CAPE-OPEN

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

Open Interface Specification:
Simulation Context COSE Interface

www.colan.org

ARCHIVAL INFORMATION

Filename	Simulation Context COSE Interface.doc
Authors	CO-LaN consortium
Status	Public
Date	August 2003
Version	version 2
Number of pages	27
Versioning	version 2, reviewed by Jean-Pierre Belaud, August 2003
	version 1, Methods & Tools group, November 2001
Additional material	
Web location	www.colan.org
Implementation specifications version	CAPE-OPENv1-0-0.idl (CORBA) CAPE-OPENv1-0-0.zip and CAPE-OPENv1-0-0.tlb (COM)
Comments	

IMPORTANT NOTICES

SUMMARY

This document describes the Simulation Context COSE Interface. This interface proposes to deal with the need for a PMC to request services from the PME, rather than they other way around, which is the normal configuration. Enabling the PMCs to benefit from centralised PME services will enhance the integration of the system. At the same time, it will prevent from forcing the PMC developers to write code for tasks already present in most PME.

The *COSE Interfaces* are defined by the Methods & Tools group. They own horizontal interface specifications dedicated to the PME side and are interfaces for environment simulation such as simulator executive (hence the naming **COSE for CAPE-OPEN Simulator Executive**). The COSE Interfaces defines services of general use provided by PME in order to be called by PMCs through a call back usage.

The Simulation Context Interface specification gathers three functionalities, **Diagnostic, Material Template System and Utilities**. That results to three interfaces, one for conveying verbose information to the PME, one for allowing the unit to choose between all the Thermo Material factories supported by the COSE, and one for requesting diverse values from the PME using a holdall interface concept. These three interfaces are straightforward and belong to the Simulation Context specification.

ACKNOWLEDGEMENTS

CONTENTS

LIST OF FIGURES

1. Introduction

The Interoperability Task Force tested the CAPE-OPEN standard by taking CAPE-OPEN components and making sure that they can be used in CAPE-OPEN compliant Simulation Executives. As part of this exercise the task force has identified some areas where the standard needs to be improved or extended to allow sophisticated components to be developed.

The Unit specification versions 0.9 and 0.93 included a method called SetSimulationContext. Its intention was to provide the PMC with a reference to a PME object that could provide some standard functionality. Unfortunately, since no interface was defined for the PMC to communicate with the PME, the method was useless. Due to its utility, instead of discarding the method, it was decided to move it into one of the *Common Interfaces*, so that any kind of PMC could benefit of this functionality.

It is extremely important to define the Simulation Context interface in order to avoid proprietary solutions to be implemented by diverse vendors. A similar situation occurred with ICapeThermoMaterialTemplate interface. This interface was defined by the Thermo specification document, but no methodology had been defined to access or creates instances of objects that implemented these interfaces.

The Interoperability Task Force proposed that, instead of a single Simulation Context interface, there should be a number of small interfaces belonging to the Simulation Context. The advantage with this approach is that there is no need to define all the interfaces at once, since the definition of a Simulation Context may be extended by adding new interfaces.

The Methods & Tools Integrated Guidelines document introduces the new category for *COSE Interfaces*, defining them as horizontal interface specifications dedicated to the PME side.

2. Requirements

2.1 Textual requirements

The Interoperability Task Force detected the need for supporting the following Simulation Context functionalities, which have in common the need of a PMC to request services from its unique hosting PME:

- The **communication of verbose information** from the PMC to the PME (and hence to the user). PMCs should be able to log or display information to the user while it is executing a flowsheet. Rather than each PMC performing these tasks by the means of different mechanisms, it is much preferable to redirect them all to the PME services for communicating with the user. The Error Common Interfaces do not fulfil these requirements, since they stop the execution of the PMC code and signal an abnormal situation to the PME. The document deals with the transferral of simple informative or warning messages.

- When a Unit Operation needs to obtain thermodynamic calculations, it will typically perform them on the material objects attached to the Unit ports. However, in some cases, like distillation columns, there may be the need to utilise a different Property Package. Even the user could be requested to choose which thermodynamic model to must be used. All the mechanisms for **accessing CAPE-OPEN Property Packages** are already in the COSE´s, as part of the functionality necessary for making use of CAPE-OPEN Property Packages. Therefore, instead of each PMC implementing support for performing this selection and creation of thermo engine, delegating that responsibility to the COSE will result in thinner and easier to code Unit Operation Components. If configuration of Material Templates is in the PME side, the only additional functionality the Unit Operation would require is that for accessing the list of already configured Material Templates, and picking one of them.

- When a PMC is wrapping a FORTRAN dll, there may be a technical problem when the PMC is loaded in the same process as the PME such as Simulator Execution. In this case, there may be a clash between different FORTRAN modules if two of them select the same output channel for FORTRAN messaging. Hence the PME should centralise the generation of unique output channels for each PMC that may require them. This requirement only occurs when PME and PMC belong to the same computing process, obviously this FORTRAN channel functionality is only applicable when the architecture is not distributed. As we can have in the future this kind of information to exchange, a generic and extensible mechanism has to be set up. The calling pattern is a good candidate. Thus a specific string value for FORTRAN channel would be standardised.

2.2 Use-Cases

2.2.1 Actors

- **Flowsheet User.** The person who uses an existing flowsheet. This person will put new data into the flowsheet, rather than change the structure of the flowsheet.

- **Simulator Executive.** The part of a simulator whose job it is to create, or load, a previously stored flowsheet, solve it and display the results.

- **PMC**. Any type of PMC inserted in a simulation flowsheet.

- **Flowsheet Unit.** A software representation of a physical unit operation, or a non-physical unit such as a controller or optimizer.

2.2.2 List of Use Cases

- **UC-001**: Convey Critical Information
- **UC-002**: Convey additional Information
- **UC-003**: Create a material object with a particular configuration
- **UC-004**: Obtain a Unique Fortran channel

Use Case Categories:

- **Diagnostic**: Uses case related to the conveying of verbose information to the PME.
- **Material Template**: Uses case for requesting the creation of material objects.
- **COSE value**: any kind of value that the COSE must centralise but may be include in a wider category.

2.2.3 Use Cases Maps

2.2.4 Use Cases

UC-001 CONVEY CRITICAL INFORMATION

Actors: PMC

Priority: low

Classification: Diagnostic

Context:

Pre-conditions: A PMC is executing one of its methods. The PME must have previously passed the PMC a reference to its CAPE-OPEN interfaces.

Flow of events: The PMC decides to communicate the user a piece of information of critical importance. The PMC wants to make sure that the user will receive the message before resuming the calculations.

Post-conditions: The user has received the informative message.

Errors:

Uses:

Extends:

UC-002 CONVEY ADDITIONAL INFORMATION

Actors: PMC

Priority: low

Classification: Diagnostic

Context:

Pre-conditions: A PMC is executing one of its methods. The PME must have previously passed the PMC a reference to its CAPE-OPEN interfaces.

Flow of events: The PMC decides to communicate the user a piece of information of low importance. It is enough if the user receives the information after the flowsheet has been totally calculated.

Post-conditions: The Simulator Executive has received the informative message, and will use its tracing or logging mechanism to make it available to the user.

Errors:

Uses:

Extends:

UC-003 CREATE A MATERIAL OBJECT WITH A PARTICULAR CONFIGURATION

Actors: PMC (usually a Flowsheet Unit)

Priority: low

Classification: Material template

Context: A flowsheet user is configuring a PMC.

Pre-conditions: A PMC is executing one of its methods. The PME must have previously passed the PMC a reference to its CAPE-OPEN interfaces.

Flow of events: The PMC requests the PME the list of available types of material template. The PMC might automatically select one of them, or request the user for manual selection through its custom Graphical User Interface or public parameters. After that the PMC will request the creation of the desire type of material template. Once the PMC obtains a reference to this material template, it may be used to create as many instances of material object as required.

Post-conditions: The PMC obtains a new instance of a material object of the desired type.

Exceptions: <The PME does not support the creation of any material template>

<The PME does not support the creation of any material template of the particular type required by the PMC or the user>

Uses:

Extends:

UC-004 OBTAIN A UNIQUE FORTRAN CHANNEL

Actors: PMC

Priority: high

Classification: COSE value

Context: The PMC is setting up its internal legacy FORTRAN modules.

Pre-conditions: The PME must have previously passed the PMC a reference to its CAPE-OPEN interfaces.

Flow of events: The PMC checks that the PME supports the generation of unique FORTRAN channels. The PMC requests the PMC a unique FORTRAN channel for each FORTRAN module it contains.

Post-conditions: The PMC FORTRAN modules may safely work with unique FORTRAN channels.

Errors: <The PME does not support generating unique FORTRAN channels > Uses: Extends:

2.3 Sequence diagrams

SQ-001 CONVEY REFERENCE TO PME INTERFACES

This sequence is detailed and explained within the Utilities Common Interface document. All the posterior sequence diagrams have SQ-001 as a prerequisite.

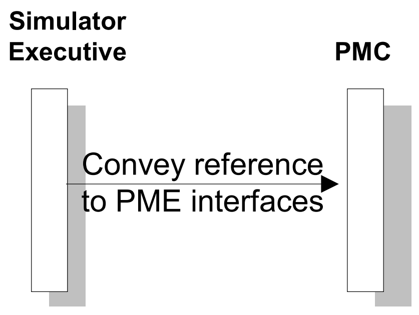


Figure 1 Convey reference to PME interfaces

SQ-002 CONVEY CRITICAL INFORMATION

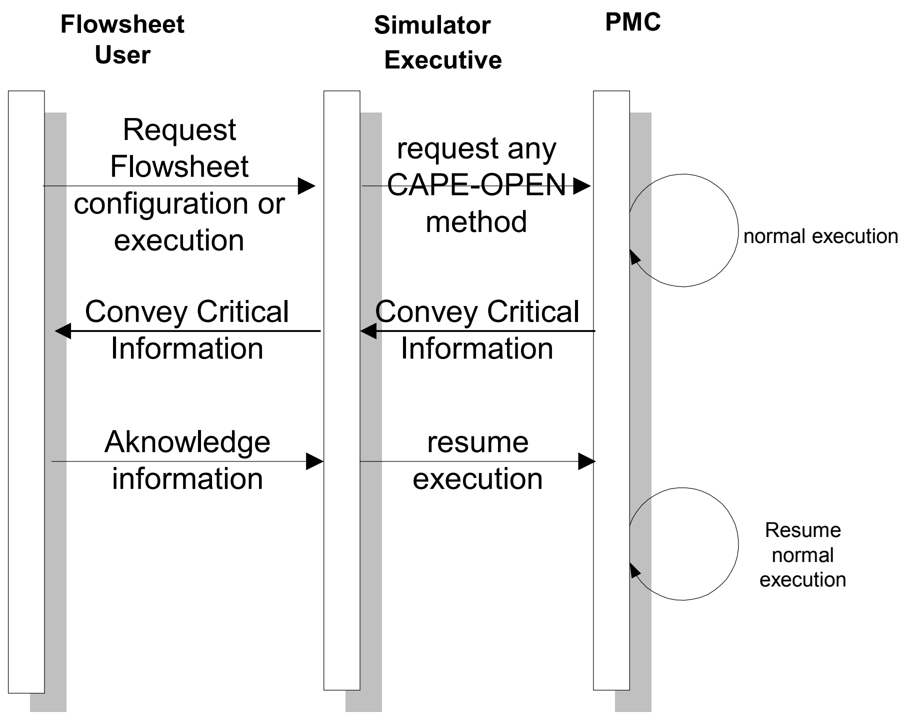
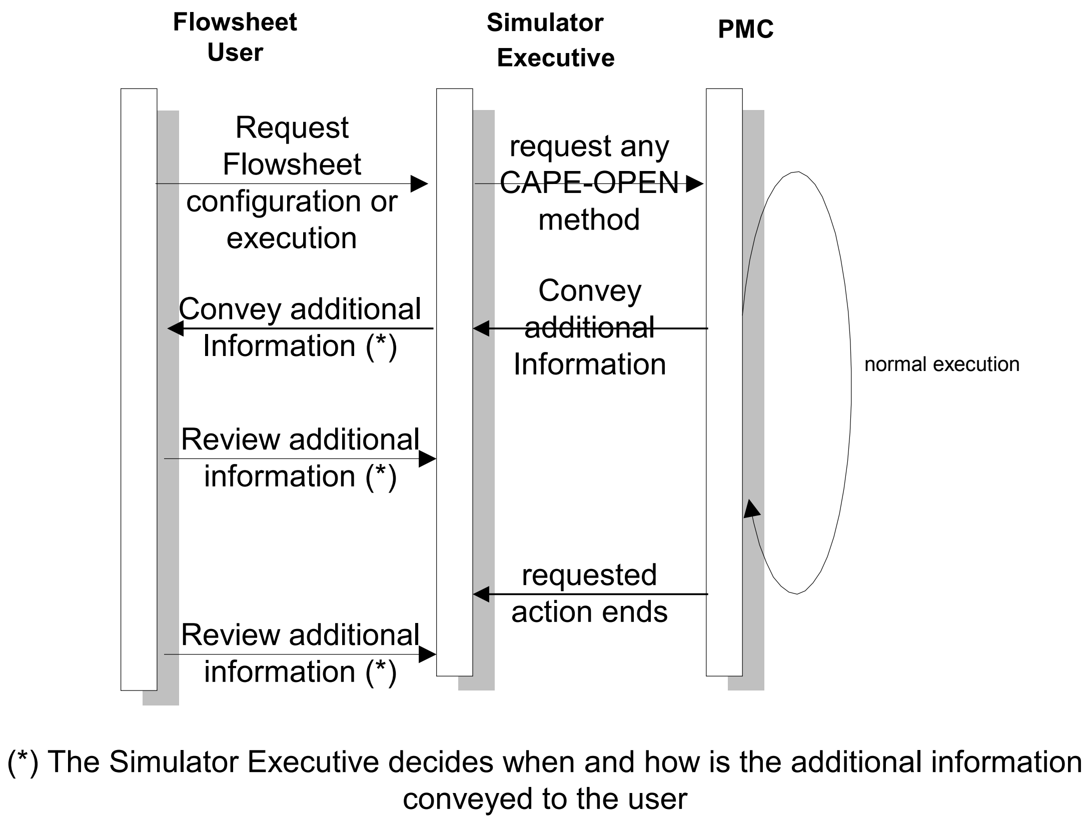

Flowsheet User
Simulator Executive
PMC
Request Flowsheet configuration or execution
request any CAPE-OPEN method
normal execution
Convey Critical Information
Convey Critical Information
Aknowledge information
resume execution
Resume normal execution

Figure 2 Convey critical information

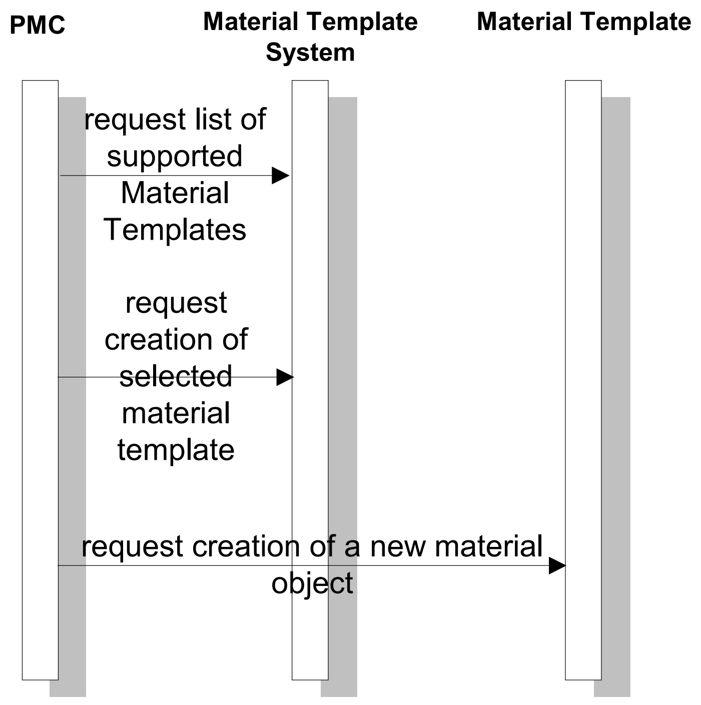
SQ-003 CONVEY ADDITIONAL INFORMATION



Flowsheet User
Simulator Executive
PMC
Request Flowsheet configuration or execution
request any CAPE-OPEN method
Convey additional Information (*)
Convey additional Information
normal execution
Review additional information (*)
requested action ends
Review additional information (*)
(*) The Simulator Executive decides when and how is the additional information conveyed to the user

Figure 3 Convey additional information

SQ-004 CREATE A MATERIAL OBJECT WITH A PARTICULAR CONFIGURATION



PMC
Material Template System
Material Template
request list of supported Material Templates
request creation of selected material template
request creation of a new material object

Figure 4 Create a material object

SQ-005 OBTAIN A UNIQUE FORTRAN CHANNEL

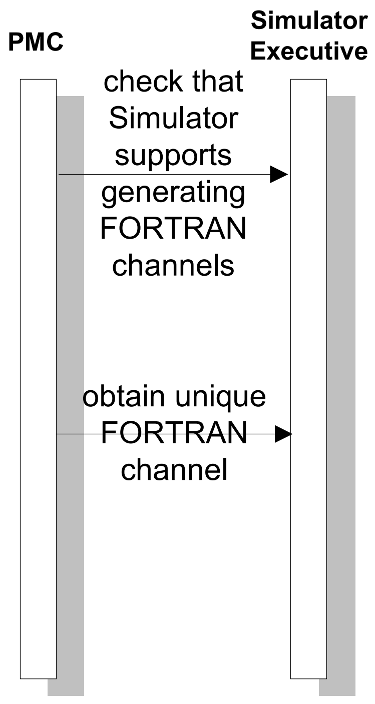

PMC
Simulator Executive
check that Simulator supports generating FORTRAN channels
obtain unique FORTRAN channel

Figure 5 Obtain a unique FORTRAN channel

3. Analysis and Design

3.1 Overview

The Simulation Context implies three major interfaces:

(i) ICapeDiagnostic which encloses the diagnostic functionality

(ii) ICapeMaterialTemplateSystem which allows to manage the material template

(iii) ICapeCOSEUtilities which provides a holdall concept. That allows gathering many (very) basic functionalities without links between them within a single interface. At present it contains operations handling standardised values. Later other operations can be easily added in this utility interface.

The Simulation Context interfaces are implemented by PME.

3.2 Sequence diagrams

3.3 Interface diagrams

This section presents the design of the Simulation Context interface.

IN-001 INTERFACE DIAGRAM

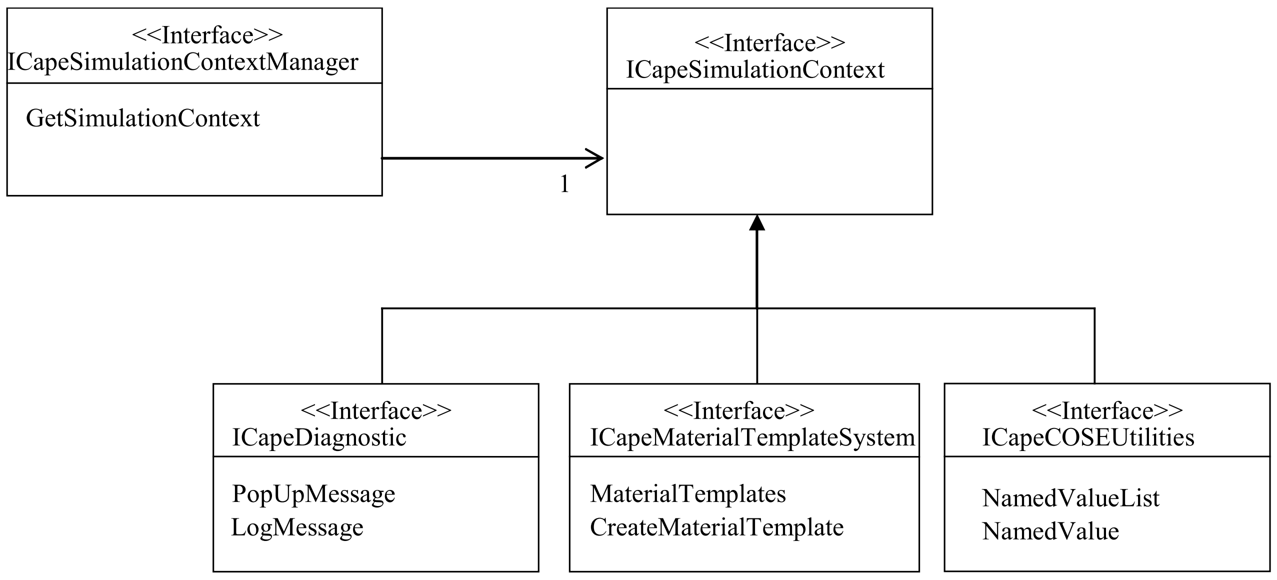


Figure 6 Interface diagram

The section 3 comes from a conceptual view which is independent from the distributed platform. However it is worth noting that actually the interfaces ICapeSimulationContextManager and ICapeSimulationContext don't exist in COM. In this middleware, the other three interfaces are independent, although they're always implemented by a single COSE object.

3.4 State diagrams

3.5 Other diagrams

CO-001 COMPONENT DIAGRAM

Before a PMC may use any of the PME COSE interface, the PME use the PMC's ICapeUtilities:SetSimulation in order to pass a reference to them.

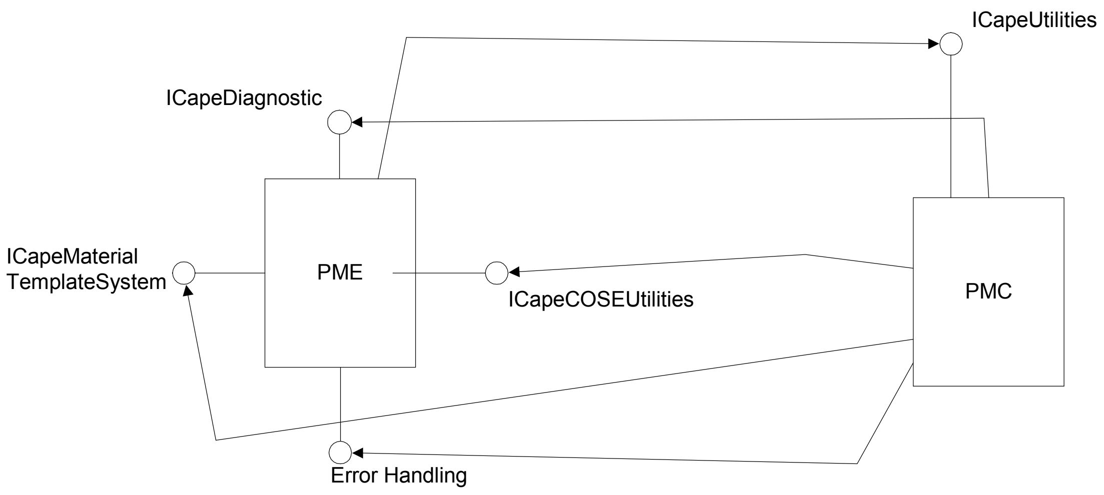


Figure 7 Component diagram

As recommended by the Methods & Tools Integrated Guidelines, the *COSE interfaces* (such as here Simulation Context) are only related to one common interface that is Error Common Interface.

3.6 Interfaces descriptions

Each interface is presented together with its corresponding methods.

3.6.1 ICapeDiagnostic

Interface Name	ICapeDiagnostic
Method Name	PopUpMessage
Returns	--

Description

Write a string to the terminal.

This method is called when a message needs to be brought to the user's attention. The implementation should ensure that the string is written out to a dialogue box or to a message list that the user can easily see.

A priori this message has to be displayed as soon as possible to the user.

Arguments

Name	Type	Description
[in] message	CapeString	The text to be displayed.

Errors

ECapeUnknown, ECapeInvalidArgument

Interface Name	ICapeDiagnostic
Method Name	LogMessage
Returns	--

Description

Write a string to a log.

This method is called when a message needs to be recorded for logging purposes. The implementation is expected to write the string to a log file or other journaling device.

Arguments

Name	Type	Description
[in] message	CapeString	The text to be logged.

Errors

ECapeUnknow, ECapeInvalidArgument

3.6.2 ICapeMaterialTemplateSystem

Interface Name	ICapeMaterialTemplateSystem
Method Name	MaterialTemplates
Returns	CapeArrayString

Description

Returns StringArray of material template names supported by the COSE. This can include:

- CAPE-OPEN standalone property packages
- CAPE-OPEN property packages that depend on a Property System
- Property packages that are native to the COSE.

Arguments

None

Errors

ECapeUnknow, ECapeNoImpl

Interface Name	ICapeMaterialTemplateSystem
Method Name	CreateMaterialTemplate
Returns	CapeInterface (ICapeThermoMaterialTemplate)

Description

Creates a new thermo material template of the specified type.

Arguments

Name	Type	Description
[in] materialTemplate Name	CapeString	The name of the material template to be resolved (which must be included in the list returned by MaterialTemplates)

Errors

ECapeUnknow, ECapeInvalidArgument, ECapeNoImpl

3.6.3 ICapeCOSEUtilities

Interface Name	ICapeCOSEUtilities
Method Name	NamedValueList
Returns	CapeArrayString

Description

Returns StringArray of list of named values supported by the COSE. List of defined named values:

Arguments

None

Errors

ECapeUnknow

Interface Name	ICapeCOSEUtilities
Method Name	NamedValue
Returns	CapeVariant

Description

Returns the value corresponding to the value named *name*. Be aware that two consecutive calls passing the same *name* may return different values.

Arguments

Name	Type	Description
[in] name	CapeString	Name of the requested value (which must be included in the list returned by NamedValueList)

Name	Description	Return value Type
FreeFORTRANchannel	The COSE will return a different FORTRAN channel each time this NamedValue is called for this property. The COSE may not use any of the returned FORTRAN channels for any internally used FORTRAN module.	CapeLong

Errors

ECapeUnknow, ECapeInvalidArgument

3.7 Scenarios

4. Interface Specifications

4.1 COM IDL

```
// You can get these intructions in Cose.idl file from CAPE-OPENv1-0-0.zip
```

4.2 CORBA IDL

```
// You can get these intructions in CAPE-OPENv1-0-0.idl within the
CAPEOPEN100::Cose::SContext module
```

5. Notes on the interface specifications

5.1 COM IDL

As described in section 3.3, the interfaces ICapeSimulationContextManager does not exist in COM. In this middleware, the COSE will call the PMC's method ICapeUtilities::SetSimulationContext passing an IDispatch of one of its objects. The PMC will then be able to convert it (through standard QueryInterface), to each of the ICapeDiagnostic, ICapeMaterialTemplateSystem and ICapeCOSEUtilities, independently. Be aware that not all COSEs will expose all these three interfaces.

5.2 CORBA IDL

The module SContext which contains all the interfaces defined by this specification belongs to the CORBA Cose module.

The CORBA specification has a manager called ICapeSimulationContextManager which returns the reference of Simulation Context objects which implements the interfaces such as ICapeDiagnostic, ICapeMaterialTemplateSystem and ICapeCOSEUtilities.

In fact the manager interface does not imply that these PME objects are created by the manager object. That is software implementation dependant, the Simulation Context PME objects can be created on demand, when the manager operation is called, or the PME has already built its PME objects during its internal processes.

In order to get the right reference an enum is defined: CapeSimulationContextType {DIAGNOSTIC,MTS,UTILITIES};

Note that the ICapeSimulationContextManager reference is passed to any PMC through the Common::Utilities::ICapeUtilities::SetSimulationContext() operation. This operation is described in the Utilities Common Interface specification document. Then through a call-back usage, the PMC when necessary can use the ICapeSimulationContextManager in order to access to the Simulation Context interfaces.

6. Prototypes implementation

The PMC must keep the Simulation Context pointer.

```
CapeInterface m_simulationContext=null;

HRESULT simulationContext([in] CapeInterface simContext){
    m_simulationContext = simContext
}
```

6.1 Diagnostics

- ❑ Step 1: Get the ICapeThermoTemplateSystem pointer.

```
ICapeDiagnosticPtr diagnostics;
diagnostics = m_simulationContext;
If (diagnostics==NULL) {
//ICapeDiagnostic not supported by theCOSE
}
```

- ❑ Step 2 send message to COSE

```
//a problem is detected
if (problemIsCritical)
      diagnostics.PopUpMessage("critical problem detected")
else
      diagnostics.LogMessage("Recoverable problem detected")
```

6.2 Material Template System

- ❑ Step 1: Get the ICapeThermoTemplateSystem pointer.

```
ICapeMaterialTemplateSystem templateSystem;
if (m_simulationContext==null)
      //The simulation context was not set by the COSE
else {
      if (FAILED(m_simulationContext.QueryInterface(
      IID_ICapeThermoTemplateSystem,
            &templateSystem)))
{
      //ICapeThermoTemplateSystem not supported by the COSE
}
}
```

- ❑ Step 2: Get the names of all the supported material templates and choose one

```
CapeArrayString supportedTemplates;
supportedTemplates=templateSystem.MaterialTemplates();
//The Unit can
//  *display the list of supported templates through the Edit dialog
//   and the user will choose one.
//  *Or check that the desired Material Template is supported.
```

- ❑ Step 4: Get the pointer to the desired Material Template

```
CapeString desiredMT; //set in step 3
CapeInterface MTdispatch;
ICapeThermoMaterialTemplate matTemplate;

MTdispatch=templateSystem.CreateMaterialTemplate(desiredMT);
if (FAILED(MTdispatch.QueryInterface(
      IID_ICapeThermoMaterialTemplate,
            &matTemplate)))
{
      //ICapeThermoMaterialTemplate not exposed
}
```

- ❑ Step 5: Create a material Object

```
CapeInterface MOdispatch;
ICapeThermoMaterialObject matObject;

MOdispatch=matTemplate.CreateMaterialObject();
if (FAILED(MOdispatch.QueryInterface(
       IID_ICapeThermoMaterialObject ,
              &matObject)))
{
       //ICapeThermoMaterialObject not exposed
}
```

- ❑ Step 6: Configure the Material Object, request the flash and get results

```
matObject.SetProp(“temperature”,....)
matObject.SetProp(“pressure”,....)
matObject.SetProp(“fraction”,....)
matObject.SetProp(“flow”,....)
matObject.CalcPpop(<"fugacityCoefficient">, ....)
fugacityCoefficientV =matObject.GetProp(“fugacityCoefficient”,....)
```

6.3 Named Values

- ❑ Step 1: Get the ICapeCOSEUtilities pointer.

```
ICapeCOSEUtilities Ptr capeCOSE;
capeCOSE = m_simulationContext;
If (capeCOSE ==NULL) {
// ICapeCOSEUtilities not supported by the COSE
}
```

- ❑ Step 2: Check that named value is supported and get it

```
CapeArrayString namesList;
CapeVariant namedValue;
CapeLong freeFORTRANchannel;
namesList = capeCOSE.NamedValueList();
if (isContained(namesList, “freeFORTRANchannel”))
       namedValue = capeCOSE.NamedValue(“freeFORTRANchannel”);
       freeFORTRANchannel = namedValue;
}
```

7. Specific Glossary Terms

8. Bibliography

8.1 Process simulation references

- Methods & Tools Integrated Guidelines
- Open Interface Specification: Utilities Common Interface

8.2 Computing references

8.3 General references

9. Appendices