rigaudy and S P Klesney. [ISBN 0-08-022369-9]
15. "A Guide to IUPAC Nomenclature of Organic Compounds, Recommendations 1993", Blackwell Scientific Publications, 1993. Edited by R Panico, W H Powell and J C Richer. [ISBN 0-632-03488-2]
16. "Nomenclature of Inorganic Chemistry: IUPAC Recommendations 2005", Royal Society of Chemistry, 2005. Edited by N G Connelly and T Damhus (with R M Hartshorn and A T Hutton) [ISBN 0-85404-438-8].
17. "Open Interface Specification: Utilities Common Interface", CAPE-OPEN, 2003.
18. Open Interface Specification: Parameter Common Interface, CAPE-OPEN 2003.
19. Open Interface Specification: Persistence Common Interface, CAPE-OPEN 2003