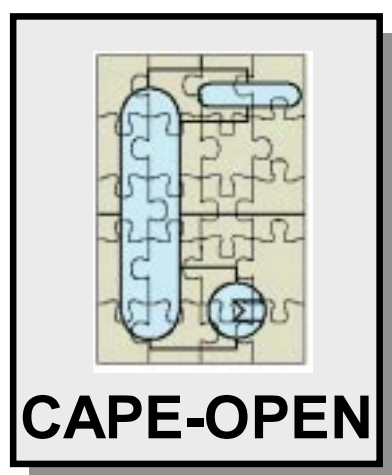


OPEN INTERFACE SPECIFICATION THERMODYNAMIC AND PHYSICAL PROPERTIES



Thermo Work Package

CO-THRM-1 Version 1.07

IMPORTANT NOTICES

Disclaimer of Warranty

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

Copyright © 1999 CAPE-OPEN and project partners and/or suppliers. All rights are reserved unless specifically stated otherwise.

CAPE-OPEN is a collaborative research project established under BE 3512 "Industrial and Materials Technologies" (Brite-EuRam III), reference BRPR-CT96-0293.

Trademark Usage

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

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

Netscape Navigator is a registered trademark of Netscape Corporation.

Adobe Acrobat is a registered trademark of Adobe Corporation.

Visio is a registered trademark of Visio Corporation.

CAPE-OPEN Archival Information

Reference	CO-CTHRM-1 Version 1.07
Coordinated by	BASF
Date	30 September 1999
Number of Pages	
Version	Version 1.07
Filename	Thrsep99.doc
Developmental Editor(s)	Thermo Work Package
Copy Editor(s)	Thermo Work Package
Proofreading Editor(s)	Thermo Work Package

Contents

1.	INTRODUCTION	7
2.	PHYSICAL PROPERTIES INTERFACE MODEL & SPECIFICATION	9
2.1.	INTRODUCTION	9
2.2.	INTERFACE SPECIFICATION – DOCUMENT ORGANIZATION & PROCESS.....	9
2.3.	PROCESS OVERVIEW.....	9
2.3.1.	<i>Component Diagram</i>	9
2.3.2.	<i>Component Diagram – Use Case View</i>	10
2.3.3.	<i>Interface Diagram</i>	10
2.3.4.	<i>Entity Descriptions & Interface Glossary</i>	10
2.3.5.	<i>IDL Specification</i>	10
2.3.6.	<i>Code Examples</i>	10
2.4.	COMPONENT DIAGRAM & SUPPORTING INTERFACES.....	11
2.4.1.	<i>Material Classes - Description</i>	11
2.5.	BRIEF DESCRIPTION OF CALCPROP & GETPROP STANDARD METHODS.....	14
2.6.	CAPE-OPEN CALLING PATTERN DESCRIPTION.....	15
2.7.	CAPE-OPEN CALLING PATTERN & MATERIAL OBJECT.....	16
2.8.	CAPE-OPEN USE CASE DRIVEN COMPONENT MODEL.....	17
2.8.1.	<i>Native Property Package Diagram</i>	18
2.9.	CAPE-OPEN THERMO INTERFACE DIAGRAM.....	19
2.9.1.	<i>Thermo Interface Model</i>	19
2.10.	CODE SAMPLE OF CAPE-OPEN CALLING PATTERN & MATERIAL OBJECT.....	21
2.10.1.	<i>Declare Material Object</i>	21
2.10.2.	<i>Example 2: Calling a flash and following calling of viscosity:</i>	22
2.11.	CAPE-OPEN THERMO WORKPACKAGE INTERFACE GLOSSARY.....	25
2.11.1.	<i>IcapeThermoMaterialTemplate</i>	25
2.11.2.	<i>ICapeThermoMaterialObject</i>	27
2.11.3.	<i>ICapeThermoSystem</i>	45
2.11.4.	<i>ICapeThermoPropertyPackage</i>	48
2.11.5.	<i>ICapeThermoCalculationRoutine</i>	56
2.11.6.	<i>ICapeThermoEquilibriumServer</i>	60
2.12.	CAPE-OPEN THERMO WORKPACKAGE MIDL.....	64
2.12.1.	<i>interface ICapeThermoSystem : IDispatch</i>	64
2.12.2.	<i>interface ICapeThermoPropertyPackage : IDispatch</i>	64
2.12.3.	<i>interface ICapeThermoMaterialTemplate : IDispatch</i>	66
2.12.4.	<i>interface ICapeThermoMaterialObject : IDispatch</i>	66
2.12.5.	<i>interface ICapeThermoCalculationRoutine : IDispatch</i>	69
2.12.6.	<i>interface ICapeThermoEquilibriumServer : IDispatch</i>	70
2.13.	CAPE-OPEN THERMO WORKPACKAGE CIDL.....	72
2.13.1.	<i>Base</i>	72
2.13.2.	<i>COSE</i>	72
2.13.3.	<i>Thermo System</i>	77
2.13.4.	<i>Calculation routine</i>	80
2.13.5.	<i>Equilibrium Server</i>	81
2.14.	CAPE-OPEN PROPERTIES LIST.....	83
2.14.1.	<i>Constant Properties</i>	83
2.14.2.	<i>Non-constant Properties (or Model Dependent Properties)</i>	85
2.15.	CAPE-OPEN PHASE LIST.....	88
2.15.1.	<i>Phase Details</i>	88
2.16.	CAPE-OPEN FLASH TYPE LIST.....	89
2.16.1.	<i>Flash Type Details</i>	89
2.17.	CALCULATION TYPE LIST.....	90
2.17.1.	<i>Calculation Type Details</i>	90

3. PHYSICAL PROPERTIES USE CASES.....	91
3.1. INTRODUCTION	91
3.2. SUMMARY OF RELEVANT USER REQUIREMENTS.....	93
3.2.1. <i>Interchangeable components</i>	93
3.2.2. <i>Other requirements</i>	97
3.3. GLOSSARY OF TERMS USED	99
3.4. USE CASE ACTORS.	103
3.5. THE USE CASES.....	105
3.5.1. <i>Component Installation Package</i>	105
3.5.2. <i>Simulator Package</i>	111
3.5.3. <i>Flowsheet Setup Package</i>	115
3.5.4. <i>Physical Properties System</i>	121
3.5.5. <i>Runtime Package</i>	149
3.5.6. <i>Neutral File Interface</i>	169

1. Introduction

This document is divided into two parts.

The first part, provides a practical guidelines for using and implementing CAPE-OPEN Thermodynamic interfaces. And in a second part attempts to develop a set of Use Cases that encapsulate the Users Requirements.

2. Physical Properties Interface Model & Specification

2.1. Introduction

The Thermodynamic Interface Specification is intended to provide practical guidelines for using and implementing CAPE-OPEN Thermodynamic interfaces. The purpose of the CAPE-OPEN Thermo interfaces is to allow for easier integration to existing commercial thermodynamic systems. These interfaces have been created, documented and implemented with the help of some very important Thermo documents. These documents include but are not limited to” Proposal for Thermo, BASF & Thermo Workpackage Team, Modeling of Matter, Laurent Jourda - INP Toulouse, “CDD2” - Requirements Document, “Thermo Use Case Document,” Bill Johns - Quantisci, “Comment of Streams, Thermo & Units,” Marquardt, Bogusch, Schneider, Von Wedel - Aachen. Also Sergi Sama from Hyprotech has contributed much appreciated understanding of Material Object.

In addition the interfaces specified in this document have been refined with reviews from the Thermo Workpackage team. The first sections provide an overview of the interfaces and successive sections provide greater detail. Also included with the specification are detailed CORBA IDL and Microsoft IDL. Finally, code examples are sited on how the interfaces are constructed and used.

2.2. Interface Specification – Document Organization & Process

2.3. Process Overview

In addition to the documents mentioned previously, the Interface Specification was driven and modified based upon the practical development of a Thermo Prototype, initially demonstrated at the CAPE-OPEN Mid-term meeting in Lyon on March 17th, 1998. The functionality of the Thermo Interface Specification is captured in the Thermo Use Cases. The Thermo Use Cases highlight three main areas of functionality: Property Package Management, Material Template Management, and Flowsheet Definition. This document transitions the details of the interface to the physical design view captured in both the IDL and Code Examples.

2.3.1. Component Diagram

The Component Diagram depicts the components and their associated interfaces. The associations that are drawn from the exposed interface to the internal components depict the usage of the interfaces from a component-based and implementation view. This diagram is not intended to, nor implies, how these components are actually implemented. This Component Model represents the components that expose interface functionality. The usage and functional flow of these components are captured in the above mentioned, “Thermo Use Case Document.” The execution of these components and the internal architecture of these components are proprietary to the individual vendor’s simulation environments.

2.3.2. Component Diagram – Use Case View

The component diagram – use case view depicts the CAPE-OPEN components as they are segregated across the functional Use Case View. This diagram clarifies the functional/actor relationship to the components and interfaces.

2.3.3. Interface Diagram

The Interface Diagram Model represents the pictorial overview representation of the Thermo Interface specifications. This representation is intended to provide a graphical depiction of the interfaces for a summary view. Note: the associations depicted on the diagram describe logical associations. These logical associations do not imply traversal paths or implementation. This interface diagram describes the details of the interfaces exposed on the preceding Component Diagram & Component Diagram – use case view.

2.3.4. Entity Descriptions & Interface Glossary

This appendix glossary is intended to give a brief description of each of the key CAPE-OPEN Thermo Interfaces as well as parameter descriptions.

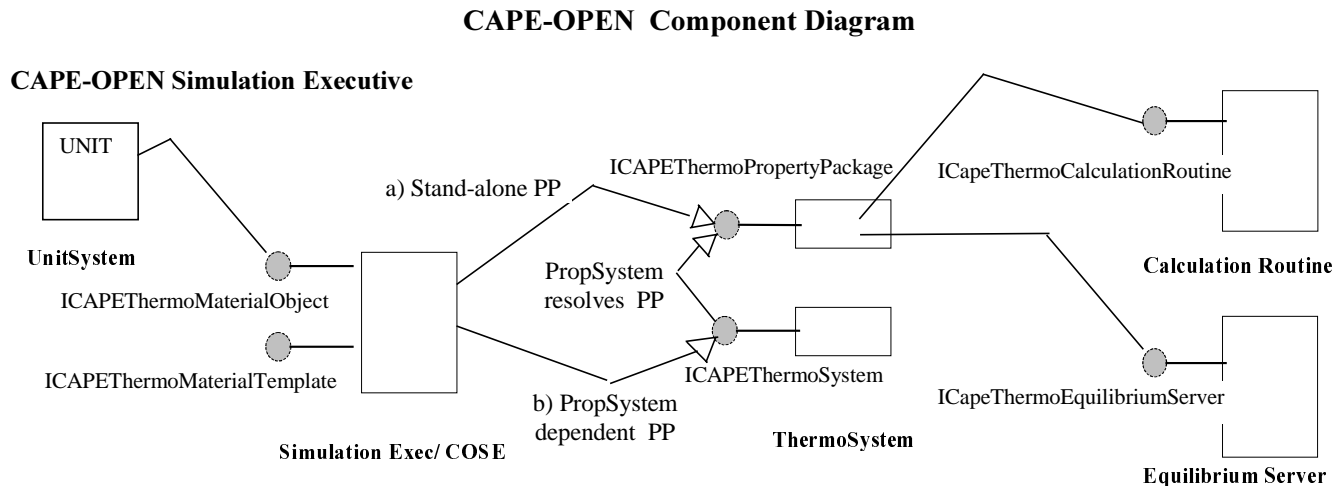
2.3.5. IDL Specification

The detailed interface specification is captured within both CORBA IDL (CIDL) and Microsoft IDL (MIDL).

2.3.6. Code Examples

Finally, code examples are included to show how the executions of interfaces work from a practical implementation view. This implementation view, although part of this document, is intended to provide the foundation of a separate document focused on the practical step by step aspects of implementing and using CAPE-OPEN interfaces.

2.4. Component Diagram & Supporting Interfaces



The component model depicts the view of the CAPE-OPEN Thermo Package. The CAPE-OPEN Component Diagram shows the interfaces supported by each of the components. The associations, represented by the lines from the components to the interfaces, are also detailed. The way in which these associations of the Component Diagram are implemented is proprietary to the component/simulation vendor. In this diagram are shown the two ways that any COSE can use to instantiate a Property Package:

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

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

Each white arrow represents the interface the COSE gets.

As independent software components these associations could be different depending on functional flow. The associations depicted in the Component Diagram reflect the usage of the components dictated from the Thermo Use Cases and CDD2 documents.

2.4.1. Material Classes - Description

The creation of the Material Interfaces is consistent with **Figure 3-1**. This diagram documents the implementation rather than the CAPE-OPEN interface view. The Material Template defines the characterization of a material, and the Material Object defines an instance of material. Material Objects are created from Material Templates. The Material Template definition consists of:

- A Component List ,
- A Phase Assumption,

- Reference to CAPE-OPEN property package,
- List of custom attributes for pseudo components

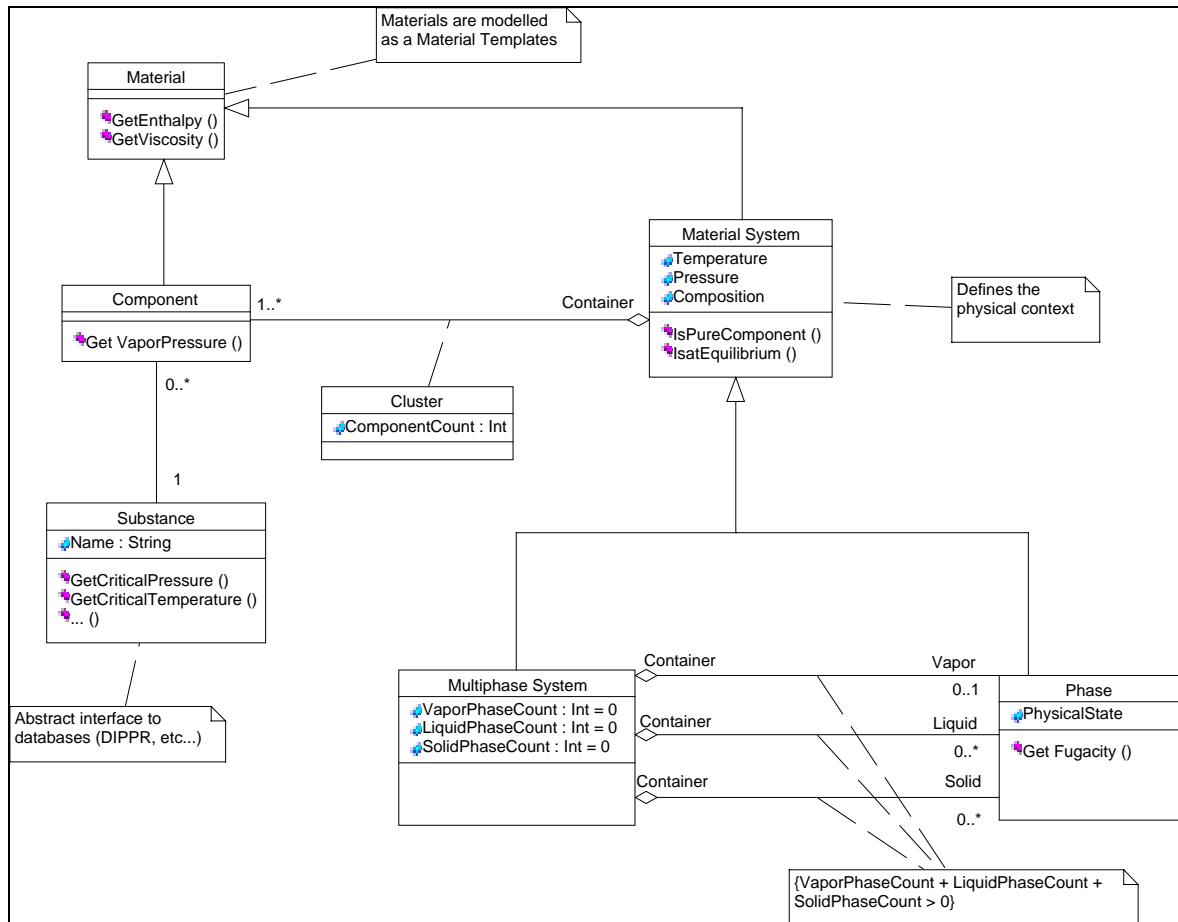


Figure 3-1: UML Material Class Diagram

Figure 3-1 splits components into components and substances. Substances are the actual species, components are instances of the substances used in the simulation. Since we can have more than one component based on the same concept of a substance. All parameters are ultimately associated with components, since any substance parameter could be overridden for a component. The end user of the Material Object is only concerned about components. The substance and database branch of 8-2 is really implemented *internal* to a property system.

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

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

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

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

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

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

```
CapeError GetUniversalConstants(in CapeArrayString constantList, out  
CapeArrayDouble propVals);
```

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

2.5. Brief Description of CalcProp & GetProp Standard Methods.

□ CalcProp

The CalcProp method on both the material object and the property package interfaces allows a single call to be made across system level components. CalcProp takes as parameters a list of properties (allowing the delegation of multiple property calculations to the Thermo Sever), a list of phases to calculate for, and the CalculationType (pure or mixture).GetProp

The GetProp method allows the retrieval of qualified results from the material object. These results can be qualified for a single property and these qualifications include: Phase, Components, Calculation Type (pure or mixture), and basis (molar or mass).

2.6. CAPE-OPEN Calling Pattern Description

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

In the case of the CAPE-OPEN Calling Pattern the contracts between the Simulation Executive, Unit, and Thermo systems are separated from the software calling mechanism. This allows the following advantages:

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

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

2.7. CAPE-OPEN Calling Pattern & Material Object

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

- Step 1: Declare Material Object and set Independent Variables

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

- Step 2: Set Values

The client of the material object adds properties and corresponding values.

- Step 3: Calculate

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

CalcEquilibrium
CalcProp

- Step 4: Get Results

After results are calculated, the values are then retrieved from the Material Object using the generic Material Object GetProp method. Results are further qualified for phase, components, calculation type, and basis. Property results are in SI units.

Specific Code Examples of this Pattern are included in this document and are provided by Werner Drewitz of BASF.

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

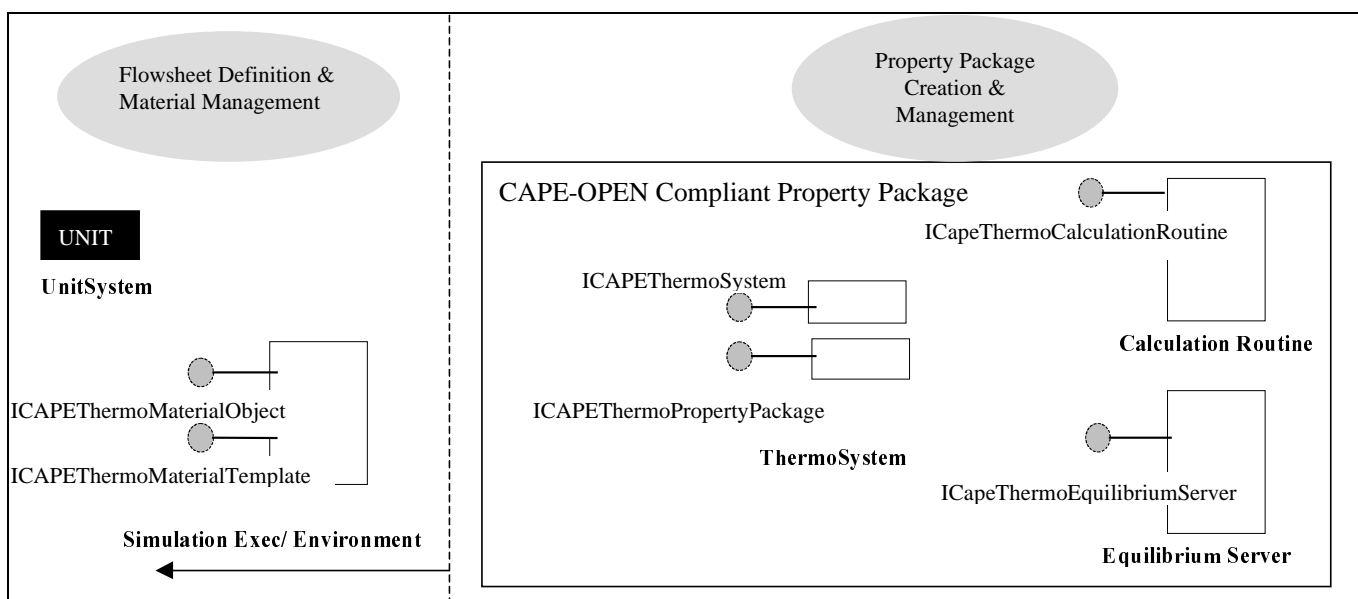
Alternatively, creating a separate method for every property retrieved and calculated leads to a number of issues and problems that are further documented in [Anti Patterns Refactoring Software, Architectures...](#)

by Mowbray, Malveau, McCormick, Brown. In fact, the direct calling alternative to the CAPE-OPEN calling pattern is detailed as a common negative pattern. (See: Spaghetti Code Anti Pattern.)

2.8. CAPE-OPEN Use Case Driven Component Model

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

CAPE-OPEN Component Model – Use Case Driven



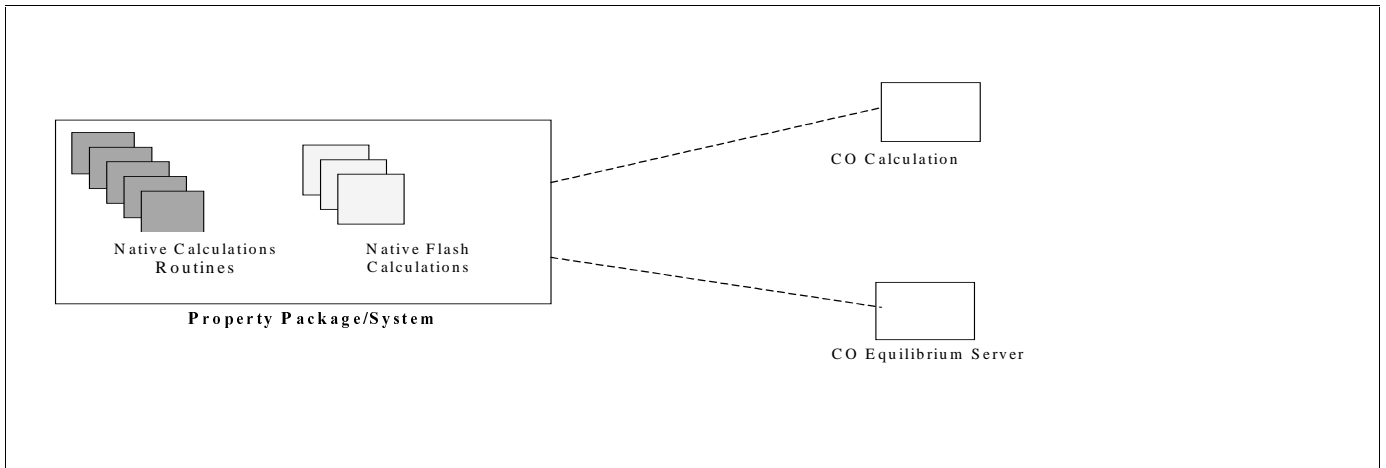
The attached model depicts the CAPE-OPEN components and interfaces in the context of CAPE-OPEN defined workflow. Property Package Creation & Management is the process by which existing Property Packages are made CAPE-OPEN compliant and properly registered. Flowsheet Definition requires that materials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

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

Equilibrium Servers and Calculation Routines, along with the proprietary structure of the property package provide the structure for a CAPE-OPEN compliant property package.

2.8.1. Native Property Package Diagram

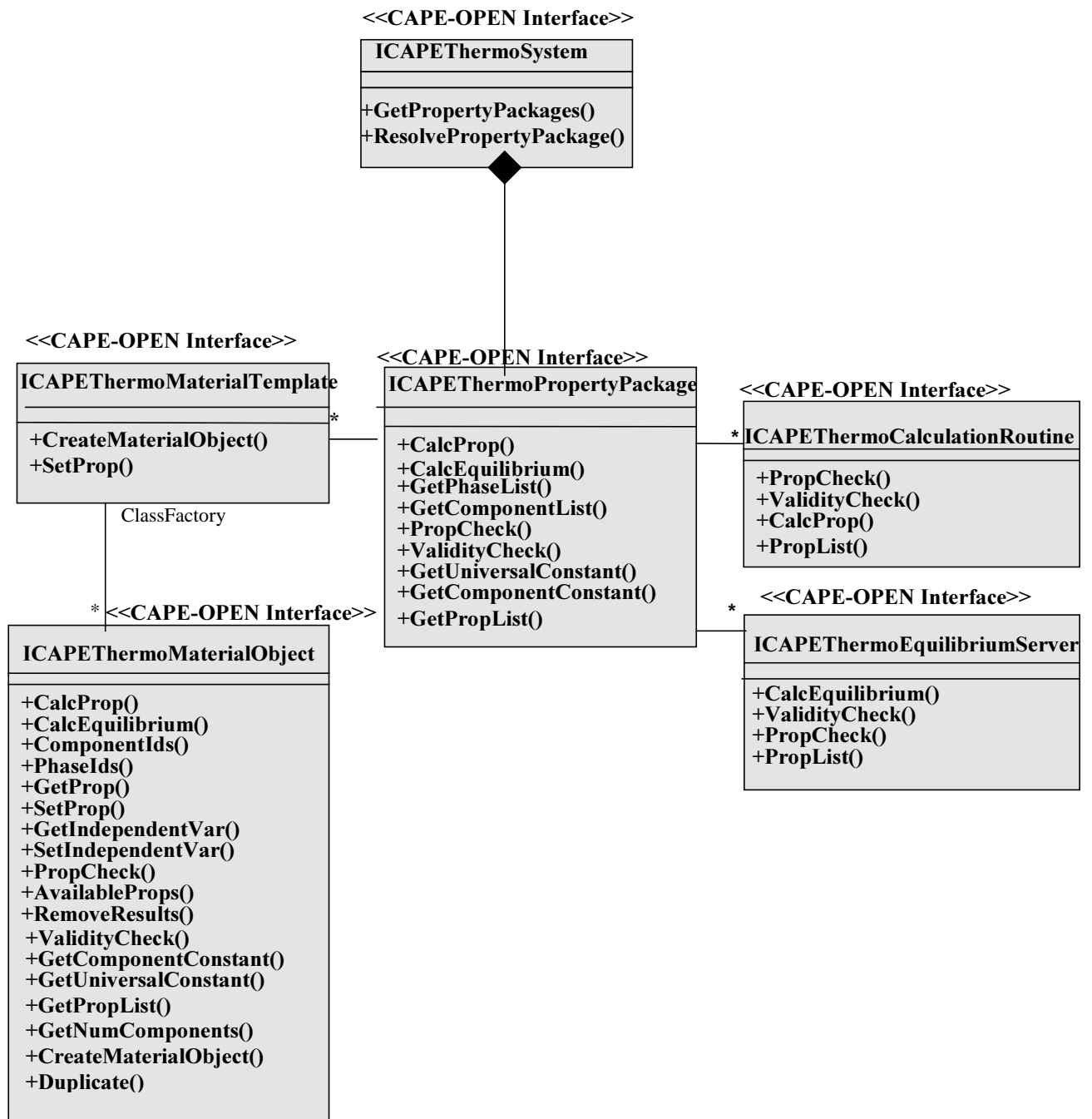
It is important to note that making a property package CAPE-OPEN compliant does not upset the native structure of a property package. The full capabilities of a commercial property system are still available in a CAPE-OPEN property package.



2.9. CAPE-OPEN Thermo Interface Diagram

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

2.9.1. Thermo Interface Model



A more detailed view can be found in the corresponding IDL and code samples. It is very important to note that the interface diagrams expose the necessary interfaces for using plug and play components. These diagrams do not imply the internal traversal path for how these interfaces are executed. Note: The results are stored in the Material Object and accessed through the GetProp method.

2.10. Code Sample of CAPE-OPEN Calling Pattern & Material Object

The following code example is detailed for the Material Object. However, the pattern by which values are set, calculated and retrieved is consistent for all Thermo System components.

Example 1: Calling of liquid enthalpy property of a mixture

2.10.1. Declare Material Object

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

□ Step 1: Set Values

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

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

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

Strncpy(phaseQualifier, "Liquid"); // set phase qualifier

Imo.SetProp("temperature", phaseQualifier, NULL, NULL, NULL, T);
Imo.SetProp("pressure", phaseQualifier, NULL, NULL, NULL, P);
// set temperature and pressure on the material object
Imo.SetProp("molFraction", phaseQualifier, NULL, NULL, NULL, F);
// set liquid composition on the material object
```

□ Step 2: Calculate Mixture Property

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

Strncpy(calculationType, "Mixture"); // set calculation type

Strncpy(properties[0], "specificEnthalpy"); // set property name

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

□ Step 3: Get Results

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

Strncpy(basisQualifier, "molar"); // set basis qualifier

Imo.GetProp("specificEnthalpy", phaseQualifier, NULL,
calculationType, basisQualifier, val);
//get property enthalpy in the liquid phase
```

2.10.2. Example 2: Calling a flash and following calling of viscosity:

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

□ Step 1: Set Values

```
CapeString phaseQualifier;

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

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

Strcpy(phaseQualifier, "Overall"); // set phase qualifier

// set temperature, pressure and composition on the material object

Imo.SetProp("temperature", phaseQualifier, NULL, NULL, NULL, T);
Imo.SetProp("pressure", phaseQualifier, NULL, NULL, NULL, P);
Imo.SetProp("molFraction", phaseQualifier, NULL, NULL, NULL, F);
```

□ Step 2 : Calculate Flash

```
CapeString flashTypeQualifier;

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

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

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

□ Step 3: Calculate Viscosity

```
CapeString calculationType;
// set calculation type
Strcpy(calculationType, "Mixture");

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

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

□ Step 4: Get Results

```
CapeDoubleArray val;
//double created.
```

```
Imo.GetProp("viscosity", phaseQualifier, NULL, calculationType,  
basisQualifier, val);
```

```
//get property viscosity for the liquid phase from the  
//Material Object.
```


2.11. CAPE-OPEN THERMO WORKPACKAGE INTERFACE GLOSSARY

2.11.1. IcapeThermoMaterialTemplate

2.11.1.1 CreateMaterialObject

Interface Name IcapeThermoMaterialTemplate

Method Name CreateMaterialObject

Returns CapeError

Description

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

Arguments

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

2.11.1.2 SetProp

Interface Name IcapeThermoMaterialTemplate

Method Name SetProp

Returns CapeError

Description

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

Arguments

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

2.11.2. ICapeThermoMaterialObject

2.11.2.1 ComponentIds

Interface Name ICapeThermoMaterialObject

Method Name ComponentIds

Returns CapeError

Description

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

Arguments

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

2.11.2.2 Phaselds

Interface Name **IcapeThermoMaterialObject**

Method Name PhaseIds

Returns CapeError

Description

Returns the list of phases supported by the Material Object.

Arguments

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

2.11.2.3 GetUniversalConstant

Interface Name **IcapeThermoMaterialObject**

Method Name GetUniversalConstant

Returns CapeError

Description

Retrieves universal constants from the property package.

Arguments

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

2.11.2.4 GetComponentConstant

Interface Name **IcapeThermoMaterialObject**

Method Name GetComponentConstant

Returns CapeError

Description

Retrieve component constants from the property package.

Arguments

Name	Type	Description
[in] props	CapeVariant (String Array)	List of component constants
[in] compIds	CapeVariant (String Array)	List of component IDs for which constants are to be retrieved. NULL for all components in the Material
[out,retval] *propvals	CapeArrayDouble	Component Constant values returned from the property package for all the components in the Material Object

2.11.2.5 CalcProp

Interface Name **IcapeThermoMaterialObject**

Method Name CalcProp

Returns CapeError

Description

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

Arguments

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

2.11.2.6 GetProp

Interface Name **IcapeThermoMaterialObject**

Method Name GetProp

Returns CapeError

Description

This method is responsible for retrieving the results from calculations from the MaterialObject.

Arguments

Name	Type	Description
[in] property	CapeString	The Property for which results are requested from the MaterialObject.
[in] phase	CapeString	The qualified phase for the results.
[in] compIds	CapeVariant (String Array)	The qualified components for the results. NULL to specify all components in the Material Object. For mixture property such as liquid enthalpy, this qualifier is not required. Use NULL as place holder.
[in] calcType	CapeString	The qualified type of calculation for the results. (valid Calculation Types: Pure and Mixture)
[in] basis	CapeString	Qualifies the basis of the result (i.e., mass /mole). Default is mole. Use NULL for default or as place holder for property for which basis does not apply.
[out, retval] *results	CapeArrayDouble	Results vector containing property values in SI units arranged by the defined qualifiers.

2.11.2.7 SetProp

Interface Name **IcapeThermoMaterialObject**

Method Name SetProp

Returns CapeError

Description

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

Arguments

Name	Type	Description
[in] property	CapeString	The property for which the values need to be set.
[in] phase	CapeString	Phase, if applicable. Use NULL for place holder.
[in] compIds	CapeVariant (String Array)	Components for which values are to be set. NULL to specify all components in the Material Object. For mixture property such as liquid enthalpy, this qualifier is not required. Use NULL as place holder.
[in] calcType	CapeString	The calculation type. (valid Calculation Types: Pure and Mixture)
[in] basis	CapeString	Qualifies the basis (mole / mass).
[in] values	CapeVariant (Double Array)	Values to set for the property.

2.11.2.8 CalcEquilibrium

Interface Name **IcapeThermoMaterialObject**

Method Name CalcEquilibrium

Returns CapeError

Description

This method is responsible for delegating flash calculations to the associated thermo system.

Arguments

Name	Type	Description
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. NULL for no properties.

2.11.2.9 SetIndependentVar

Interface Name **IcapeThermoMaterialObject**

Method Name SetIndependentVar

Returns CapeError

Description

Sets the independent variable for a given Material Object.

Arguments

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

2.11.2.10 GetIndependentVar

Interface Name **IcapeThermoMaterialObject**

Method Name GetIndependentVar

Returns CapeError

Description

Returns the independent variables of a Material Object.

Arguments

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

2.11.2.11 PropCheck

Interface Name	IcapeThermoMaterialObject
Method Name	PropCheck
Returns	CapeError

Description

Checks to see if given properties can be calculated.

Arguments

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

2.11.2.12 AvailableProps

Interface Name **IcapeThermoMaterialObject**

Method Name AvailableProps

Returns CapeError

Description

Gets a list properties that have been calculated.

Arguments

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

2.11.2.13 RemoveResults

Interface Name **IcapeThermoMaterialObject**

Method Name RemoveResults

Returns CapeError

Description

Remove all or specified property results in the Material Object.

Arguments

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

2.11.2.14 CreateMaterialObject

Interface Name IcapeThermoMaterialObject

Method Name CreateMaterialObject

Returns CapeError

Description

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

Arguments

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

2.11.2.15 Duplicate

Interface Name IcapeThermoMaterialObject

Method Name Duplicate

Returns CapeError

Description

Create a duplicate of the current Material Object.

Arguments

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

2.11.2.16 ValidityCheck**Interface Name** **IcapeThermoMaterialObject****Method Name** ValidityCheck**Returns** CapeError

Description

Checks the validity of the calculation.

Arguments

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

2.11.2.17 GetPropList

Interface Name	IcapeThermoMaterialObject
Method Name	GetPropList
Returns	CapeError

Description

Returns list of properties supported by the property package and corresponding CO Calculation Routines.

Arguments

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

2.11.2.18 GetNumComponents

Interface Name **IcapeThermoMaterialObject**

Method Name GetNumComponents

Returns CapeError

Description

Returns number of components in Material Object.

Arguments

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

2.11.3. ICapeThermoSystem

2.11.3.1 GetPropertyPackages

Interface Name ICapeThermoSystem

Method Name GetPropertyPackages

Returns CapeError

Description

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

Arguments

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

2.11.3.2 ResolvePropertyPackage

Interface Name	IcapeThermoSystem
Method Name	ResolvePropertyPackage
Returns	CapeError

Description

Resolves referenced property package to a property package interface.

Arguments

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

2.11.3.3 GetPhaseList

Interface Name **IcapeThermoPropertyPackage**

Method Name GetPhaseList

Returns CapeError

Description

Provides the list of the supported phases.

Arguments

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

2.11.4. ICapeThermoPropertyPackage

2.11.4.1 GetComponentList

Interface Name **ICapeThermoPropertyPackage**

Method Name GetComponentList

Returns CapeError

Description

Returns the list of components of a given property package..

Arguments

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

2.11.4.2 GetUniversalConstant

Interface Name **IcapeThermoPropertyPackage**

Method Name GetUniversalConstant

Returns CapeError

Description

Returns the values of the Universal Constants.

Arguments

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

2.11.4.3 GetComponentConstant

Interface Name **IcapeThermoPropertyPackage**

Method Name GetComponentConstant

Returns CapeError

Description

Returns the values of the Component Constants on the Material Object.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The material object.
[in] props	CapeVariant (String Array)	The list of properties.
[out,retval] *propvals	CapeArrayDouble	Component Constant values.

2.11.4.4 CalcProp

Interface Name **IcapeThermoPropertyPackage**

Method Name CalcProp

Returns CapeError

Description

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

Arguments

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

2.11.4.5 CalcEquilibrium

Interface Name **IcapeThermoPropertyPackage**

Method Name CalcEquilibrium

Returns CapeError

Description

Method responsible for calculating/delegating flash calculation requests.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	The MaterialObject
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. NULL for no properties.

2.11.4.6 PropCheck

Interface Name **IcapeThermoPropertyPackage**

Method Name PropCheck

Returns CapeError

Description

Check to see if properties can be calculated.

Arguments

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

2.11.4.7 ValidityCheck

Interface Name **ICapeThermoPropertyPackage**

Method Name ValidityCheck

Returns CapeError

Description

Checks the validity of the calculation.

Arguments

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

2.11.4.8 GetPropList

Interface Name **IcapeThermoPropertyPackage**

Method Name GetPropList

Returns CapeError

Description

Returns list of Thermo System supported properties.

Arguments

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

2.11.5. ICapeThermoCalculationRoutine

2.11.5.1 CalcProp

Interface Name ICapeThermoCalculationRoutine

Method Name CalcProp

Returns CapeError

Description

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

Arguments

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

2.11.5.2 PropCheck

Interface Name	IcapeThermoCalculationRoutine
Method Name	PropCheck
Returns	CapeError

Description

Checks to see if a given property can be calculated.

Arguments

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

2.11.5.3 PropList

Interface Name **IcapeThermoCalculationRoutine**

Method Name PropList

Returns CapeError

Description

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

Arguments

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

2.11.5.4 ValidityCheck

Interface Name **IcapeThermoCalculationRoutine**

Method Name ValidityCheck

Returns CapeError

Description

Checks the validity of the calculation.

Arguments

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

2.11.6. ICapeThermoEquilibriumServer

2.11.6.1 CalcEquilibrium

Interface Name ICapeThermoEquilibriumServer

Method Name CalcEquilibrium

Returns CapeError

Description

Calculates the equilibrium properties requested.

Arguments

Name	Type	Description
[in] *materialObject	CapeInterface	MaterialObject of the calculation
[in] flashType	CapeString	Flash calculation type.
[in] props	CapeVariant (String Array)	Properties to be calculated at equilibrium. NULL for no properties.

2.11.6.2 PropCheck

Interface Name **IcapeThermoEquilibriumServer**

Method Name PropCheck

Returns CapeError

Description

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

Arguments

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

2.11.6.3 ValidityCheck

Interface Name **IcapeThermoEquilibriumServer**

Method Name ValidityCheck

Returns CapeError

Description

Checks the reliability of the calculation.

Arguments

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

2.11.6.4 PropList

Interface Name **IcapeThermoEquilibriumServer**

Method Name PropList

Returns CapeError

Description

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

Arguments

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

2.12. CAPE-OPEN Thermo Workpackage MIDL

2.12.1. interface ICapeThermoSystem : IDispatch

```

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

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

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

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

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

```

2.12.2. interface ICapeThermoPropertyPackage : IDispatch

```

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

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

// Material Template and Material Object

```

```

#include "Cose.idl"

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

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

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

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

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

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

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

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

    // Check the validity of the given properties
    [id(8), helpstring("method ValidityCheck")]
    HRESULT ValidityCheck([in] CapeInterface materialObject,

```

```

[in] CapeVariant props,
[out, retval] CapeArrayThermoReliability
*valid);

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

#endif // _THERMOPROPERTYPACKAGE_IDL_

```

2.12.3. interface ICapeThermoMaterialTemplate : IDispatch

2.12.4. interface ICapeThermoMaterialObject : IDispatch

```

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

// Include GUIDs
#include "COGuids.idl"

// Fundamental types
#include "Fundamental.idl"

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

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

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

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

```

```

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

    // Set some property value(s)
    [id(2), helpstring("method SetProp")]
    HRESULT SetProp( [in] CapeString property,
                    [in] CapeVariant values);
};

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

// Material object interface
[
    object,
    uuid(CapeThermoMaterialObject_IID),
    dual,
    helpstring("ICapeThermoMaterialObject Interface"),
    pointer_default(unique)
]

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

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

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

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

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

```

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

#endif // _COSE_IDL_
```

2.12.5. interface ICapeThermoCalculationRoutine : IDispatch

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

// Include GUIDs
#include "COGuids.idl"

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

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

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

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

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

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

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

#endif // _CALCROUTINE_IDL_
```

2.12.6. interface ICapeThermoEquilibriumServer : IDispatch

```

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

// Fundamental types
#include "Fundamental.idl"

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

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

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

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

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

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

#endif // _EQSRVR_IDL_

```


2.13. CAPE-OPEN Thermo Workpackage CIDL

2.13.1. Base

```
// always put something like this in to prevent multiple processing
#ifndef CAPEBASE_IDL
#define CAPEBASE_IDL

module Cape {

    // Forward declarations

    interface Identification;

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

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

    // sequence definitions

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

    // interface definitions

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

#endif /* CAPEBASE_IDL */
```

2.13.2. COSE

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

#include "base.idl"

#ifndef THERMO_IDL
#define THERMO_IDL

module Cose {
```

```

/* Forward declaration of interfaces */

interface ICapeThermoMaterialObject;
interface ICapeThermoReliability;
interface ICapeThermoMaterialTemplate;

/* Type definitions and sequences */

typedef sequence<ICapeThermoReliability>
CapeThermoReliabilitySequence;

/* Exceptions */

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

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

Cape::CapeString message;
};

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

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

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

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

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

/* Interfaces */

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

interface ICapeThermoMaterialObject {

    // MATERIAL OBJECT DATA

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

```

```

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

    // MATERIAL OBJECT FUNCTIONS

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

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

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

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

    // props: identifiers of properties to calculate

    // phases: identifiers of phases for which to calculate

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

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

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

    props                the list of properties to be calculated

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

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

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

    */

```

<pre>void SetProp(in Cape::CapeString property, in Cape::CapeString phase, in Cape::CapeStringSequence compIds, in Cape::CapeString calcType, in Cape::CapeString basis, in Cape::CapeDoubleSequence values) raises(CapeThermoBoundsException, CapeThermoUnknownIdentifierException);</pre>	
<pre>/* This routine will set properties to calculate requested in the list properties for the given material object. The results are accessed using GetProp call. An error in The calculation is indicated with a ICapeBoundsError exception.</pre>	
props	the list of properties to be calculated
phases	the list of phases for which properties are to be calculated
calcType	"Mixture" or "Pure", indicating type of calculation
Values	Vector of property values. Allocated and freed by caller.
returns:	Nothing.
*/	
<pre>void CalcEquilibrium(in Cape::CapeString flashType, in Cape::CapeStringSequence props) raises(CapeThermoBoundsException, CapeThermoUnknownIdentifierException);</pre>	
<pre>/* This routine will calculate the equilibrium properties requested in the list props for the given material object by performing a flash calculation. An error in the calculation is indicated with a CapeThermoBoundsException or a CapeThermoUnknownIdentifierException</pre>	
props	the list of properties to be calculated
flashType	the type of flash calculation to be performed according to list of flashTypes in specification
*/	
<pre>void SetIndependentVar(in Cape::CapeStringSequence indVars, in Cape::CapeDoubleSequence values) raises (CapeThermoBoundsException, CapeThermoUnknownIdentifierException);</pre>	
/* Sets the values of independent variables	
indVars	identifier for the independent variables to set
values	values of the variables
throws	CapeThermoBoundsException if bounds are inappropriate CapeThermoUnknownIdentifierException if

```

        variable is not independent or unknown. */
    Cape::CapeDouble GetIndependentVar(in Cape::CapeStringSequence
indVars)
        raises (CapeThermoUnknownIdentifierException);

    /* retrieves values independent Variables

        indVars        identifier for the independent variables to set

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

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

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

        props        the list of properties to be calculated

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

    */

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

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

        props        the list of properties to validate
        returns        the reliability measure for each
        calculation

    */
};

interface ICapeThermoMaterialTemplate : Cape::ICapeIdentification {

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

    ICapeThermoMaterialObject CreateMaterialObject();

    // Creates a new material object

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

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

```

```

    to obj have been removed. */

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

```

2.13.3. Thermo System

```

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

#ifndef THERMO_SYSTEM_IDL
#define THERMO_SYSTEM_IDL

module ThermoSystem {

    interface ICapeThermoPropertyPackage;
    interface ICapeThermoSystem;

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

    typedef sequence<ICapeThermoSystem>
        CapeThermoSystemSequence;

    /* Interface definitions */

    interface ICapeThermoSystem : Cape::ICapeIdentification {

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

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

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

    interface ICapeThermoPropertyPackage : Cape::ICapeIdentification {

        // PROPERTIES OF THE CONFIGURED PHYSICAL PROPERTY PACKAGE

        Cape::CapeStringSequence GetPhaseList();

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

        void GetComponentList (out Cape::CapeStringSequence compIds,
                               out Cape::CapeStringSequence formulae,

```

```

        out Cape::CapeStringSequence names,
        out Cape::CapeDoubleSequence boilTemps,
        out Cape::CapeDoubleSequence molWeight,
        out Cape::CapeStringSequence casNo);

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

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

// calculate component properties

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

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

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

// This routine will calculate the equilibrium properties
// requested in the list props for the given material object
// by performing a flash calculation. The results are returned
// in ICAPEResultSet. An error in the calculation is indicated
// with a ICAPEBoundsError exception.
//
// matObj        the material object where the results are stored
// in
// flashType     the flash calculation to be performed
//
// props        the list of properties to be calculated

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

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

```

```

//          successfully be calculated).
//          valid[i] is the result for
//          property props[i]

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

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

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

Cape::CapeStringSequence GetPropList();

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

};

#endif

```

2.13.4. Calculation routine

```

#ifndef CALCROUTINE_IDL
#define CALCROUTINE_IDL

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

module CalculationRoutine {

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

interface ICapeThermoCalculationRoutine : Cape::ICapeIdentification
{

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

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

```

```

// CapeUnknownIdentifierException

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

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

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

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

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

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

```

2.13.5. Equilibrium Server

```

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

#ifndef EQSERVER_IDL
#define EQSERVER_IDL

module EquilibriumServer {

    interface ICapeThermoEquilibriumServer : Cape::ICapeIdentification
    {

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

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

```

```
//  
// props is a generalization of "property" that allows  
// calculation of multiple properties simultaneously.  
  
void  
PropList(out Cape::CapeStringSequence flashType,  
         out Cape::CapeStringSequence props,  
         out Cape::CapeStringSequence phases,  
         out Cape::CapeStringSequence calcType);  
  
// List of properties supported by the Calculation Routine.  
// and phases and types.  
  
Cape::CapeBooleanSequence  
PropCheck(in Cose::ICapeThermoMaterialObject matObj,  
          in Cape::CapeString flashType,  
          in Cape::CapeStringSequence props);  
  
// check that this external calculation routine can calculation  
// the property for this material object. Return TRUE if the  
// physical property can be evaluated; and otherwise, return  
// FALSE.  
  
Cose::CapeThermoReliabilitySequence  
ValidityCheck(in Cose::ICapeThermoMaterialObject matObj,  
             in Cape::CapeStringSequence props)  
raises (Cose::CapeThermoUnknownIdentifierException);  
  
// This function is used to check the reliability of the  
// property calculation at the converged point of the solution  
// strategy or presumably at any point along the way to  
// convergence. The return value indicates the reliability of  
// this foreign calculation on mobject.  
};  
};  
#endif
```

2.14. CAPE-OPEN Properties List

2.14.1. Constant Properties

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

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

The name of a component should be the same in the whole flowsheet. There is a need in every flowsheet calculation for a global component list! If one uses different Physical Packages it is very probable that there are different names used for the same component. So a translation list may help in which for each name of the global component list there is the name for the correspondent components in the used physical packages.

names	Meaning
molecularWeight	mass per mol
criticalTemperature	Critical Temperature
criticalPressure	Critical Pressure
criticalVolume	Critical Volume
criticalCompressibilityFactor	Critical Compressibility Factor
criticalDensity	Critical Density
acentricFactor	Pitzer Acentric Factor
dipoleMoment	Dipole Moment
parachor	Parachor
gyrationRadius	Radius of Gyration
associationParameter	Association-Parameter (Hayden-O'Connel)
diffusionVolume	diffusion volume

diffusionCoefficient	diffusion coefficient
vanderwaalsVolume	van der Waals Volume
vanderwaalsArea	van der Waals Area
energyLennardJones	Lennard-Jones Energy
lengthLennardJones	Lennard-Jones Length
normalBoilingPoint	Temperature at boiling point (1.01325 bar)
HeatOfVaporizationAtNormalBoilingPoint	Heat of Vaporization at boiling point (1.01325 bar)
normalFreezingPoint	Temperature of normal melting point (1.01325 bar)
heatOfFusionAtNormalFreezingPoint	Heat of Melting at melting point (1.01325 bar)
liquidDensityAt25C	Liquid Density at 25 C
liquidVolumeAt25C	Liquid Volume at 25 C
IdealGasGibbsFreeEnergyOfFormationAt25C	idealGasEnthalpyOfFormationAt25C
standardFormationEnthalpySolid	Standard Formation Enthalpy of Solid
standardFormationEnthalpyLiquid	Standard Formation Enthalpy of Liquid
standardFormationEnthalpyGas	Standard Formation Enthalpy of Gas
standardFreeFormationEnthalpySolid	Standard Free Formation Enthalpy of Solid
StandardFreeFormationEnthalpyLiquid	Standard Formation Enthalpy of Liquid
standardFreeFormationEnthalpyGas	Standard Formation Enthalpy of Gas
standardEntropySolid	Standard Entropy of Solid
standardEntropyLiquid	Standard Entropy of Liquid
standardEntropyGas	Standard Entropy of Gas

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

2.14.2. Non-constant Properties (or Model Dependent Properties)

names	Meaning
vaporPressure	Vapor,Pressure
sublimationPressure	Sublimation Pressure
meltingPressure	Melting Pressure
glassTransitionTemperature	GlassTransitionPressure
solidSolidPhaseTransitionTemperature	SolidSolidPhaseTransitionPressure
virialCoefficient	Second Virial Coefficient
surfaceTension	Surface Tension
expansivity	coefficient of linear expansion (Expansivity) $\frac{1}{L} \left. \frac{\partial L}{\partial T} \right $
compressibilityCoefficient	$\frac{1}{V} \left. \frac{\partial V}{\partial P} \right _T$
compressibilityFactor	Compressibility Factor $Z = \frac{PV}{RT}$
jouleThomsonCoefficient	$\left. \frac{\partial T}{\partial P} \right _H$
heatOfVaporization	
heatOfSublimation	
heatOfFusion	
heatOfSolidSolidPhaseTransition	(how are the solid phases specified?)
volumeChangeUponVaporization	
volumeChangeUponSublimation	
volumeChangeUponMelting	
volumeChangeUponSolidSolidPhaseTransition	(again: how are the solid phases specified?)
heatCapacity	Heat Capacity
idealGasHeatCapacity	Heat Capacity of ideal Gas

idealGasEnthalpy	Enthalpy of ideal Gas
excessEnthalpy	
excessEnergy	
excessGibbsFreeEnergy	
excessHelmholtzFreeEnergy	
excessEntropy	
excessVolume	
specificEnthalpy	specific Enthalpy
specificEnergy	specific Energy
specificGibbsFreeEnergy	specific Gibbs Free Energy
specificHelmholtzFreeEnergy	specific Helmholtz Free Energy
specificEntropy	specific Entropy
specificVolume	
partialMolarEnthalpy	
partialMolarEnergy	
partialGibbsFreeEnergy	
partialHelmholtzFreeEnergy	
partialMolarVolume	
compressibility	
viscosity	Viscosity
thermalConductivity	Thermal Conductivity
selfDiffusionCoefficient	
fugacity	Fugacity
fugacityCoefficient	Fugacity Coefficient
activity	Activity
activityCoefficient	Activity Coefficient
bubblePointPressure	

bubblePointTemperature

dewPointPressure

dewPointTemperature

names for state variables/global variables

temperature

pressure

volume

density

enthalpy

entropy

energy

freeEnergy

gibbsFreeEnergy

helmholtzFreeEnergy

molFraction

moles

mass

molflow

massflow

Derivatives:

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

property.Dtemperature derivative of property according to Temperature

property.Dpressure derivative of property according to Pressure

property.DmolFraction derivative of property according to mole fraction

property.Dmoles derivative of property according to mole numbers

2.15. CAPE-OPEN Phase List

2.15.1. Phase Details

Phase	Descriptions
Vapor	Vapor phase
Liquid	Liquid phase
Solid	Solid phase
VaporLiquid	Vapor/liquid
LiquidLiquid	Liquid/liquid
Overall	All phases

2.16. CAPE-OPEN Flash Type List

2.16.1. Flash Type Details

Flash Type	Descriptions
TP	Temperature-Pressure
PH	Pressure-Enthalpy
TH	Temperature-Enthalpy
TVF	Temperature-Vapor Fraction
PVF	Pressure-Vapor Fraction

2.17. Calculation Type List

2.17.1. Calculation Type Details

Calculation Type	Descriptions
Mixture	Mixture and partial properties
Pure	Pure component properties

3. Physical Properties Use Cases

3.1. Introduction

This second attempts to develop a set of Use Cases that encapsulate the Users Requirements presented in the CAPE-OPEN Conceptual Design report (October 1997). The aspects of the users requirements relevant to Physical Properties are summarized in a first part. A second part presents a brief glossary of terms given special meaning in this document. A fuller glossary is being developed to cover the whole CAPE-OPEN activity; terms in this document may be revised at a later date to ensure consistency between the glossaries. The Use Cases are presented in third part.

To set the scene, the context in which these use cases have been developed will be described. We suppose that there is a self-standing PHYSICAL PROPERTIES SYSTEM. The system will contain extensive correlations (methods) and data for a large number of properties and chemical components. The system will have its own PHYSICAL PROPERTIES EXECUTIVE which may provide a range of services. Specifically, it will provide tools to add methods, data and chemical components and to select from the information available to produce a PHYSICAL PROPERTIES PACKAGE that includes methods data and chemical components focussed on a specific application area. This package will be interfaceable with any CAPE-OPEN compliant simulator. Simulators will be able to accommodate one or more packages. The simulators will be able to interact with the packages to select all or a subset of the chemical species to form a MATERIALS TEMPLATE (called MATERIALS CONCEPT in some earlier CAPE-OPEN documents). The templates can be used to define the specific materials that are present at various locations in a process at any given time. In the Object Oriented materials concept introduced by INPT (Ref), each instance of a material is a material object derived from a specific materials template (or class).

More specifically, the Use Cases describe the steps in developing a properties package from a properties system and in developing a material template from a properties package.

A physical properties package may include one or more option sets, each of which provides a completely unambiguous way of computing any available property given an unambiguous definition of the material for which the property is required (for example, for a thermodynamic property, the composition and thermodynamic state). Commercial or in-house properties systems may have a number of alternative routines for calculating any given property (for example, vapour pressure). In order to create a physical properties package option set, it will be necessary to select only one routine for each property (or to set conditions defining when it should switch from one routine to another). The package may also optionally incorporate individual CAPE-OPEN compliant routines for specific properties (for example, a user may have a specific in-house method that is preferred for some properties). These individual routines may be used instead of methods provided in the system, or may provide methods for properties not available in the system (for example paramagnetism). The package will normally only include methods for properties that are relevant for the required simulation studies. Note that, where a package includes more than one option set, it may be given a title chosen by the specialist setting up the set, for example "low pressure set", "high pressure set". The ability to create option sets may be vendor

dependent; a system that can only produce one set within a package can also be CAPE-OPEN compliant.

The description above removes an ambiguity in the Users Requirements. It makes clear that a routine from one system (call it system "A") can only be used in another system (call it system "B"), if either the routine is interfaced to system A with a CAPE-OPEN compliant interface, or if the routine is provided stand-alone with a CO interface. As a consequence, it will not be possible initially to take methods from 2 separate proprietary properties systems and include them in one package. It will only be possible as the suppliers break their systems down into smaller components with the individual methods becoming CAPE-OPEN compliant. This approach allows the user requirement of being able to select methods from different systems to be achieved incrementally. This approach falls naturally into the developing component philosophy. Any alternative approach would require a complete rewriting of some of the existing properties systems and would consequently not be achievable. In the same way that calculation methods can be selected, added to, or replaced, so can data items (for example, correlated coefficients for selected methods) and chemical components. In this way a user could build up, for example, a package specific for chlorinated hydrocarbons in which they have quality assured the physical property values computed. This relatively compact package can then be interfaced with any simulation that the user wishes to undertake for that type of process. It is anticipated that a properties system may be able to create several properties packages, and that these packages will be interfaced to simulators via an interface to their parent systems, rather than by creating separate package components. Interfacing details do, however, depend on detailed software engineering decisions to be made in consultation with the major vendors.

A simulator executive may access several physical properties packages and, for each, select a subset of chemical components to give a material template. This template will incorporate the standard way of storing material stream information in the specific simulator and may apply the template to the whole process or to selected streams or units, depending on the options provided by the simulator. It is not a CAPE-OPEN requirement that all Properties Packages provides option sets. For those packages that do so, however, the executive will also select a specific option set in creating a materials template. By making different selections of components and/or option sets, several material templates can be generated from one properties package.

3.2. Summary of relevant User Requirements

Listed below are a selection of the over 60 “essential” users requirements that are most relevant to physical properties.

3.2.1. Interchangeable components

The CAPE-OPEN project shall define standard model interfaces to enable **the interchange of the following elements of simulators**:

- Thermodynamics/Physical Properties systems (**EL 1**)
- Data for chemical species [and addition of new chemical species] (**EL 2**)
- Individual Physical Properties methods within physical properties systems (**EL 3**)
- “Raw” physical property data sets (**EL 4**)
- Physical properties methods for as yet undefined properties (**EL 5**)

3.2.1.1 Types of simulators

This section is probably of less relevance to physical properties than to other aspects of CAPE-OPEN because the wide variety of simulator types all currently handle physical properties in a closely similar way.

The general **types of simulators** that shall be catered for are:

- Sequential Modular Simulators, such as Aspen Plus and Pro II (**TY 1**)
- Equation-Based Simulators, such as SPEEDUP (**TY 2**)
- Non-sequential Modular Simulators, such as Hysys (**TY 3**), and
- Modular Hierarchical Simulators (**TY 4**). (This requirement is probably not relevant to the properties interfaces, so will not be discussed in this document).

3.2.1.2 Material forms

The CAPE-OPEN interfaces shall be able to handle **material forms** consisting of:

- uniform fluids as molar compositions of specific chemical species (**PH 1**)
- uniform fluids defined as mixtures of pseudo-components (**PH 2**)
- uniform fluids with electrolytes (**PH 3**)
- uniform fluids with species in chemical equilibria such as dimers (**PH 4**)
- particulate systems (**PH 5**), and
- polymers and polymeric mixtures (**PH 6**).

3.2.1.3 Properties

The **physical properties** that shall be catered for within the CAPE-OPEN standards are:

- properties required for the calculation of vapour-liquid equilibria (**PP 1**) (for example, fugacities and activity coefficients)
- properties required for the calculation of liquid-liquid equilibria (**PP 2**)
- properties for the calculation of vapour-liquid-solid equilibria (**PP 3**) (for example, vapour pressures and activity coefficients)
- other thermodynamic properties for vapours, liquids and solids (**PP 4**) (Examples of specific properties that fall in this category include enthalpy, molar volume and density)
- transport properties (**PP 5**) (for example, viscosity, thermal conductivity, and diffusivity)
- as yet undefined properties of fluid systems (**PP 6**)
- properties of particulate systems(**PP 7**)
- properties of polymer systems (**PP 8**)

3.2.1.4 Error handling

These requirements are common to all the CAPE-OPEN work packages and a common approach should be adopted for all component types. In this draft, relatively little attention is paid to exceptions and error handling which will, however, be important in assuring the ultimate success of CAPE-OPEN. The **error recognition capabilities** that shall be provided are:

- The recognition of system differences (**ER 1**)
(for example, the recognition that an attempt has been made to interface incompatible systems, which if not trapped might give an obscure run-time error)
- The transmission of error messages (**ER 2**)
(Ability to transmit an error message [such as “value out of range” or “failure to converge”] to the simulation executive, rather than simply fail or return wrong values).
- The correction of incompatibilities (**ER 3**)
(Ability of the interface to make good minor incompatibilities between simulation systems [for example, providing missing enthalpy values or derivatives or discarding superfluous values or derivatives]. It is recognized that all simulators differ in some respects from all others. It must be possible to interchange CAPE-OPEN components between non-identical systems).
- The transmission of information relating to the reliability of calculations (**ER 4**)
(for example signalling a warning if a correlation is extrapolated beyond its valid range).

3.2.2. Other requirements

Interfaces to pdXi or some other neutral file shall be provided (**SI 5**) for transmitting information between systems (for example, initialising a dynamic simulation from the steady-state simulation of another system).

In terms of **performance**, when components from the same supplier are interfaced using the CAPE-OPEN standards there shall be a negligible (less than 10%) run-time overhead as compared to the “native” interface (**PE 1**). When interfacing components and systems from the same supplier, use of the CAPE-OPEN interface shall lose no functionality compared to the “native” interface (**PE 2**). It is noted that this requirement only applies to functionality that falls within the bounds defined by CAPE-OPEN standards. If a proprietary system provides a functionality that is not provided for within the CAPE-OPEN standards, that functionality may disappear when CAPE-OPEN standard interfaces are applied. There may be similar circumstances in which performance deteriorates.

In terms of **hardware and software**, the CAPE-OPEN standards shall cater for stand-alone PC's running MS-Windows (**HS 1**), stand-alone UNIX boxes (**HS 2**), all-PC networks with NT servers (**HS 4**), all-UNIX networks (**HS 5**), and heterogeneous networks with UNIX servers and PC clients (**HS 6**). There is also a strong desire to be able to include VMS (**HS3/7**). The standards shall also allow for links from/to FORTRAN 77 and FORTRAN 90 code (**HS 12**), from/to C code (**HS 13**), from/to C++ code (**HS 14**), and from/to Java code (**HS 15**). This requirement does not demand that every vendor must supply simulators that

run on every system that is covered by CAPE-OPEN standards. The requirement only demands that there must be standards applicable to all these platforms because some users will be employing them and will want to be able to use CAPE-OPEN compliant systems.

The CAPE-OPEN **end users** shall include FORTRAN Programmers (**US 1**), expert system programmers familiar with OLE/COM interfacing (**US 2**), programmers familiar with other languages (**US 3**), and simulator end users with no specific programming skills (**US 4**).

In terms of **maintenance**, the CAPE-OPEN standards shall provide a mechanism to allow new functionality to be added to the CAPE-OPEN interfaces beyond the duration of the project (**MA 1**). Adding such new functionality shall not require any re-coding of the components, interfaces or systems previously interfaced (**MA 2**). In addition, given clear descriptions of the native interfaces, it shall be possible for a user to write a “wrapper” to the CAPE-OPEN interface without requiring access to the source code of the component being wrapped (**MA 3**).

It was also stated that it was **DESIRABLE** to be able to handle porous **materials** with high specific surface (**PH 8**).

3.3. Glossary of Terms Used

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

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

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

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

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

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

Option Set. An Option Set is a Simple Properties Package. It is used when 2 or more Simple Properties Packages refer the same set of components and have the majority of methods and data in common. Some vendors then find it more efficient to combine the SPP's into one Properties Package; each SPP being referred to as an Option Set. The Option Sets are identified by simple titles, such as "high pressure" or "low pressure".

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

Physical Property Calculation Method. An equation or algorithm, which can be used to calculate one or more physical properties. It should be emphasised that, except in the very limited number of cases defined in (Ref), CAPE-OPEN will not define standard methods. In referring to calculation methods, this document includes both any standard methods to be defined and all proprietary methods that may be included in commercially available, or other, packages.

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

Physical Properties Data. Physical Properties Data includes both Physical Properties Parameters and Raw Physical Properties Data. When used without further description, the term will generally refer just to parameters.

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

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

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

Physical Properties System. A software system that includes a physical properties executive, a set of physical properties routines and access to data for a number of chemical components. It will often access a large properties data bank. The system is likely to include text information which the user can access to help select the most appropriate properties, methods and data for the particular application. Properties Systems implement two kinds of interfaces, external interfaces to the Simulator Executive and unit clients, and internal interfaces to calculation routines and neutral files. Since only the Properties System understands the interrelations between its constituent properties routines, a UNIT cannot communicate to a calculation routine directly, but only through a properties package set up within a specific Properties System.

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

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

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

3.4. Use Case Actors.

The major Use Case Actors are described in the following text.

- ? **Material Object** The Material Object holds information defining a material completely at a particular point in a process. It accesses the necessary methods to obtain any properties not held in the object itself (or its associated template).
- ? **Neutral File System.** The Neutral File System is a software component that controls access to and from a neutral file, or other persistent storage mechanism.
- ? **Physical Properties Client.** The client may be one of Stream, Unit Model, or Simulator Executive. Any of these actors may require access to physical properties.
- ? **Physical Properties Developer.** The human being who is notionally a physical properties expert and will set up the physical property options for use by the normal simulator end user. In principle, the Physical Properties developer is responsible for physical properties quality assurance in the organization using simulation.
- ? **Physical Properties System.** The software system controlling access to physical property methods and the associated physical properties databank. It is associated with a Physical Properties Executive, which is the software that enables the Physical Properties System to operate as a self-standing program, independent of any particular simulator.
- ? **Simulation Engineer.** The Simulation Engineer is one of the end-users of a process simulation package. He is distinguished from the "Simulation End User" in that it is assumed that he will be a frequent user with more intimate knowledge of simulation than the routine user. He may set up a whole simulation so that the end-user only has to change data on repeat runs.
- ? **Simulator End User.** A less frequent user than the simulation engineer. Assumed to be a domain expert (for example intimate knowledge of particular chemical processes) but with no programming or special simulation software expertise.
- ? **Simulator Executive.** The software that controls the simulation (for example, determines computational sequence, and calls physical property computations at the appropriate times).
- ? **Stream.** A coding of the material flowing between process units (that is, connecting the Unit Models). Streams may wish to compute properties, for example, when required to be presented as a tabular output with flowrate, temperature, pressure and any other stream properties requested by the user.
- ? **Unit Model.** A Unit Model is a piece of software (for example, a software component, function or subroutine), that models the process behaviour of a particular unit operation, or a class of unit operations. In these physical properties Use Cases, we have not differentiated between individual unit models and classes of unit models; this differentiation has, of course, been made in the Use Cases referring to Unit Models directly. To allow for greater generality of the Use Cases, in this document we also use the term Unit Model to refer to

software models of sub-models (for example a distillation tray) of which a unit model may be built.

3.5. The Use Cases

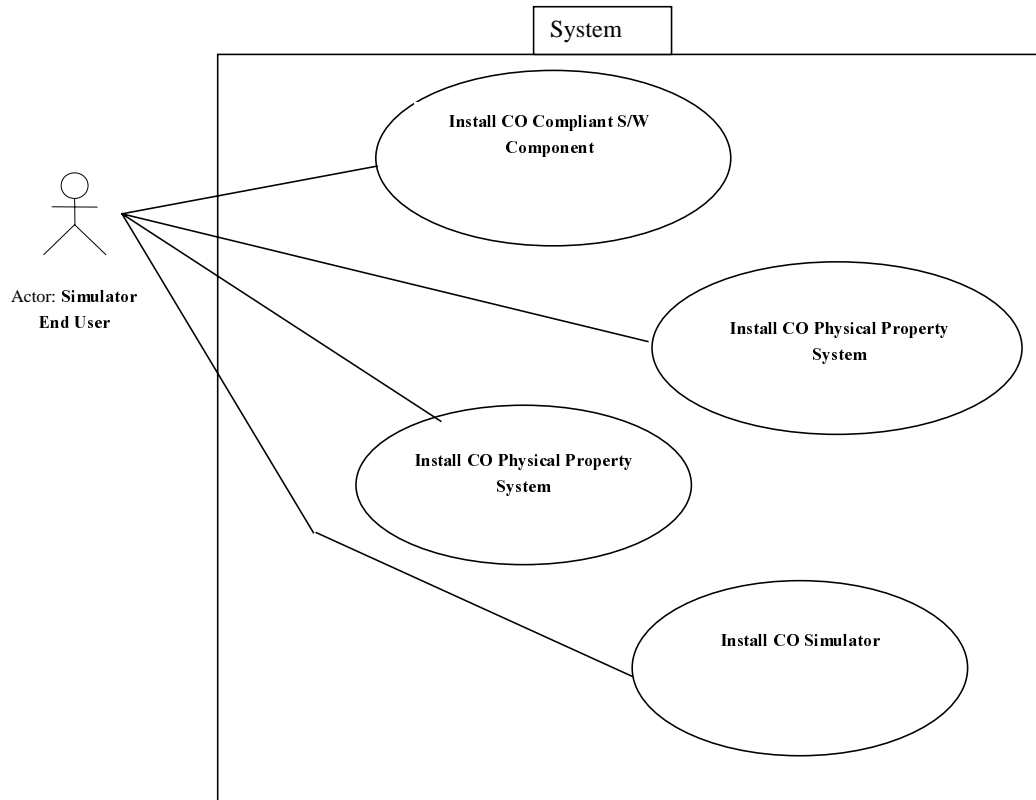
There will be several places in the “Use Cases” where we define separate Actors which may actually be the same person. For example, we separate the Simulation End User (SEU) from the Physical Properties Developer (PPD), to indicate the level of specific expertise that might be required of the relevant actor and to indicate how routinely the Use Case may be performed.

The Actors are described in 3.4

Use Case Actors.

The Use Cases are divided up into groups of related cases, each starting on a new page.

3.5.1. Component Installation Package



3.5.1.1 USE CASE: Install CAPE-OPEN Compliant Software Component

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Simulator End User** requests that the **Set Up Manager** install its CAPE-OPEN compliant software components onto his computer. The **Set Up Manager** asks the **Simulator End User** to indicate the location on the computer where it should install the software component(s), and then copies all the simulator executable and auxiliary files onto the computer at the indicated location.

It then registers all the software components that it contains with the CAPE-OPEN register.

Assumptions :

Exceptions :

Uses :

Extends :

Note: This use case is global to all packages, it is NOT specific to the Thermo work Package. The Use Case as specified above is not complete, but provides a starting point for further refinement. **Set Up Manager** is part of the CAPE-OPEN Compliant Software Component. Neither the component nor the Set Up Manager need to be standardised to be CO-compliant. It is only necessary that the relevant interfaces are standardised.

3.5.1.2 USE CASE: Install CAPE-OPEN Simulator

PRINCIPAL ACTOR: Simulation Engineer. The description “Simulation Engineer” is used here to indicate that the relevant human actor may require more software system expertise than the average end user.

Description:

As in **Install CAPE-OPEN Compliant Software Component** except:

The **Simulator Set Up Manager** acts as the **Set Up Manager**.

Assumptions :

Exceptions :

Uses :

Extends : Install CAPE-OPEN Compliant Software Component

Note : This use case is does not fall within any of the CAPE-OPEN work packages. It is of concern to CAPE-OPEN that simulators should be installable in a way that provides access to CAPE-OPEN compliant components, but should have little impact on the main CAPE-OPEN activities. The Use Case as specified above does not attempt to be complete, because separate solutions are expected to be provided by each vendor. Exactly similar use cases apply to the installation of other simulation components, such as physical properties systems (see below). The Use Case should not be taken to imply that every simulation system has separate Set Up Managers for every major component.

3.5.1.3 USE CASE: Install CO Physical Properties System

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

As in **Install CAPE-OPEN Compliant Software Component** except:
The **Physical Properties System Set Up Manager** acts as the **Set Up Manager**.

Assumptions :

Exceptions :

Uses :

Extends : Install CAPE-OPEN Compliant Software Component

Note : The installation is only relevant to CAPE-OPEN insofar that the installation process should provide the middleware interfaces to be defined elsewhere. The Use Case as specified above is not complete, but provides a starting point for further refinement

3.5.1.4 USE CASE: Install CO Physical Properties Calculation Routine

PRINCIPAL ACTOR: Physical Properties Developer

Description:

As in **Install CAPE-OPEN Compliant Software Component** except:

- i) The **Physical Properties Calculation Routine Set Up Manager** acts as the **Set Up Manager**.
- ii) The component installed is a Physical Properties Calculation Routine.

Assumptions :

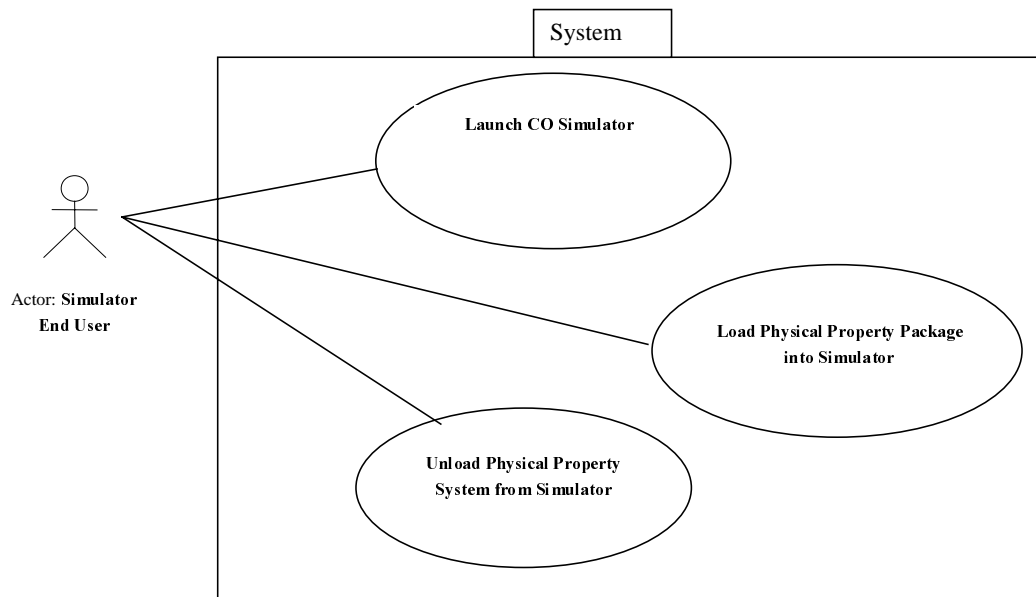
Exceptions :

Uses :

Extends : Install CAPE-OPEN Compliant Software Component

Note : This Use Case is specific to the Thermo work package and describes a facility not currently available in commercial simulators. The Use Case as specified above is not complete, but provides a starting point for further refinement.

3.5.2. Simulator Package



3.5.2.1 USE CASE: Launch CAPE-OPEN Simulator

PRINCIPAL ACTOR: Simulator End User

Description:

The **Simulator End User** requests a **CAPE-OPEN compliant Simulator Executive** to launch itself. The **Simulator Executive** acknowledges the request and loads itself into memory.

It then enquires from the CAPE-OPEN registry whether there are any CAPE-OPEN compliant software components on the computer. If there are some, it provides the **Simulator End User** with an appropriate means of accessing the services provided by these software components (e.g. placing the component's icon on its toolbar, or updating a menu appropriately).

It also loads any default **Physical Properties Systems**.

Assumptions :

Exceptions :

Uses :

Extends :

Note: This use case only slightly extends the behaviour expected of any proprietary simulator. The only novelty relevant to the Thermo work package is that the simulator must have the ability to recognise and link with non-native physical properties systems. The user interface for making the selection from amongst those available is not expected to be CAPE-OPEN standard. The interface to the properties systems must, however, be standardised to a sufficient degree for potential physical properties packages to be recognised by the simulator launched.

There will be a choice as to whether the link to a physical properties system is by creating a copy of the system linked to the simulator, or whether it is by access to a separately maintained properties system. The benefit of the former is that the simulation can be exactly reproduced at a future time. The benefit of the latter is that only one quality-assured copy of the physical properties system is maintained.

3.5.2.2 USE CASE: Load Additional Physical Properties System

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests that an additional **Physical Properties System** be loaded into the simulator.

The **Simulator Executive** acknowledges the request and gets the list of CAPE-OPEN compliant **Physical Properties Systems** from the **CAPE-OPEN registry**. It communicates this list to the **Physical Properties Developer** and asks him to select one from the list. He selects one or aborts.

If the **Physical Properties Developer** selects one of the available **Physical Properties Systems**, when the **Simulator Executive** generates a link to it, and the components, methods, and routines known to the **Physical Properties System** become “known” to the **Simulator Executive**.

If the user aborts the operation, no action is taken, and the components, methods, and routines known to the **Physical Properties System** remain “unknown” to the **Simulator Executive**.

Assumptions :

A **Physical Properties System** can generate one or more **Physical Properties Packages**, which will be linked to a simulator when the system is linked. It is expected that a Properties Package can be generated before or after the system is linked to the simulator.

Exceptions :

Simulator End User aborts the operation.

Uses :

Extends :

Notes: (1) As for the <<Launch CAPE-OPEN Simulator>> Use-Case, there will be a choice between a direct and an indirect link to the loaded Physical Properties System. This choice will be common to a number of other CAPE-OPEN compliant components, so is not unique to the Thermo Work Package. (2) An alternative, and possibly better way of linking in Physical Properties Packages, would be to perform “Retrieve Physical Properties Package”. Reference to the package would automatically load the Physical Properties System Executive that created the package (in the same way that accessing, an Excel file loads excel.exe). The appropriate Properties System would still, of course, need to link to the simulator.

3.5.2.3 USE CASE: Unload Physical Properties System from Simulator

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests a **Physical Properties System** be unloaded from the simulator.

The **Simulator Executive** acknowledges the request and displays the list of loaded **CAPE-OPEN compliant Physical Properties Systems**. It then asks the **Physical Properties Developer** to select one of them from the list. The **Physical Properties Developer** selects one from the list, or aborts the operation.

If the **Physical Properties Developer** selects one of the **Physical Properties Systems**, the **Simulator Executive** unloads it. All the components, methods, and routines provided within the **Physical Properties Packages** generated within the System then become “unknown” to the **Simulator Executive**.

An exception is thrown if a current Materials Template refers to a package that is thereby removed.

If the Developer aborts the operation, no action is taken.

Assumptions :

Each Properties Package has a record of a link to every Materials Template that uses it so that an appropriate exception can be thrown if a request is made to delete the package.

Exceptions :

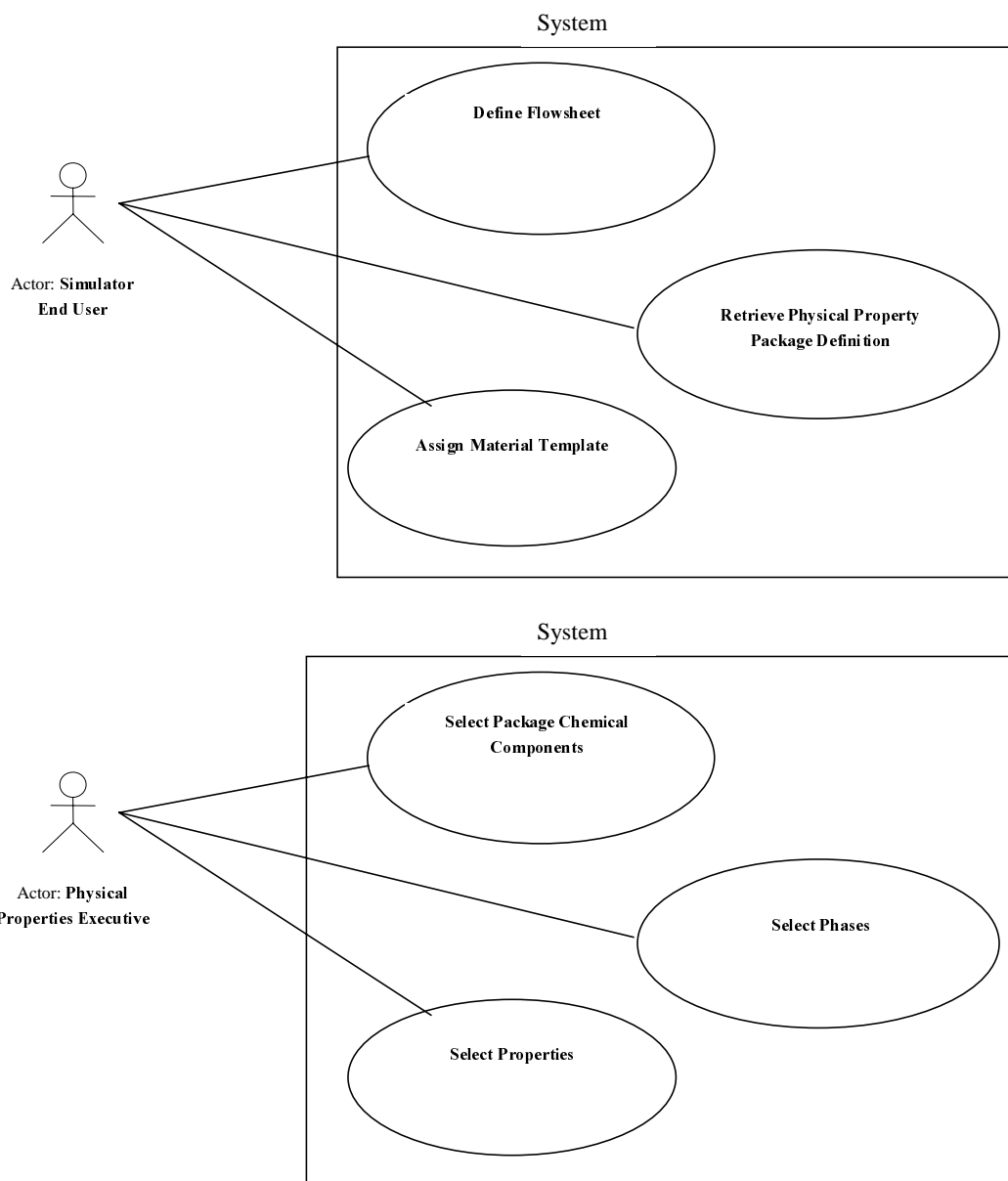
- (i) **Simulator End User** aborts the operation.
- (ii) A current Materials Template is using a package that is about to be deleted. (The **Physical Properties Developer** is then requested as to whether they wish first to delete the Template or whether they wish to abort).

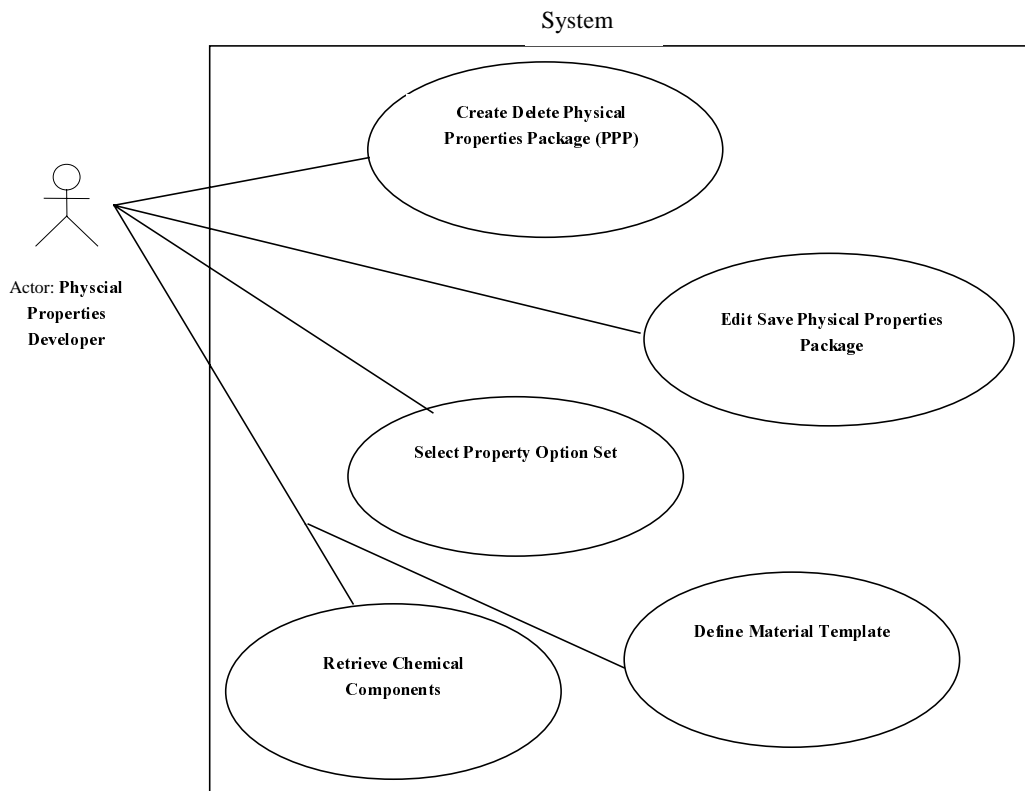
Uses :

Extends :

Note : “Assumptions” above indicates relevance to Thermo work package.

3.5.3. Flowsheet Setup Package





3.5.3.1 USE CASE: Define Flowsheet

PRINCIPAL ACTOR: Simulator End User

Description:

In an order dictated by the **Simulator Executive**, the **Simulator End User** does the following:

- (i) sets up one or more Material Templates using the USE CASE: Define Material Template use case
- (ii) selects and specifies the unit models to be included in the flowsheet
- (iii) links the units up by defining the process topology, including feeds and products.
- (iv) associates each stream with a previously defined Material Template using the <<Assign Material Template to Flowsheet, Streams, or Units>> Use Case.
- (v) specifies any other information required for the simulation.

Assumptions :

The **Simulator Executive** ensures that a given piece of information is provided by the **Simulator End User** before it allows him to initiate another process that requires it.

Components that are the same in different Material Templates are given the same name by the Simulator End User (or Physical Properties Developer) when the templates are created from the packages. It is thus the users' responsibility to ensure that the system recognises when the same components appear in different streams.

Exceptions :

Uses :

Select Chemical Component.

Extends :

Note : The Use-Case is not specific to the Thermo work package, but is relevant in that Material Templates must be definable. The definition of the form of these actions and the sequence in which they must be performed is outside the scope of CAPE-OPEN.

3.5.3.2 USE CASE: Define Material Template

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests to define a new **Material Template**. The **Simulator Executive** acknowledges the request and prompts the **Physical Properties Developer** to select a **Physical Properties Package**. The **Developer** uses the **PRINCIPAL ACTOR: Physical Properties Developer** (or Simulator End User). use case to select the required components.

The **Simulator Executive** requests the package to provide a list of possible phases and displays a list of all the phases possible with the package selected. The **Executive** prompts the **Physical Properties Developer** to select the phases required for the template. The **Physical Properties Developer** may select from one to all phases. If all phases are selected, the template will allow equilibrium between all the phases possible. If only one phase is selected (for example, vapour) all properties are computed as if that were the only phase that could form; metastable conditions may thus be computed. Any subset of less than all possible phases may also give rise to metastable conditions.

The **Simulator Executive** requests the package whether it provides Option Sets (for example, “high pressure conditions” or “low pressure conditions”). The **Physical Properties Package** returns a list of Option Sets. If there is more than one, the **Simulator Executive** prompts the **Physical Properties Developer** to select one.

Finally, when the **Physical Properties Developer** signals that he has completed the definition of the Material Template, the **Simulator Executive** stores the template.

Assumptions :

The <<**Select Chemical Component**>> Use Case enables components to be renamed for reference in the simulation. In this way components common to several templates can be recognised as the same because they have the same names.

Exceptions :

Uses : Select Chemical Component.

Extends :

Notes: (1) The Template contains all the information to compute physical properties for any defined material, plus identification information such as list of components present. (2) Some simulators may provide a default for including all the package components in a template. (3) Several templates may be developed from one package.

3.5.3.3 USE CASE: Select Chemical Components (for Template)

PRINCIPAL ACTOR: Physical Properties Developer (or Simulator End User).

Description:

The **Physical Properties Developer** indicates that he wishes to select a component from a specified **Physical Properties Package**.

If there is no specified **Physical Properties Package**, the **Simulator Executive** informs the **Physical Properties Developer** of this, and takes no further action.

Otherwise, the **Simulator Executive** displays the list of chemical components (for example, identifier, formula, chemical name, and CAS number) in the package and asks him to select one, or abort the operation.

If the **Physical Properties Developer** aborts the operation, the **Simulator Executive** takes no further action.

If the **Physical Properties Developer** selects one from the list, he is prompted to give it another name by which it will be known in the template (and the simulation). If the **Physical Properties Developer** elects not to give it a new name, the package name is employed as default. The component and its template name is then loaded into the Material Template. (In this context, it is expected that “loaded” will actually imply that a pointer or similar index into the package will be loaded).

Assumptions :

All interaction coefficients are assigned in setting up the physical properties package. The Developer therefore needs to take no specific steps to handle interaction coefficients.

Exceptions :

No specified Physical Properties Package.

Uses :

Extends :

Note : One Material Template can only employ one Physical Properties Package. The components can thus only be selected from that package. The actions specified in this Use Case are all expected to be simulator specific, so will not be made CAPE-OPEN standard. The requirement that these actions are available to the user does, however, imply that the standard interfaces transmit the information needed to achieve these features (for example transmits component, property and option names).

3.5.3.4 USE CASE: Assign Material Template to Flowsheet, Streams or Units

PRINCIPAL ACTOR: Simulator End User

Description:

The **Simulator End User** requests to associate a previously defined **Material Template** with a whole flowsheet, a set of “streams” and/or units. The **Simulator Executive** acknowledges the request and makes the requested association. Any previous association of the streams and/or units is superseded.

Assumptions :

The **Material Template** defines the list of components (pseudo-components etc) within the stream. For dispersed phases it also defines methods used to record the characteristics of the dispersed phase(s); for example size ranges to be used.

Exceptions :

Uses :

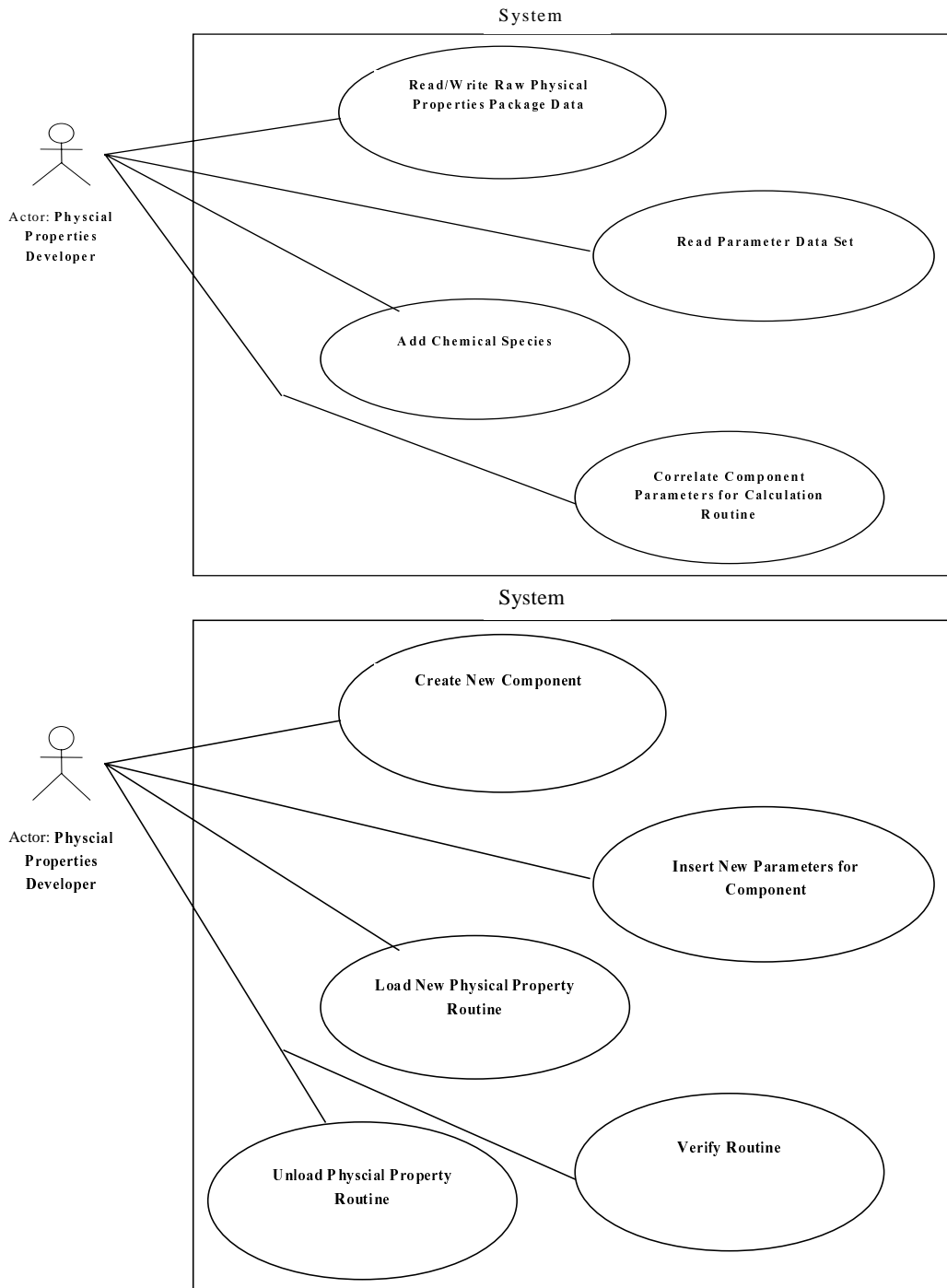
Extends :

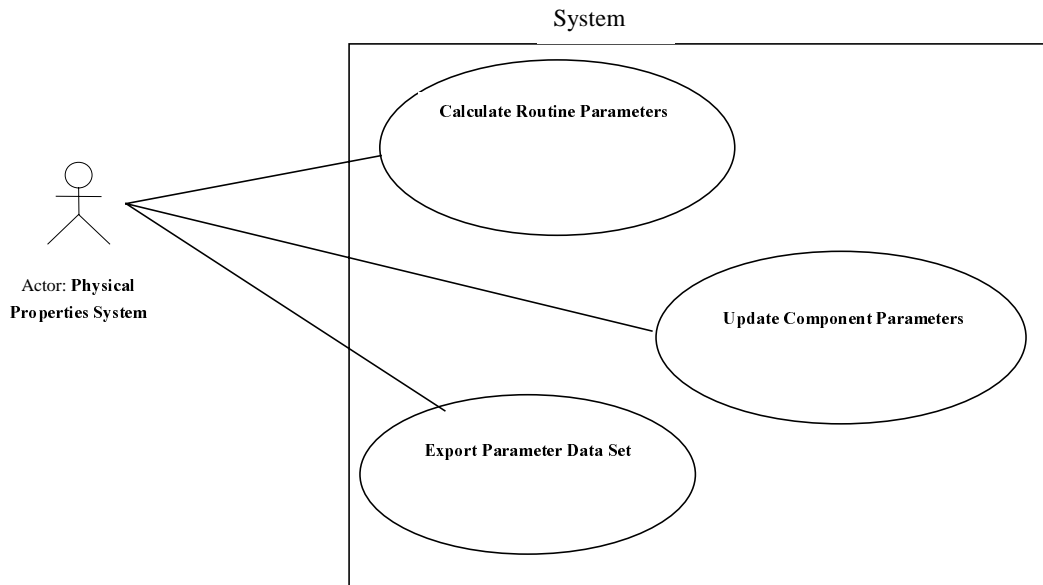
Note : Association with streams ensures that whenever properties are requested for any stream, the calculation methods associated with the relevant material are used. Similarly association with units ensures that when the unit requests the properties of any material it processes, the calculation methods within the associated Physical Properties Package can be used.

This is a facility already provided by every simulator but different simulators perform the association in different ways (e.g. some on a stream basis [with a maximum of 1 package per stream] and others on a unit basis). The nominal association with streams is the most flexible because it includes, as a subset, the other possible associations.

The Use Cases for UNITS have the corresponding use case viewed from the unit perspective. It will be important to ensure that the use cases are compatible.

3.5.4. Physical Properties System





3.5.4.1 USE CASE: Correlate Component Parameters

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer User** requests to fit parameters for a chemical component (or set of chemical components) for a physical properties calculation routine (or routines). The **Physical Properties System** acknowledges the request, provides the **Physical Properties Developer** with a list of routines, and asks him to select one.

The **Physical Properties Developer** selects a **Physical Properties Routine(s)** from the list. Each Routine returns the set of parameters that needs to be fitted. It also optionally provides default initial estimates and gives the Physical Properties Developer the option of entering alternative initial estimates. The “USE CASE: Access Raw Physical Properties Data” Use Case is employed to provide the data to be fitted.

If sufficient data is available in the data set, the **Physical Properties System** performs a regression to obtain the parameter values that best fit the data set. The **Physical Properties System** invites the **Physical Properties Developer** either to use the “USE CASE: Update Component Parameters” Use Case to enter or update the fitted component parameters in an existing data set, or to use <<export parameter data set>> to create a new data set.

If sufficient information is not available, the **Physical Properties System** informs the **Physical Properties Developer** of the deficiency and terminates. [Some **Systems** may prompt the **Physical Properties Developer** to supply missing information or provide values for some parameters so that less need be fitted].

Assumptions : The fitting package can perform the statistical degrees of freedom analysis to advise on data adequacy.

Exceptions : (1) Parameter data already exists - abort. (2) Insufficient data.

Uses : Fehler! Textmarke nicht definiert., Fehler! Textmarke nicht definiert., Fehler! Textmarke nicht definiert.

Extends :

Note : The User can open the Physical Properties System at any time, either when the system is stand alone, or when it is embedded in the Simulator. This action, or any other capability of the Physical Properties System, can thus be initiated whilst using the simulator. The facility will be entirely Physical Properties System dependent, but will require CAPE-OPEN standards to work successfully. For example, interfaces need to be able to transfer information on required parameters and default initial estimates between CAPE-OPEN standard routines and the properties systems.

3.5.4.2 USE CASE: Access Raw Physical Properties Data

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests to import a data set containing physical property data for a component (or set of components). The **Physical Properties System** acknowledges the request and presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**. The Developer selects one of the systems. The **Physical Properties System** then employs the <<**Import Data**>> Use Case to access to required data.

Assumptions :

The fitting package can perform the statistical degrees of freedom analysis to advise on data adequacy.

Exceptions :

Uses :

Extends :

Notes: Comments as for previous use-case.

3.5.4.3 USE CASE: Export Raw Physical Properties Data

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** requests to create a physical properties data set for a component (or set of components).

The **Physical Properties System** acknowledges the request and asks the **Physical Properties Developer** to supply the list of components, properties and other information required to generate the data (for example, the compositions, temperatures and pressures of the points for which data values are needed).

The **Physical Properties System** requests the **Physical Properties Developer** to supply the name under which the data set is to be saved. The **Physical Properties System** presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**. The **Physical Properties Developer** selects one of the **Neutral File Systems**.

The **Physical Properties System** performs the necessary calculations and employs the <<Export Data>> Use Case to store the required data.

Assumptions :

Exceptions :

Uses :

Extends :

Note : Comments as before.

3.5.4.4 USE CASE: Update Component Parameters

PRINCIPAL ACTOR: Physical Properties System

Description:

The **Physical Properties System** asks the **Physical Properties Developer** whether he wishes to save the calculated parameter data for all the components in a separate file or update the current data set. If the Developer requests a new file, the **Physical Properties System** uses the **PRINCIPAL ACTOR: Physical Properties System » Use Case**. Otherwise it employs the <<**Update Parameter Data Set**>> Use Case to store the new values.

Assumptions :

Exceptions :

Uses : USE CASE: Create a New Component

Extends :

Notes:

3.5.4.5 USE CASE: Export Parameter Data Set

PRINCIPAL ACTOR: Physical Properties System

Description:

The **Physical Properties System** presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**. The **Physical Properties Developer** selects one of the **Neutral File Systems**.

The **Physical Properties System** requests the **Physical Properties Developer** to supply a name under which the data is to be saved.

The parameter data of all the components are saved using the <<**Export Data**>> Use Case. Where the components or methods are Physical Properties System specific, the key words etc employed may only be recognized by the System generating the file.

Assumptions :

Exceptions :

Name invalid, User aborts

Uses :

Extends :

Notes:

- (i) Individual methods may come from a variety of sources each with its own format. There will, however, still need to be some standard header within the data set so that the appropriate (possibly proprietary) method is associated with the appropriate data.
- (ii) The ability to extend the neutral file format indefinitely by adding new key words (which may be System specific) creates an important flexibility. If, at any later date, it is decided to extend the initial small sub-set of CAPE-OPEN standard methods/routines, it is only necessary to extend the list of standard key words (or other identifiers) to extend the CAPE-OPEN standard.

3.5.4.6 USE CASE: Import Parameter Data Set

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** requests to import in a set of parameter data for one or more components. The **Physical Properties System** acknowledges the request and presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**.

The **Physical Properties Developer** selects one of the systems. The **Physical Properties Developer** queries the selected **Physical Properties Routines** for the parameters they require. The **Physical Properties Routines** (which may be internal to or external to the **Physical Properties System**) return a list of identifiers for the property parameters needed. The **Physical Properties System** then employs the <<Import Data>> Use Case to access the required parameter data.

Assumptions :

Exceptions :

Uses :

Extends :

Note: This Use Case could be part of a common Use Case with the <<Import Raw Physical Properties Data>> use case if we decide that one **Neutral File System** can handle both parameters and “raw data”. It is a matter of deciding which is more convenient to use. Other comments are as for the previous similar Use Cases.

3.5.4.7 USE CASE: Add New Chemical Species to System

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** uses the USE CASE: Create a New Component » use case followed by either the «USE CASE: Insert New Parameters for Component » use case and/or the «USE CASE: Correlate Component Parameters. » use case for each routine to be correlated.

Assumptions :

Exceptions :

Name already exists, User aborts

Uses : Fehler! Textmarke nicht definiert., USE CASE: Insert New Parameters for Component, USE CASE: Correlate Component Parameters

Extends :

Note : This use case could take place within a simulator (by editing the Properties System linked from the Simulator Executive) or “off-line” using the [G]UI of the Physical Properties System directly. Other comments are as before. The use case falls almost entirely within the proprietary software orbit.

3.5.4.8 USE CASE: Create a New Component

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** selects a **Physical Properties System** and indicates that he wishes to add a new component into it.

The **Physical Properties System** acknowledges the request and asks the **Physical Properties Developer** to supply the name for the new component. The **Physical Properties Developer** may at this stage request a list of the names of existing chemical species, to help him select a name that is not already in use. After the **Physical Properties Developer** has supplied the name, the **Physical Properties System** checks whether the name is already in use by another component within the system.

If the name is not in use, it creates a component with the name specified by the **Physical Properties Developer**.

If the name is in use, it informs the **Physical Properties Developer** of this and prompts him to supply an alternative name or abort the process.

Assumptions :

Exceptions :

Name already exists

Uses :

Extends :

Notes: Probably all within the Properties System orbit and outside CAPE-OPEN, although the interfaces have to provide means for passing component names and data for components that did not previously exist.

3.5.4.9 USE CASE: Insert New Parameters for Component

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** requests to insert parameter data for a selected component. The **Physical Properties System** acknowledges the request and checks to see if the selected component's parameters are "read-only".

If the component's parameters are "read-only", the Physical Properties System informs the **Physical Properties Developer** of this and takes no further action.

If the component's parameters are not "read-only" (i.e. they can be replaced), the **Physical Properties System** asks the **Physical Properties Developer** whether he wishes to read in the new parameter data, or whether he wishes to insert the parameter data manually.

If the **Physical Properties Developer** opts to insert the data manually, he uses the <<Edit Parameter Data>> Use Case.

If the **Physical Properties Developer** opts to read in the data, he uses the « USE CASE: Import Parameter Data Set» use case. The **Physical Properties System** then searches within the imported data to find the parameters relating to the selected component. If it finds the relevant parameters, it overwrites the old parameter data of the selected component with the new data. If it does not find any parameters for the selected component within the imported data, it informs the **Physical Properties Developer**. The **Developer** then has two options, either to abort, or to select the name of a component that does exist within the data set whose parameters are to be used.

Assumptions: It is assumed that all data previously created by the Physical Properties Developer can be overwritten.

Exceptions: Data for component not found; Developer aborts.

Uses : Edit Parameter Data, Import Parameter Data Set.

Note: Parameters include "name", constant properties and parameters in correlating equations. The principle impact on CAPE-OPEN appears to be the requirement for a standard interface to a **Neutral File System**.

3.5.4.10 USE CASE: Edit Parameter Data

Description:

This use case is outside the scope of CAPE-OPEN, as each simulator vendor will provide (or not provide) its own facilities for editing of component parameter data at its discretion

3.5.4.11 USE CASE: Load a New Physical Properties Routine

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests the **Physical Properties System** to load a new **Physical Properties Routine**. The **Physical Properties System** gets the list of CAPE-OPEN compliant Physical Properties calculation routines from the CAPE-OPEN registry and displays it. The **Physical Properties Developer** selects one of the Physical Properties Calculation Routines from the list, or aborts the operation.

If the **Physical Properties Developer** selects one of the displayed routines, the **Physical Properties System** loads the routine and uses the « USE CASE: Verify Routine» use case. If the verification fails, the **Physical Properties System** informs the **Physical Properties Developer** of the failure and unloads the routine.

If the verification is successful, and any of the physical properties was unknown to the **Physical Properties System**, it adds the physical property to its list of known physical properties, and informs the **Physical Properties Developer** that it has done so. If the routine is not previously known, it adds the routine to its list of known routines, and informs the **Physical Properties Developer**.

If the verification is successful, and both the method and property are already known to the **Physical Properties System**, it informs the **Simulator End User** of this, and provides the routine as an alternative for performing calculations using the method.

Assumptions: The calculation routine has previously been installed using the <<Install CAPE-OPEN Physical Properties Calculation Routine>> Use Case.

Exceptions:

Uses: USE CASE: Verify Routine

Extends:

Notes : (1) This use case provides the mechanism for adding a new physical property to the system. (2) It is expected that each routine will be accompanied by a clear description of the methods it implements. (3) Each routine may optionally specify the type/format of chemical component parameters it requires (also initial values etc so that they can be fitted). It is recognized that some routines may have their data “hard wired” in, or may have their own GUI and data storage. Such private parameters might be expected for very specialist routines developed for specific mixtures. (4) This Use Case applies to adding CAPE-OPEN compliant routines (vendors may offer similar methods for proprietary routines).

3.5.4.12 USE CASE: Unload Physical Properties Routine

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests that a Physical Properties System unload a Physical Properties Routine from a Physical Properties package.

The **Physical Properties System** acknowledges the request and shows the **Physical Properties Developer** the list of currently loaded **CAPE-OPEN** compliant routines and asks him to select one from the list. The **Physical Properties Developer** then selects one of the Physical Properties calculation routines from the list, or aborts the operation.

If the **Physical Properties Developer** selects a routine, the **Physical Properties System** unloads the selected routine. The **Physical Properties System** then removes the “name” of the routine from the list of known routines.

If the routine is associated with a current **Physical Properties Package**, the Developer is informed and requested whether to delete the package, modify the package or abort.

If the **Physical Properties Developer** aborts the process, the operation is abandoned and no calculation routine is unloaded.

Assumptions: This Use Case has meaning only if the calculation routine has previously been installed using the <<**Install CAPE-OPEN Physical Properties Routine**>> Use Case.

Exceptions: (1) The **Physical Properties Developer** aborts the process. (2) The **Properties Routine** is in use in a still-active Material Template.

Uses :

Extends :

Notes: Some routines will deliver values for several properties. As the routines are added (previous Use Case) and deleted, the Properties Executive will have to handle adding properties to (and removing them from) the list of available properties. The CAPE-OPEN standard interface must provide facilities for transmitting information on which **Properties Routines** are linked to which **Properties Systems** and which **Material Templates** so that exceptions can be flagged at this stage, rather than causing run-time failures.

3.5.4.13 USE CASE: Verify Routine

PRINCIPAL ACTOR: Physical Properties System

Description:

The **Physical Properties System** interrogates the routine to identify its “name” and associated physical properties, associated method (if any), other parameters or properties required for calculation. The System provides a check facility (possibly including a GUI) to ensure that the properties provided by the loaded routine correspond to those required.

If any errors occur, such as

- Routine “name” not found
- Associated physical property does not match property expected

any other error (not yet identified)

then the verification is deemed to have failed. Otherwise the verification is successful.

Assumptions :

Exceptions : See above, plus others to be identified.

Uses :

Extends:

Notes:

The extent of verification will be System dependent and not part of the CAPE-OPEN standard. It is only necessary that the CAPE-OPEN standard provides interfaces that allow such a routine to be used because there is always potential for incompatibility when a new routine is added to a System and there must be provision for ensuring adequate compatibility.

It must be possible for a new routine to introduce new parameters. The Properties System must provide a mechanism for entry, retrieval, or fitting of these parameters without knowing what they mean. This requirement should be achievable simply by adding the new parameters to the list of known parameters. Some descriptive text information should be part of the new parameter definition.

3.5.4.14 USE CASE: Create a Physical Properties Package

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** selects a **Physical Properties System** from amongst the registered properties systems and then informs the **Physical Properties System** that he wishes to define a **Physical Properties Package**. The **Physical Properties System** acknowledges the request.

The **Physical Properties System** then invokes the following Use Cases, in turn:

<< Create option set>>

<< Select package chemical components>>

<< Select phases>>

<< Select properties>>

<< Select routine >>

<< Select data set >>

When all these actions are complete, the properties package has been created

Assumptions :

Exceptions :

Uses : Create option set, Select package chemical components, Select phases, Select Properties, Select routine and Select data set.

Extends :

Note : This Use Case is expected to be internal to the Physical Properties System. It is only relevant to CAPE-OPEN in that the interface between a simulator and a Physical Properties System must transfer package information (rather than information on everything that is in the System). The content of the Package is thus relevant to CAPE-OPEN.

3.5.4.15 USE CASE: Delete a Physical Properties Package

PRINCIPAL ACTOR: Physical Properties Developer

Description:

The **Physical Properties Developer** informs the **Physical Properties System** that he wishes to delete a previously defined Physical Properties Package. The **Physical Properties System** acknowledges the request and provides the **Physical Properties Developer** with a list of defined Physical Properties Packages, from which to select the package to be deleted. The **Physical Properties Developer** may then select a package or abort the operation.

If the **Physical Properties Developer** selects a package, the **Physical Properties System** removes all knowledge of the package. If however the **Physical Properties Developer** aborts the operation, the **Physical Properties System** takes no further action.

The **Physical Properties System**, before the deletion, checks that there are no references to the package from Material Templates in a loaded simulation. If there are, an Exception is thrown. (Throwing an exception at this stage avoids possible run-time errors if a template should refer to a non-existent package).

Assumptions :

It is assumed that properties packages are built up and controlled by Physical Properties Systems, not by the simulators themselves. There is thus a clear demarcation between simulator and properties system.

Exceptions:

- (i) User aborts operation.
- (ii) Package required by simulator (user may also abort operation).

Uses :

Extends :

Notes: Only relevant to CAPE-OPEN in that some record must exist of the link between a Properties Package and a Materials Template. That record may require transmission across the Simulator/Properties System interface. It will be recalled that this interface itself may be achieved by a direct or indirect link.

3.5.4.16 USE CASE: Select Package Chemical Components

PRINCIPAL ACTOR: Physical Properties System.

Description:

The **Physical Properties System** requests the **Physical Properties Developer** to select a list of chemical components.

The **Physical Properties Developer** selects a list of components from the components available within the **Physical Properties System**. We do not define here the details of this operation which is expected to be System dependent. To present a list of all chemical components for which the system has properties may be impractical. There will probably be search facilities to find suggested components.

The **Physical Properties System** requests the **Physical Properties Developer** to add chemical components not present in the properties system. The **Physical Properties Developer** either accepts or rejects the invitation. If he rejects the invitation, the use case is complete. If he accepts the invitation, he enters the names of new chemical components until the list is complete, when the use case is also complete.

Assumptions :

Exceptions :

Uses :

Extends :

Note : is expected to be System dependent, and almost entirely outside the scope of CAPE-OPEN. The feature that is relevant to CAPE-OPEN is that current physical properties systems probably do not work in this way to create Physical Properties Packages.

3.5.4.17 USE CASE: Create a Properties Option Set

PRINCIPAL ACTOR: Physical Properties System.

Description:

The **Physical Properties System** requests the **Physical Properties Developer** whether he requires that the **Physical Properties Package** being created has more than one **Physical Properties Option Set**.

If the reply is “yes”, the **Physical Properties System** requests the developer to name each option set (for example, “low pressure properties”, “high pressure properties”).

The **Physical Properties System** creates the requested number of option sets, labelling each. If the list of package chemical components has already been selected, each option set is assigned the same list of components. (If chemical components are selected after the option sets have been assigned, components are allocated to all option sets as they are selected).

When all these actions are complete, the use case is complete.

Assumptions :

Exceptions :

Uses :

Extends :

Notes:

- i. This Use Case will be entirely vendor dependent; some vendors will not offer the facility for “option sets”.
- ii. Physical Properties Systems that do not offer option sets may still be fully CAPE-OPEN compliant because the options can be delivered by providing alternative packages to cover the alternative sets of conditions.
- iii. The **Simulator** may wish to access a particular option set (because it wants to access a Materials Template that is constructed from an option set). The CAPE-OPEN standard interface must, therefore, provide provision for passing on option-set identifier.

3.5.4.18 USE CASE: Select Phases

PRINCIPAL ACTOR: Physical Properties System.

Description:

The **Physical Properties System** requests the **Physical Properties Developer** to select a list of phases to assign to the **Physical Properties Package** just being created.

The **Physical Properties Developer** selects a list of phases from the phases available within the **Physical Properties System**. The **System** will probably present a list of all phases available within it (vapour, aqueous liquid, ... , particulate etc) from which the user selects those that will be required in the **Package**.

The **Physical Properties System** requests the **Physical Properties Developer** to add phases not present in the properties system. The **Physical Properties Developer** either accepts or rejects the invitation. If he rejects the invitation, the Use Case is complete. If he accepts the invitation, he enters the names of new phases (e.g. liquid crystal) until the list is complete, when the Use Case is also complete.

The selected phases are assigned to all Properties Option Sets available within the package.

Assumptions :

The Properties Option Sets are created before the phases are assigned. The obvious modifications are applied if not.

Exceptions :

Uses :

Extends :

Note : When **Material Templates** are subsequently being created from the packages, there may be a further selection of a subset of the phases selected here.

3.5.4.19 USE CASE: Select Properties

PRINCIPAL ACTOR: Physical Properties System.

Description:

The **Physical Properties System** requests the **Physical Properties Developer** to select a list of properties.

The **Physical Properties Developer** selects a list of properties from the properties available within the **Physical Properties System**. The **Physical Properties System** will probably present a list of all properties available (density, partial pressure, thermal conductivity etc) from which the user selects those that will be required in the **Properties Package** just being created.

The **Physical Properties System** requests the **Physical Properties Developer** to add properties not present in the properties system. The **Physical Properties Developer** either accepts or rejects the invitation. If he rejects the invitation, the use case is complete. If he accepts the invitation, he enters the names of new properties (e.g. magnetic susceptibility) until the list is complete, when the use case is also complete.

Selected properties are applied to all Properties Option Sets within the Package just being created.

Assumptions :

The Option Sets are created before the properties are assigned. The obvious modifications are applied if not.

Exceptions :

Uses :

Extends :

Note : It is not expected that Materials Templates will select a subset of the properties selected here because little would be saved by so doing.

3.5.4.20 USE CASE: Select Properties Routines

PRINCIPAL ACTOR: Physical Properties System.

Description:

For each property that the **Physical Properties Developer** Selects, the **Physical Properties System** will present a list of routines capable of computing the property and requests that one is selected. The Physical Properties Developer selects one from the list. (For example, he may select a particular implementation of the Redlich-Kwong-Souave equation from a list of alternative equations of state). Several alternative scenarios now arise, which are briefly outlined as follows:

- (i) the selected routine may also compute other properties, in which case the Developer is requested whether he wishes the further properties also to be computed by the routine. If so he accepts the properties suggested.
- (ii) the selected routine may require further properties in order to compute the requested property (for example, computing Thermal Conductivity from Prandtl Number also requires values for Heat Capacity and Viscosity). The **Physical Properties System** displays the required further properties. The Developer marks any such properties that are to be computed by standard methods already selected; such properties are removed from the list. Where a special routine is to be used, the special routine is selected and the property removed from the list. As further routines are selected, properties are added to and deleted from the list. If any remain at the end of the selection process, an exception is thrown.
- (iii) the selected routine may require, or optionally benefit from, additional (e.g. correction) routines, for example activity coefficient routines. These further routines are selected (in effect a recursive use of the selection process).

Where the package just being created has more than one Properties Option set, each selected routine may be applied to any number of selected Properties Option Sets. Within each option set, the routine may be applied to any number of selected phases.

Assumptions: The Option Sets are created before the properties are assigned.

Exceptions :

(1) There are properties still left with no routines selected. (2) There are no routines for one or more selected properties. (3) One or more Option Sets has no routine for one or more of its properties.

Note : Materials Templates cannot select individual routines. This Use Case is vendor specific (i.e. not CAPE-OPEN standard).

3.5.4.21 USE CASE: Select Data Set

PRINCIPAL ACTOR: Physical Properties System.

Description:

For each **Properties Routine** that the **Physical Properties Developer** Selects, the **Physical Properties System** will present a list of suitable Data Sets. Where there is more than one suitable Data Set, the Physical Properties Developer selects one from the list. The relevant data from that set are then added to the Physical Properties Package just being developed. Thus the values for the pure component parameters and for the interaction parameters are selected for the subset of chemical components in the Package. (The final Package then contains hundreds of parameters, rather than the thousands to millions likely to be available from a Properties Data Bank associated with the Properties System).

Where the package just being created has more than one Properties Option set, each selected data set may be applied to any selected Properties Option Sets. (From one set up to all sets). Thus different data sets may be optionally assigned to each of the Properties Option Sets. Within each Option Set, the data may be applied to any selected phases (from 1 to all phases).

Assumptions :

- i. The Option Sets are created before the data sets are assigned.
- ii. The chemical components are selected before the data sets are assigned.
- iii. The data have already been loaded from any Neutral File using the <Import Parameter Data Set>> Use Case.

Exceptions :

Uses :

Extends :

Note : Materials Templates cannot select individual Data Sets. This Use Case is vendor specific (i.e. not CAPE-OPEN standard).

3.5.4.22 USE CASE: Edit Physical Properties Package

PRINCIPAL ACTOR: Physical Properties Developer

Description:

This use case is developed by analogy with the <<Create a Physical Properties Package>> use case.

Assumptions :

Exceptions :

Uses :

There will be a set of subsidiary use-cases similar to those for the <<Create a Physical Properties Package>> use case.

Extends:

Note :

3.5.4.23 USE CASE: Save Physical Properties Package Definition

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

When the **Physical Properties Developer** has selected all the components, correlations, data etc needed for a **Physical Properties Package** he may indicated to the **Physical Properties System** that he wishes to save it.

The **Physical Properties System** acknowledges the request and presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**. The **Physical Properties Developer** selects one of the **Neutral File** systems. The **Physical Properties System** requests that the **Physical Properties Developer** supply a name under which the Properties Package will be saved. It employs the << Export Data>> Use Case to save the Package Definition.

Assumptions :

The Package Definition includes the names of the routines used (which must be unique), the names of the properties provided by each routine, the names of the parameters and lists of the numerical value of the parameters. There must be a standard CAPE-OPEN interface by which this information can be transmitted to a Neutral File System.

Exceptions :

Uses :

Extends :

Note : There will be a guarantee that the Properties System that stored the package definition can read it. Since many of the routines and data sets will be proprietary; there will be no guarantee that any other Properties System can retrieve the data set.

3.5.4.24 USE CASE: Retrieve Physical Properties Package Definition

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** indicates to the **Physical Properties System** that he wishes to retrieve a **Physical Properties Package**.

The **Physical Properties System** acknowledges the request and presents the **Physical Properties Developer** with a list of linked **Neutral File Systems**. The **Physical Properties Developer** also provides a name for the **Properties Package**. The **Physical Properties System** employs the <<**Import Data**>> Use Case to import the complete description of the **Properties Package** from the linked **Neutral File System**. Where there are Properties Routine names not recognised by the **Physical Properties System**, an exception is thrown.

The response to the exception will vary depending on the proprietary **Properties System** used. Typical responses would be to invite the used. Typical responses would be to invite the **Physical Properties Developer** to load a routine of the correct name, to provide a dialogue similar to that described under <<**Import Data**>>, or to ignore that set of data.

Assumptions :

Exceptions :

- | |
|---|
| (i) Unknown Physical Property |
| (ii) Unknown physical properties routine found in package |
| (iii) Unknown parameter name. |

Uses :

Extends :

Notes: Where the retrieved Properties Package contains unrecognised Properties Routines etc. so that the Package is not fully recovered, the **Properties Developer** would typically select the nearest equivalent within the current Properties System and use the <<**Export Raw Physical Properties Data**>> Use Case from the donor system and the <<**Correlate Component Parameters**>> from the recipient Properties System to refit the parameters.

3.5.4.25 USE CASE: Update Parameter Data Set

PRINCIPAL ACTOR: Physical Properties System

Description: The **Physical Properties System** presents the **Physical Properties Developer** with a list of linked Neutral File Systems. The **Physical Properties Developer** selects one. The <<**Import Data**>> Use Case is employed to import the matching data set.

The **Physical Properties System** checks whether it already has parameters for the component or components. If it has none, it merges the new data set with the old and uses the <<**Export Data**>> Use Case to re-write the data. If data for one or more of the components already exists, the **Physical Properties System** presents the **Physical Properties Developer** with three options:

- i. Ignore the relevant new data. If this option is selected, the original data is unaltered and no action is taken.
- ii. Create a variant of the existing components (for example with modified names). If this option is selected, the new data is merged with the old data set (as above when no parameter for a component exists).
- iii. Replace the existing parameters. (Note that some Neutral Files may have a facility for marking data “read only”, when this option will not be offered). If this option is selected, the new data replaces the old data set.

The choices may be offered on a chemical component by chemical component basis (and/or a property by property basis). An exception is thrown if interaction parameters exist and one of the pure components is replaced, but not the other. (It would then be unclear whether or not to replace the interaction parameters; it is also probable that the Physical Properties developer is attempting to create an inconsistent data set).

When the above dialogue is completed, the <<**Export Data**>> Use Case is employed to store the modified data set:

Assumptions:

Exceptions: Inconsistent treatment where interaction parameters exist.

Uses:

Extends:

Notes: (1)The <<**Import Data**>> Use Case allows for user interaction to rectify some mismatch between the format of the master data set and the format of the data to be saved. It will be system specific whether that information is used to recreate a data set in the original format, or whether the modified data is stored in the new format. (2) It is expected that the <<**Export Data**>> Use Case handles deleting, overwriting and replacing of data, and the method used will depend on the proprietary Neutral File System.

3.5.4.26 USE CASE: Label Phases.

PRINCIPAL ACTOR: Physical Properties Developer.

Description:

The **Physical Properties Developer** requests of the **Physical Properties System** that he wishes to assign labels (names) to the phases. The **Physical Properties System** replies by giving a list of the components and a list of the phases (with previously assigned labels or blank labels). The **Physical Properties Developer** selects each phase in turn, provides a new label, and indicates the key components that identify that phase. (For example, he may specify that specific components, or groups of components, are present in major proportions, or are absent).

Assumptions :

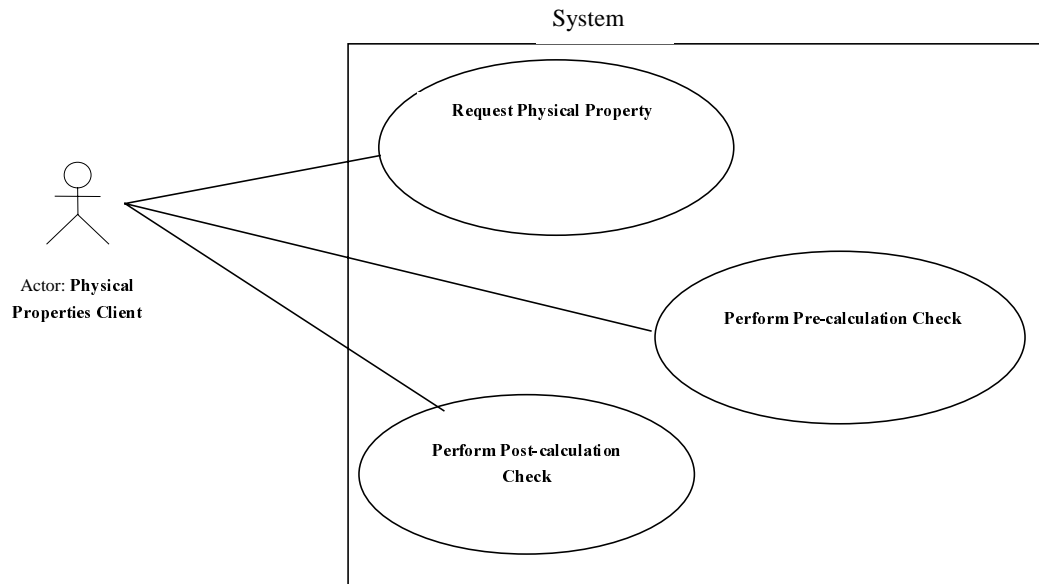
Exceptions :

Uses :

Extends :

Notes: It is expected that this Use Case will be largely proprietary, so does not need elaborating in detail for CAPE-OPEN. Typically, the user would enter “water” as a key component in identifying a phase labelled “aqueous”, and a group of hydrocarbons in identifying an “oil” phase. This type of labelling is an important feature of Physical Properties Systems which deal with multiple liquid and/or multiple solid phases. Where a large number of phases may be present, but it is expected that in most parts of the process only a small subset of the phases will be present, the end user will want to know which phases they have. It is clearly possible to run a simulation with unlabelled phases, or for some default naming to be generated related to the components in large concentration in the phases. This facility is, therefore, optional. CAPE-OPEN should, nevertheless, be aware that it is used by some packages in case it has implications on the information to be passed across interfaces.

3.5.5. Runtime Package



3.5.5.1 USE CASE: Simulate Flowsheet

PRINCIPAL ACTOR: Simulator Executive

Description:

The **Simulator End User** requests that the **Simulator Executive** simulate the flowsheet that he has defined.

The **Simulator Executive** uses the « *check flowsheet* » use case [TO BE DEFINED]. If there is no fatal error, it then calls each unit and asks it to use the « USE CASE: Perform Pre-Calculation Check » use case. The units may be called in any order and any number of times depending on the simulator used. The Executive does not call Physical Properties associated with streams directly, because only the units “know” what properties they will call for. The output routines may be treated as units for this purpose where they output stream properties that might not be required within the units.

If there is no fatal error received from the pre-calculation checks, the **Simulator Executive** then uses the « *run simulation* » use case [TO BE DEFINED, this will be common to all CAPE-OPEN work packages].

If the simulation is successful, the **Simulator Executive** then calls each unit in turn and asks it to use the « USE CASE: Perform Post-Calculation Check » use case. The units may be called in any order and any number of times depending on the simulator used. Again, the output routines may be subject to a similar call to test stream properties to be output.

The **Simulator Executive** then reports the results of the simulation to the **Simulator End User**.

Assumptions :

Exceptions :

Uses : USE CASE: Perform Pre-Calculation Check, USE CASE: Perform Post-Calculation Check

Extends :

Notes:

This use case is global to all packages, it is not part of the Thermo work Package. Only a small proportion is directly relevant to CAPE-OPEN. (2) The pre and post calculation check interfaces will be CAPE-OPEN standard but may not be employed by any given simulator. For example, a simulator may chose not to perform any post-calculation checks, and so give no warnings to the User.

3.5.5.2 USE CASE: Perform Pre-Calculation Check

PRINCIPAL ACTOR: Physical Properties Client

Description:

The **Physical Properties Client** passes a list of one or more physical properties to each **Material Template** with which it is associated, and requests that each check that the properties passed to it can be calculated.

Each **Material** then passes the requests to the routines specified in the associated Physical Properties Package. The **Routines** perform the requested checks, and pass their results back to the **Material Template** which, in turn, passes the results back to the **Physical Properties Client**.

For each component or set of components for which a requested property calculation can be performed, the **Material Template** keeps a record that the property check has been successfully performed.

If the **Physical Properties Client** receives information that a requested property is not available (or cannot be calculated), it analyses the seriousness of this error and passes an appropriate message to the **Simulator Executive**.

Assumptions:

Exceptions :

Requested property is unavailable/cannot be calculated (for some or all components/combination of components).

Uses :

Extends :

Notes: This use case mirrors a similar use case in the UNITS work package. The important task achieved by this test is that it leaves a marker that the property is available. There is no a-priori reason why a package should provide a property required by a particular unit because, until this test, the Materials Template has no knowledge of the requirements of the unit. By setting the marker, the test avoids complex conditional tests in the unit models and provides a convenient way of giving the user a simple, meaningful error message. Whether or not any particular simulator or unit takes advantage of this feature is optional. It is, however, a CAPE-OPEN requirement that interfaces are capable of transferring the information for these tests to be performed.

3.5.5.3 USE CASE: Unit Model Performs Pre-Calculation Check

PRINCIPAL ACTOR: Unit Model.

Description:

As in the «USE CASE: Perform Pre-Calculation Check» use case except:

The **Unit Model** is the **Physical Properties Client**.

Assumptions :

Exceptions :

Uses :

Extends : USE CASE: Perform Pre-Calculation Check

3.5.5.4 USE CASE: Perform Post-Calculation Check

PRINCIPAL ACTOR: Simulator Executive

Description:

The **Physical Properties Client** passes the composition and “state” information together with some named properties and their values to its associated **Material Template** and requests it to check that each property is within the valid range of the calculation routine used.

The **Material Template** passes the request to the routine(s), which have been specified within the associated Physical Properties Package, for each property concerned. The **Calculation Routine(s)** then perform the requested checks and pass the results back to the **material**, which in turn passes the results to the **Physical Properties Client**.

Assumptions :

Exceptions :

Uses :

Extends :

Notes:

Avoids time consuming checks during iterative stage of calculation. The user is generally only concerned whether the final result is correct, not whether every unrealistic step on the path to the solution gave rise to reasonable physical properties. Again, whether individual simulators or units take advantage of the facility is optional. The only CAPE-OPEN requirement is that the interfaces have the facility to transmit the required error-check information.

3.5.5.5 USE CASE: Unit Model Performs Post-Calculation Check

PRINCIPAL ACTOR: Unit Model.

Description:

As in the «USE CASE: Perform Post-Calculation Check» use case except:

The **Unit Model** is the **Physical Properties Client**.

Assumptions :

Exceptions :

Uses :

Extends :

USE CASE: Perform Post-Calculation Check

3.5.5.6 USE CASE: Request Physical Properties

PRINCIPAL ACTOR: Physical Properties Client (Stream, Unit Model, or Simulator Executive)

Description:

The **Physical Properties Client** passes composition and “state” information to an associated **Material Template** and requests it to calculate a property (or set of properties). The **Physical Properties Client** may also pass initial estimates of the requested properties or other parameters used in computing the properties.

The **Material Template** checks to see if it has previously established that it can perform the requested property calculations.

If it has done so, it passes the calculation request to the routine(s), which have been specified within the associated Physical Properties Package, for each requested property. The **Calculation Routine(s)** then perform the requested calculations and pass their results back to the **Material Template**, which in turn passes the results to the **Physical Properties Client**.

If the **Material** finds that it has not previously performed a check on the requested physical properties, then it informs the **Physical Properties Client** of this and does not perform the calculation. [Or it then performs the check, and if it is successful it continues to calculate the requested property, if it cannot it sends an error message to the **Physical Properties Client**].

Assumptions: The conditions provide a sufficient definition of the state of the material.

Exceptions :

Property check has not yet been performed.

Uses :

Extends :

Note :

- i. The material may be of multiple or single phases. In the case of dispersed phases information on the properties of the dispersion are passed as well as their chemical composition. The relevance to CAPE-OPEN is that all the required information can be passed across the standard interfaces. The features of a physical properties system described here are consistent with the earlier decisions made by the Thermo Work Package of CAPE-OPEN.
- ii. The Use Case is expanded in Use Cases 3.5.5.12 to 3.5.5.18.

3.5.5.7 USE CASE: Unit Model Requests Physical Properties

PRINCIPAL ACTOR: Unit Operation (component or subroutine)

Description:

As in the «USE CASE: Request Physical Properties» use case except:
The **Unit Model** is the **Physical Properties Client**.

Assumptions :

Exceptions :

Uses :

Extends :

USE CASE: Request Physical Properties

3.5.5.8 USE CASE: Stream Requests Physical Properties

PRINCIPAL ACTOR: Stream

Description:

As in the «USE CASE: Request Physical Properties» use case except:

The **Stream** is the **Physical Properties Client**.

Assumptions :

Exceptions :

Uses :

Extends :

3.5.5.9 USE CASE: Identify Phase

PRINCIPAL ACTOR: Physical Properties Client (Stream, Unit Model, or Simulator Executive)

Description:

The **Physical Properties Client** passes composition and “state” information of a particular phase to an associated **Material Template** and requests it to return the **Phase Label**.

The **Material Template** passes the request to the **Physical Properties Package**, together with the relevant composition information. The **Physical Properties Package** performs the logical tests and returns the phase label back to the **Material Template**, which in turn passes the results to the **Physical Properties Client**.

Assumptions :

--

Exceptions :

- 1) No label or set of conditions has been defined, when a blank is returned to the client.
- 2) The key component information cannot give an unambiguous phase label, when the response will be Properties system dependent.

Uses :

Extends :

Note :

This information is frequently required for generating caption labels in multi-phase systems. The details of the Use Case will not be CAPE-OPEN standard, but the ability to transmit the necessary text information across the CAPE-OPEN interface is relevant. As for the “request physical properties” Use Case, this case can be replicated for the various alternative clients.

3.5.5.10 USE CASE: Create and populate material object.

PRINCIPAL ACTOR: Unit Model.

Description:

The **Unit Model** selects a **Port** and requests the Stream Object to create a **Material Object** of the type defined by the port's associated Material Template.

The **Unit Model** then loads the **Material Object** with material information (for example, mole numbers, or flow rates, temperature, pressure, enthalpy).

Assumptions:

Exceptions :

The object cannot accommodate the information presented (should be detected by compiler).

Uses :

Extends :

Notes: This Use Case assumes that information on non-standard "stream objects" is passed to the unit through standard "ports". A facility that all "port objects" must have is to be able to create corresponding standard Material Objects.

3.5.5.11 USE CASE: Check material object.

PRINCIPAL ACTOR: Unit Model.

Description:

The **Unit Model** requests that a **Material Object** check itself. The **Material Object** applies a check method to ensure that it is consistently defined. It returns an exception if it is underdefined, overdefined or inconsistently defined.

Assumptions:

Exceptions :

- i) The Object is underdefined (for example, for a molecular mixture, no temperature or enthalpy defined)
- ii) The Object is overdefined (for example, for a molecular mixture, temperature, pressure and enthalpy are defined)
- iii) The Object is inconsistently defined (for example for a 2-phase pure component, temperature and pressure are both defined; this specification is overdefined in the sense that both parameters cannot be specified independently, but underdefined in that the vapour/liquid split cannot be calculated from the given information).

Notes: This Use Case describes an optional action; we cannot insist that a unit model (unit model writer) will make this call, so will have to include a similar test for every property is computed. Alternatively, we could give a hierarchy of specification, for example, if temperature is given, enthalpy values are ignored. This is just an example of a possible check call of this kind; CAPE-OPEN could decide some other form of test. As described, the test is expected to be a quick degrees of freedom test and not, for example, to perform a calculation that determines whether a redundant enthalpy value is actually consistent with other values. It is assumed that each material object includes a self-check method.

3.5.5.12 USE CASE: Compute pure component values of a common property.

PRINCIPAL ACTOR: Unit Model.

Description:

The **Unit Model** requests that a **Material Object** populate itself with a selected common property.

The **Material Object** accesses the appropriate calculation routine in the associated **Physical Properties Package** via a pointer to a middleware registry and computes values for the requested property. The values are then deposited either in the **Material Object** (if it has reserved spaces) or in a locally declared array previously set up in the Unit Model routine.

Assumptions:

Assumptions as for <<**Check material object**>> Use Case.

Exceptions :

- i) The Material Object is insufficiently defined (for example, the test in 3.5.5.11 not done)
- ii) The appropriate routine (method) is not available.

Uses :

Extends :

Notes: We refer here to “common” properties in the sense of properties that are frequently required so that all materials templates will provide methods for accessing them. The necessity of calling via the executive precludes having methods that are “unknown” to the executive. This approach of having methods specific for common properties gives greater run-time efficiency than a general method that applies to any property, which will be described below.

Clearly simple variants of this Use Case cover cases when the appropriate property is recovered for all components in the Material Object, when the appropriate property is returned for only one selected component (when the call must include the component number) or a selected sub-set of components (when the call must include an integer array identifying the selected components).

3.5.5.13 USE CASE: Get identifier to a pure component method (routine) for a less common property.

PRINCIPAL ACTOR: Unit Model.

Description:

The **Unit Model** requests that a **Material Object** obtains an identifier for a method for a less common property. It makes the request by calling a “get identifier” method whilst passing it a text string identifying the method (the same string used within the properties package, and probably the same as in any relevant GUI).

The **Material Object** accesses the appropriate search routine in the associated Physical Properties Package via a middleware registry. The search routine matches the text string passed to it against the text strings that it has for properties. It returns an integer (or other identifier) that on subsequent calls enables the Package to call the appropriate routine immediately without the necessity for a search. The Package then returns the identifier to the calling method of the Material Object (via appropriate middleware registries).

An exception is thrown if the text string cannot be matched.

Assumptions:

The assumptions are as for the “check” use case above.

Exceptions :

Text string cannot be matched.

Uses :

Extends :

Notes:

i) We refer here to “less common” properties in the sense of properties that are infrequently required so that materials templates will not provide methods for accessing them. The call to the Properties Package via the executive is then via a generic “get (identifier, ..)” method that does not require the executive to “know” any of the properties that may be accessed, or their identifiers.

ii) By sending a text string that (with suitable separators) lists a set of properties, several properties can be identified. The returned identifier then provides direct access to several properties that will subsequently be returned in one property call. This seems the simplest way of accessing multiple properties. For “Common Properties”, we describe alternative methods below.

3.5.5.14 USE CASE: Compute pure component values of a less common property.

PRINCIPAL ACTOR: Unit Model.

Description:

The Unit Model first sets aside space for storing the result (it is assumed that the template will not have space for less common properties). The **Unit Model** requests that a **Material Object** fetch a less common property (or set of less common properties), the property required being specified by the identifier created by the << **Get identifier to a pure component method (routine) for a less common property** >> Use Case.

The **Material Object** accesses the associated Physical Properties Package via a pointer to a middleware registry (the pointer is to a “generic fetch” method). The Properties Package identifies the properties routine(s) required by the identifier passed to it. It throws an exception if the identifier does not match a known property (it is expected that all property identifiers are in a range set up by the package so that invalid values are immediately identifiable). The selected routine computes values for the requested property. The values are then deposited in the locally declared array previously set up in the **Unit Model** routine.

Assumptions: As for <<Get identifier to a pure component method (routine) for a less common property>> Use Case.

Exceptions :

- i) The Material Object is insufficiently defined (For example, the test in 3.5.5.11 not done)
- ii) The property identifier is invalid.

Uses : << **Get identifier to a pure component method (routine) for a less common property** >> Use Case.

Extends :

Notes:

- i) As for “Get identifier to less common property” Use Case.
 - ii) The same options apply as for common properties in accessing values for 1 component, all components, or any selected sub-set of components.
 - iii) Multiple properties may be returned; see note (ii) of <<Get identifier to a pure component method (routine) for a less common property>> Use Case.

3.5.5.15 USE CASE: Compute mixture properties for a defined phase, or defined phases.

PRINCIPAL ACTOR: Unit Model.

Description:

The Use Case is similar to the “get pure properties” Use Cases except:

- i) The properties for the complete mixture are always obtained, so that it is not required to pass any list of selected components.
- ii) The selected phase (or phases must be transmitted to the Physical Properties Package). For very frequently occurring properties, where run-time efficiency is at a premium, there may be special methods for selected phases.
- iii) Where multiple properties are requested for one given mixture (Material Object), the method used for “less common properties” may be employed or, where run-time efficiency is at a premium, special methods that return selected sets of properties (eg those required in distillation calculations) may be provided.

Assumptions:

Exceptions :

- i) The Material Object is insufficiently defined (for example, the test in 3.5.5.11 not done)
- ii) The property identifier is invalid.

Uses :

Extends :

Notes: The Use Case differs slightly from the cases implied by the Thermo Work Package deliberations previously. Thus we anticipate that, for commonly required properties, special methods will be provided rather than the generic multiple properties procedure described for “less common” properties. The approach here is both intrinsically more run-time efficient and allows more efficient organization of the calculation within the Physical Properties Package. Thus, when faced with an arbitrary selection of properties, the Package has to find an efficient path through its calculations to minimize repetition of iteration, whereas by being asked for a standard set of properties, an efficient calculation path can be programmed in with no requirement for any run-time organization.

3.5.5.16 USE CASE: Compute pure component values of a common property of a stream.

PRINCIPAL ACTOR: Unit Model.

Description:

The **Unit Model** requests that a **Stream Object** populate itself with a selected common property.

The **Stream Object** accesses the appropriate calculation routine in the associated Physical Properties Package via a pointer to a middleware registry and computes values for the requested property. The values are then deposited either in the **Stream Object** (if it has reserved spaces) or in a locally declared array previously set up in the Unit Model routine.

Assumptions: See note (ii) to “Exceptions”.

Exceptions :

i) The **Stream Object** is insufficiently defined (For example, the test equivalent to 3.5.5.11 not done). It is assumed that, for all input streams, the Objects will be fully defined and will not require a check.

ii) The appropriate routine (method) is not available.

Uses :

Extends :

Notes: The Use Case is basically similar to the corresponding <<Compute pure component values of a common property>> with the exception that the **Stream Object** replaces the **Material Object**. The Stream Object will have a simulator dependent structure whereas the assumption is that the Material Object will have a CAPE-OPEN standard structure. It follows that the calling mechanism, in this case, is required to go via the simulator executive (whether or not the component set differs from that of the Properties Package) because only the Executive knows the internal stream structure. Similarly access to all stream properties must be via “methods” whereas for a Material Object, some of the property values may be in directly accessible locations.

3.5.5.17 USE CASE: Compute property values for a stream.

PRINCIPAL ACTOR: Unit Model.

Description:

This heading covers a whole set of Use Cases which directly mirror the corresponding Material Object Use Cases. The difference is that these Use Cases refer to a Stream Object that will be entirely simulator dependent in structure. The Stream Object is accessed through a Standard Port.

Assumptions:

Exceptions :

--

Uses :

Extends :

Notes:

3.5.5.18 USE CASE: Assign Material Object to Stream Object.

PRINCIPAL ACTOR: Unit Model.

Description:

The Unit Model assigns a Material Object to an Output Stream. The assignment operator (which will be within the Output Stream Object) copies the property values from the CAPE-OPEN standard Material Object format to the proprietary Stream format at the selected port. It checks that there are no inconsistencies (namely that the Material Object) was created using a method from the current output stream, or a stream that offered the same methods. An exception is thrown if the Output Stream format is inconsistent with the Material Object format.

Some simulators will create Stream Objects that throw an exception whenever an attempt is made to assign values to Input Streams. Others will throw an exception only when an attempt is made to assign values to input streams connected from other units (such systems will include, in their object structure information on the source of the input streams).

Assumptions: See note (ii) to “Exceptions”.

Exceptions :

i) The **Stream Object** is inconsistent with the **Material Object**. By “inconsistent” it is meant that it stores different information, or has different numbers of components.

ii) The stream object cannot be altered (for example it is an input stream).

Uses :

Extends :

Notes:

- i) (See exception i) It is not implied that the Stream and Material objects should have any common ancestry (although they might). The Stream Objects come from a proprietary structure whereas the Material Objects have a CAPE-OPEN defined structure (at least in their base classes)
- ii) (see exception ii) Since the Stream Object is a proprietary (non-standard) object, it can include whatever information it needs to record that it cannot be altered. Most (but not all) simulators will forbid assignment of values to input streams. It follows that a unit model that assigns values to input streams will be less portable than one that does not. The copy method will also be proprietary but will, of course, know about the standard Material Object (indeed, it will include an “associated” Material Object from which Material Objects can be created). This association has been one of the major features agreed by CAPE-OPEN.

3.5.5.19 USE CASE: Assign Components to Material Template

PRINCIPAL ACTOR:Unit Model

Description:

The **Unit Model** indicates to the **Materials Template** (and through that to the **Physical Properties Executive**) that it wishes to assign new components and/or physical properties parameters to the Physical Properties Package.

The **Physical Properties Executive** uses the <<**Retrieve Physical Properties Package**>> Use Case to transfer the properties from the Unit Model to the Properties Package.

Assumptions: The Unit Model has a CAPE-OPEN standard **Neutral File System** interface so that the properties are loaded in the same way as from a neutral file. It is expected that this interface will pass primarily numeric data because the standard **Neutral File System** interface will require that the File System software converts any alphanumeric coding of numeric data to numeric (eg to a binary format).

Exceptions: It is not expected to have exceptions because the same properties routines will be used throughout a simulation, only the number of components and values of parameters will be changed.

Uses: <<Retrieve Physical Properties Package>>

Extends:

Note: This Use Case is intended to deal with petroleum (and other) pseudo-components. Where a complete set of physical properties are computed from a few key parameters (normal boiling point etc), there must also be an interface estimation routine. There has been no User Requirement to provide standard interfaces to such routines.

3.5.6. Neutral File Interface

The Neutral File System consists of a (set of) Neutral File(s) and a Neutral File Executives which manages access to the files. There is a component software interface between the Neutral File system and the Physical Properties System.

There is a similar interface with the Simulator. It has to be decided whether the interface to the Simulator is the same as (identical to) the interface to the Physical Properties System, or whether it is merely similar. It also has to be decided whether the same neutral file holds connectivity information (from the Simulator) and physical properties data (from the simulator) or whether these two classes of information are held on distinct files and file types. If different files are used, it would be further necessary to determine on which file the materials template information was held.

The CAPE-OPEN standard interface is the interface between the software components. Different commercial Neutral File Systems will access different neutral files. At the time of writing the neutral files being considered include an enhanced Dechema (DPPS) system for physical properties data and a (possibly augmented) pdXi system for the more equipment and process data.

This section also includes consideration of neutral file access to Unit Model data, which requires collaboration with the CAPE-OPEN UNIT work package to refine.

3.5.6.1 USE CASE: Install Neutral File System

PRINCIPAL ACTOR: Simulation Engineer or Physical Properties Developer.

Description:

As <<Install CAPE-OPEN Compliant Software Component>> except:

The Neutral File Set Up Manager acts as the St Up Manager.

Assumptions:

Exceptions:

Uses:

Extends: Install CAPE-OPEN Compliant Software Component.

Note: The Neutral File System is a software component that reads and writes data files and which can be driven through its component software interface by a Simulator or Physical Properties System. It may also have its own User Interface (for example to select which file to read from or to change a file format, such as counting upwards 1 to n instead of downwards from n to 1).

3.5.6.2 USE CASE: Load (additional) Neutral File System

PRINCIPAL ACTOR: Physical Properties Developer or Simulation Engineer.

Description: The **Physical Properties Developer** requests that a **Neutral File System** (or an additional Neutral File System) be loaded into the Physical Properties System.

The **Physical Properties System Executive** acknowledges the request and gets the list of CAPE-OPEN compliant **Neutral File Systems** from the CAPE-OPEN registry. It communicates this list to the **Physical Properties Developer** and asks him to select one from the list. He selects one or aborts.

If the Physical Properties Developer selects one of the available **Neutral File Systems**. Its identifier, and its list of data sets now become known to the Physical Properties System.

If the User aborts the operation, no action is taken.

Assumptions:

Uses:

Extends:

Notes:

- i. Similar to Use Case **Load Additional Physical Properties System**.
- ii. There would be an exactly similar User Case for registering a File System with the Simulator

3.5.6.3 USE CASE: Import Data.

PRINCIPAL ACTOR:Physical Properties System

Description:

The **Physical Properties System** sends a message to the **Neutral File System** listing the information that it requires, for example, name of required property, names of columns or rows. The **Neutral File System** interrogates its associated file and, if there is an exact match with the requested data, reads it from the file and transfers it to the **Physical Properties System**. Where there is not an exact match, either an exception is thrown, or the **Physical Properties Developer** is presented with a list of data blocks available, and the identifiers of the requested data. The **Physical Properties Developer** is also presented with a dialogue mechanism which enables him to define a correspondence between the requested and the available data. The correspondence may require only matching labels (for example, Methanol = CH₃ OH) or may require a renumbering of columns and rows and/or an inversion of columns and rows. The **Physical Properties Developer** may also have the option of entering or padding missing information, and of identifying surplus data that is to be ignored. If the dialogue is successful, the information is read from file, modified if necessary, and transferred to the **Physical Properties System**. If the dialogue is unsuccessful, an exception is thrown.

Assumptions: (See note)

Exceptions: Data cannot be located. Dialogue to transform data is unsuccessful.

Uses:

Extends:

Notes: (1) It is expected that the <<Export Data>> Use Case will cause data to be written in exactly the form it needs to be read by the <<Input Data>> Use Case. (2). Whether or not a dialogue box option is offered will depend on the **Neutral File System** being interrogated. (3) The **Neutral File System** may write to a human-readable file (such as DPPS from Dechema), an ISO format (such as PI-STEP/pdXi), or to a commercial encrypted file. (4) Where a **Neutral File System** links to several files, it will present a dialogue for the **Physical Properties Developer** to select the specific file required.

3.5.6.4 USE CASE: Export Data

PRINCIPAL ACTOR:Physical Properties System.

Description:

The **Physical Properties System** sends a message to the **Neutral File System** together with the information that it requires to be saved by the **Neutral File System**. The **Neutral File System** saves the information (data) exported on its associated file.

Where the **Neutral File System** may be attached to more than one file, it presents the **Physical Properties Developer** with a dialogue box enabling a specific file to be selected.

Assumptions:

Exceptions:

Uses:

Extends:

Notes: (1) It is expected that the <<**Import Data**>> Use Case will read data in exactly the form written by this Use Case. (2) Whether or not a dialogue box option is offered will depend on the **Neutral File System** being employed and whether it is connected to more than one file. (3) The **Neutral File System** may write to a human-readable file (such as DPPS from Dechema), an ISO file format (such as PI-STEP/pdXi) or to a commercial encrypted file.

3.5.6.5 USE CASE Change Data Format

PRINCIPAL ACTOR: Physical Properties Developer

Description: The **Physical Properties Developer** requests the **Physical Properties System** to change Neutral File formats. The **Physical Properties System** presents the **Physical Properties Developer** with a list of attached **Neutral File Systems**. The **Developer** selects one as the Source Systems and another as the destination system.

The <<**Import Data**>> Use Case is employed to import data from the source System and the <<**Export Data**>> Use Case is employed to export data to the destination system.

Assumptions:

Exceptions:

Uses: <<**Import data**>>, <<**Export Data**>>.

Extends:

Notes: We have not described a system for protecting encrypted data from being read from an encrypted file and written to a non-encrypted file