# Physical Properties Guide

**Release v3.5**

**June 2012**

# Physical Properties Guide

Release v3.5
June 2012

Process Systems Enterprise Limited
6th Floor East
26-28 Hammersmith Grove
London W6 7HA
United Kingdom
Tel: +44 20 85630888
Fax: +44 20 85630999
WWW: http://www.psenterprise.com

**Acknowledgements**

ModelBuilder uses the following third party free-software packages. The distribution and use of these libraries is governed by their respective licenses which can be found in full in the distribution. Where required, the source code will made available upon request. Please contact support.gPROMS@psenterprise.com in such a case.

Many thanks to the developers of these great products!

**Table 1. Third party free-software packages**

| **Software/Copyright** | **Website** | **License** |
|---|---|---|
| **ANTLR** | http://www.antlr2.org/ | Public Domain |
| **Batik** | http://xmlgraphics.apache.org/batik/ | Apache v2.0 |
| Copyright © 1999-2007 The Apache Software Foundation. | | |
| **BLAS** | http://www.netlib.org/blas | BSD Style |
| Copyright © 1992-2009 The University of Tennessee. | | |
| **Boost** | http://www.boost.org/ | Boost |
| Copyright © 1999-2007 The Apache Software Foundation. | | |
| **Castor** | http://www.castor.org/ | Apache v2.0 |
| Copyright © 2004-2005 Werner Guttmann | | |
| **Commons CLI** | http://commons.apache.org/cli/ | Apache v2.0 |
| Copyright © 2002-2004 The Apache Software Foundation. | | |
| **Commons Collections** | http://commons.apache.org/collections/ | Apache v2.0 |
| Copyright © 2002-2004 The Apache Software Foundation. | | |
| **Commons Lang** | http://commons.apache.org/lang/ | Apache v2.0 |
| Copyright © 1999-2008 The Apache Software Foundation. | | |
| **Commons Logging** | http://commons.apache.org/logging/ | Apache v1.1 |
| Copyright © 1999-2001 The Apache Software Foundation. | | |
| **Crypto++ (AES/Rijndael and SHA-256)** | http://www.cryptopp.com/ | Public Domain |
| Copyright © 1995-2009 Wei Dai and contributors. | | |
| **Fast MD5** | http://www.twmacinta.com/myjava/fast_md5.php | LGPL v2.1 |
| Copyright © 2002-2005 Timothy W Macinta. | | |
| **HQP** | http://hqp.sourceforge.net/ | LGPL v2 |
| Copyright © 1994-2002 Ruediger Franke. | | |
| **Jakarta Regexp** | http://jakarta.apache.org/regexp/ | Apache v1.1 |
| Copyright © 1999-2002 The Apache Software Foundation. | | |
| **JavaHelp** | http://javahelp.java.net/ | GPL v2 with classpath exception |
| Copyright © 2011, Oracle and/or its affiliates. | | |
| **JXButtonPanel** | http://swinghelper.dev.java.net/ | LGPL v2.1 (or later) |
| Copyright © 2011, Oracle and/or its affiliates. | | |
| **LAPACK** | http://www.netlib.org/lapack/ | BSD Style |
| **libodbc++** | http://libodbcxx.sourceforge.net/ | LGPL v2 |

# Table of Contents

1. Overview ........ 1
2. Using Physical Properties for Simple Materials ........ 2
    - The set of physical properties supported by gPROMS ........ 2
    - Incorporating physical properties in Models ........ 5
    - Using Models incorporating physical property calculations ........ 7
        - Example 1: Explicit specification within a primitive Model ........ 8
        - Example 2: Explicit specification within a higher-level Model ........ 9
        - Example 3: Explicit specification within a Process ........ 9
        - Example 4: Implicit specification via parameter propagation ........ 10
    - The Multiflash physical property interface ........ 11
        - Error handling in the gPROMS/Multiflash interface ........ 11
        - Units of measurement ........ 12
        - Equilibrium flash methods in Multiflash ........ 13
        - Constructing a Multiflash input file ........ 14
        - Using Multiflash stream types ........ 19
    - The IK-CAPE physical property interface ........ 21
    - The CAPE-OPEN physical property interface (COThermoFO) ........ 21
        - Introduction to property packages and systems ........ 22
        - Using COThermoFO in a gPROMS model ........ 22
        - Obtaining a list of available property packages/systems ........ 23
        - Optional flags to COThermoFO ........ 24
        - Overriding the values returned by the COMPONENTS() method ........ 25
        - List of supported methods/properties ........ 26
        - The COThermoFO Test Tool ........ 31
3. Using Physical Properties for Complex Materials ........ 34
    - General concepts for electrolyte system modelling ........ 34
        - Species and phases ........ 34
        - Vapour-liquid equilibrium ........ 35
        - Liquid-phase reactions ........ 35
        - Liquid-solid equilibrium ........ 36
        - Fugacity and activity coefficients ........ 36
    - Electrolytic physical property methods ........ 36
    - The OLI physical property interface ........ 41
        - Electrolyte system report file ........ 42
        - Error handling in the OLI interface ........ 43

# List of Figures

2.1. The Multiflash Window ..... 16
2.2. The Components Dialog ..... 16
2.3. The Select Model Set Dialog ..... 17

# List of Tables

1. Third party free-software packages .......... 3
2.1. Physical property functions and their arguments .......... 3
2.2. Equilibrium flash physical property functions and their arguments .......... 4
2.3. Units of the properties returned by the Multiflash interface .......... 12
2.4. Equilibrium flash physical property functions and their arguments .......... 13
2.5. Results vector organisation for the Multiflash equilibrium flash methods .......... 14
2.6. Methods corresponding to CAPE-OPEN universal constants [4.12.2] .......... 26
2.7. Methods corresponding to CAPE-OPEN constant properties [4.12.1] .......... 26
2.8. Methods corresponding to CAPE-OPEN non-constant properties [4.12.3] .......... 27
2.9. Methods corresponding to CAPE-OPEN flash calculations .......... 28
2.10. Other supported methods .......... 29
2.11. Notes: .......... 29
2.12. .......... 30
2.13. .......... 30
3.1. Thermophysical property methods and their arguments .......... 38
3.2. Details of the vector methods in the interface .......... 39
3.3. Organisation of results vector for methods MolecularTPEquilibrium and TrueSpeciesTPEquilibrium ..... 40

# List of Examples

2.1. Example COThermoFOTestTool XML file ..... 32

# Chapter 1. Overview

Physical Properties interfaces to gPROMS: Physical property packages form a special class of Foreign Objects (see above) that are encountered very frequently in practice. A number of widely-used physical property packages are already interfaced to gPROMS. Simple materials describes how to incorporate physical property calculations for 'conventional' materials in gPROMS models using the interfaces to Multiflash and IK-Cape or a CAPE-OPEN compliant physical properties package, while Complex materials looks at more complex, electrolytic systems using the OLI package.

A good understanding of gPROMS at an introductory level is essential for making the most of the facilities described in this guide. The gPROMS Model Developer Guide and/or the gPROMS Introductory Training Course provide the necessary background. With the exception of the chapters on physical properties interfaces, the chapters in this manual can be read independently of each other. For these chapters, understanding of the concepts presented in the chapter on the foreign object interface is a necessary prerequisite.

# Chapter 2. Using Physical Properties for Simple Materials

Most non-trivial process models make use of physical properties, such as density, enthalpy and fugacity, all of which are typically functions of temperature, pressure and composition. The accuracy of estimation of these physical properties is one of the most critical factors in determining the accuracy of the model predictions.

In principle, physical property calculations may be coded as Equations within gPROMS Models. However, this is not practical for all but the simplest methods of computing physical properties. It may also be unnecessary in view of the wide range of general and reliable physical property calculation packages that are currently available.

This chapter describes how you can incorporate physical properties within your Models. The actual computation of these physical properties is performed by external physical property software that is interfaced to gPROMS.

gPROMS defines a minimal set of physical properties that any external package interfaced to it should provide, as well as the precise form of these properties. These are listed in: The set of physical properties supported by gPROMS. We strongly recommend that you use these and, if possible, *only* these physical properties in any Models that you write. In this way, you will be able to use your Models with any physical property package that adheres to the standard gPROMS protocol.

Incorporating physical properties in Models describes how precisely you can refer to the above physical properties in any Models that you write. This is done by treating the external physical property software as a Foreign Object interfaced to gPROMS and providing services to it. You will find that a basic understanding of Foreign Object concepts is very helpful in making the most out of incorporating physical properties in Models.

Then a description is given on how one can use Models that involve physical property calculations. The Models can be written by yourself or by others.

In addition to defining a standard protocol for interacting with physical property packages, interfaces to two such packages are already available to be licensed and used with gPROMS. The special characteristics of these interfaces are described in: Multiflash physical property interface and IK-CAPE physical property interface.

gPROMS supports the CAPE-OPEN thermodynamics standard. This allows the use of external physical properties software complying to this standard within gPROMS Model entities.

This chapter is concerned with physical property calculations of simple mixtures of molecular species existing in liquid and vapour phases. Physical properties for more complex materials (such as electrolytic systems) are considered in the next chapter on using physical properties for complex materials. However, we recommend that you read and understand the present chapter *before* proceeding to the next chapter.

If you are interested in interfacing a different physical property package to gPROMS, then you need to read the chapter entitled "Interfacing Physical Property Packages to gPROMS" in the "gPROMS System Programmer Guide" *after* you have read and thoroughly understood this chapter.

## The set of physical properties supported by gPROMS

External physical property packages are interfaced to gPROMS via the latter's Physical Property Interface (PPI). The PPI recommends that a *minimal* set of physical property calculations are supported by any such external package. These are listed in the tables below and include most of the common thermophysical and transport property calculations used in the chemical industry.

### Caution

Always consult the documentation provided with the physical property interface that you intend to use.

The physical properties listed in the tables below represent a minimal set of properties that physical property packages interfaced to gPROMS are recommended to support.

There is no guarantee that all (or any!) of these properties are actually supported by any particular interface. Also, some interfaces may provide access to properties other than those listed in the tables below.

The units of measurement of the physical properties and the various quantities needed for their calculation may also vary from package to package.

There a few points that you need to note about the first table:

- *Scalar* methods (like LiquidEnthalpy) return a single value, typically corresponding to a property of the mixture.
- *Vector* methods (like MolecularWeight) return an array of results, each element of which corresponds to a different component in the mixture.
- Most methods receive three arguments, namely *T*, *P* and *n*.
  - *T* is a scalar quantity that corresponds to the temperature and should be in Kelvin.
  - *P* is a scalar quantity that corresponds to the pressure and should be in Pascal.
  - *n* is an array-valued expression that corresponds to the number of moles of the components in the mixture.
- The enthalpy, entropy, internal energy and volume calculations return *total* quantities. However, the corresponding *specific* quantities can be obtained by normalising the mole numbers n, i.e. passing molar fractions as arguments to the various methods.
- SurfaceTension is a special case: $n_L$ and $n_V$ are the number of moles of, respectively, the liquid and vapour phases in contact with each other.
- For the vapour phase properties, the spelling Vapor*XXXX* is also recognised as an alternative to Vapour*XXXX*.

The methods presented in the second table require an equilibrium flash calculation to be performed. The following points need to be noted:

- The number of moles n in the inputs list refers to the overall composition of the mixture.
- Property methods like Enthalpy and Density return the values for the whole system (and not for a specific phase as is the case for the methods listed in the first table).
- The content and size of the results array returned by the vector methods will depend on the particular physical property package that is being used. You will need to refer to the appropriate documentation (see also: Multiflash physical property interface for one particular example).
- The method IsLiquidStable returns a Boolean result which is "true" if the phase is liquid and stable, otherwise the result is "false".
- The method IsVapourStable returns a Boolean result which is "true" if the phase is vapour and stable, otherwise the result is "false".

**Table 2.1. Physical property functions and their arguments**

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| NormalBoilingPoint | | Normal boiling point | Vector |
| CriticalTemperature | | Critical temperature | Vector |
| CriticalPressure | | Critical pressure | Vector |
| CriticalVolume | | Critical volume | Vector |
| NormalFreezingPoint | | Melting point | Vector |
| MolecularWeight | | Molecular weight | Vector |

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| IdealGasEnthalpyOfFormationAt25C | | Enthalpy of formation | Vector |
| IdealGasGibbsFreeEnergyOFFormationAt25C | | Gibbs free energy of formation | Vector |
| Components | | The names of all components | Scalar |
| NumberOfComponents | | Number of components | Scalar |
| VapourPressure | $T$ | Pure component vapour pressures | Vector |
| LiquidCpCv | $T,P,n$ | Ratio of $C_P$ to $C_V$ in the liquid phase | Scalar |
| VapourCpCv | $T,P,n$ | Ratio of $C_P$ to $C_V$ in the vapour phase | Scalar |
| LiquidCompressibilityFactor | $T,P,n$ | Compressibility factor for the liquid phase | Scalar |
| VapourCompressibilityFactor | $T,P,n$ | Compressibility factor for the vapour phase | Scalar |
| LiquidEnthalpy | $T,P,n$ | Total enthalpy of the liquid phase | Scalar |
| LiquidPartialEnthalpy | $T,P,n$ | Partial enthalpy of the liquid phase | Vector |
| VapourEnthalpy | $T,P,n$ | Total enthalpy of the vapour phase | Scalar |
| VapourPartialEnthalpy | $T,P,n$ | Partial enthalpy of the vapour phase | Vector |
| LiquidExcessEnthalpy | $T,P,n$ | Excess enthalpy of the mixture | Scalar |
| LiquidEntropy | $T,P,n$ | Total entropy of the liquid phase | Scalar |
| VapourEntropy | $T,P,n$ | Total entropy of the vapour phase | Scalar |
| LiquidFugacityCoefficient | $T,P,n$ | Liquid fugacity coefficients | Vector |
| VapourFugacityCoefficient | $T,P,n$ | Vapour fugacity coefficients | Vector |
| LiquidActivityCoefficient | $T,P,n$ | Liquid activity coefficients | Vector |
| LiquidGibbsFreeEnergy | $T,P,n$ | Total Gibbs energy of the liquid phase | Scalar |
| VapourGibbsFreeEnergy | $T,P,n$ | Total Gibbs energy of the vapour phase | Scalar |
| LiquidExcessGibbsFreeEnergy | $T,P,n$ | Excess Gibbs energy of the mixture | Scalar |
| LiquidHeatCapacity | $T,P,n$ | Liquid heat capacity at constant pressure | Scalar |
| VapourHeatCapacity | $T,P,n$ | Vapour heat capacity at constant pressure | Scalar |
| LiquidEnergy | $T,P,n$ | Total internal energy of the liquid phase | Scalar |
| VapourEnergy | $T,P,n$ | Total internal energy of the vapour phase | Scalar |
| LiquidVolume | $T,P,n$ | Total liquid volume | Scalar |
| VapourVolume | $T,P,n$ | Total vapour volume | Scalar |
| LiquidDensity | $T,P,n$ | Density of the liquid phase | Scalar |
| VapourDensity | $T,P,n$ | Density of the vapour phase | Scalar |
| LiquidThermalConductivity | $T,P,n$ | Thermal conductivity of the liquid phase | Scalar |
| VapourThermalConductivity | $T,P,n$ | Thermal conductivity of the vapour phase | Scalar |
| LiquidViscosity | $T,P,n$ | Viscosity of the liquid phase | Scalar |
| VapourViscosity | $T,P,n$ | Viscosity of the vapour phase | Scalar |
| SurfaceTension | $T,P,n_L,n_V$ | Surface tension of the mixture | Scalar |

## Table 2.2. Equilibrium flash physical property functions and their arguments

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| TPFlash | $T,P,n$ | Equilibrium temperature and pressure flash | Vector |
| TVFlash | $T,P,n$ | Equilibrium temperature and volume flash | Vector |
| PHFlash | $P,H,n$ | Equilibrium pressure and enthalpy flash | Vector |

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| PHTemperature | *P*,*H*,*n* | The equilibrium temperature of a pressure and enthalpy flash | Scalar |
| UVFlash | *U*,*V*,*n* | Equilibrium internal energy and volume flash | Vector |
| Enthalpy | *T*,*P*,*n* | Total enthalpy of the mixture | Scalar |
| PartialEnthalpy | *T*,*P*,*n* | Partial enthalpy of the mixture | Vector |
| Entropy | *T*,*P*,*n* | Total entropy of the mixture | Scalar |
| Volume | *T*,*P*,*n* | Total volume of the mixture | Scalar |
| Density | *T*,*P*,*n* | Total density of the mixture | Scalar |
| CompressibilityFactor | *T*,*P*,*n* | Total compressibility factor of the mixture | Scalar |
| Energy | *T*,*P*,*n* | Total internal energy of the mixture | Scalar |
| HeatCapacity | *T*,*P*,*n* | Total heat capacity of the mixture | Scalar |
| GibbsFreeEnergy | *T*,*P*,*n* | Total Gibbs energy of the mixture | Scalar |
| BubbleTemperature | *P*,*n* | Bubble point temperature of the mixture | Scalar |
| BubbleTemperatureCompositions | *P*,*n* | Composition of the vapour phase | Vector |
| BubblePressure | *T*,*n* | Bubble point pressure of the mixture | Scalar |
| DewTemperature | *P*,*n* | Dew point temperature of the mixture | Scalar |
| DewTemperatureCompositions | *P*,*n* | Composition of the liquid phase | Vector |
| DewPressure | *T*,*n* | Dew point pressure of the mixture | Scalar |
| IsLiquidStable | *T*,*P*,*n* | Stability indicator for the liquid phase | Scalar |
| IsVapourStable | *T*,*P*,*n* | Stability indicator for the vapour phase | Scalar |
| VapourPhaseFraction | *T*,*P*,*n* | Vapour phase fraction of the mixture | Scalar |

# Incorporating physical properties in Models

External physical property packages are interfaced to gPROMS as Foreign Objects. The usage of physical property Foreign Objects in Models is governed by a set of simple rules:

A. Each distinct Physical Property Foreign Object used by a Model is declared as a Parameter of type FOREIGN_OBJECT. The latter keyword is normally followed by the *class* of the Foreign Object (see also: Classes and instances of Foreign Objects) which identifies the external physical properties software that will be used to implement this instance of the Foreign Object. For example, a typical declaration would be:

```
PARAMETER PPP AS FOREIGN_OBJECT "Multiflash"
```

This simply states that:

- Model Flash will make use of a Physical Property Foreign Object.
- Whenever necessary within this Model (see below), the Foreign Object will be referred to as PPP.
- PPP is implemented by a piece of external software (in this case, a physical property package) called Multiflash. Thus, PPP is an "instance" of Multiflash.

One important thing to note is that the class of the Foreign Object must be enclosed in quotes ("Multiflash"). This is because it is neither a gPROMS keyword nor an identifier that has previously been defined in the gPROMS input file.

B. The standard way of referring to physical property calculations within the Model is as follows:

- for constant properties (mainly those relating to pure components properties such as molecular weights):

```
ForeignObjectName.PhysicalProperty
```

- for variable physical properties:

```
ForeignObjectName.PhysicalProperty(InputList)
```

PhysicalProperty is one of those listed in the first column of the tables in: The set of physical properties supported by gPROMS while the InputList list is as given in the second column.

C. Each input of a physical property is a scalar or vector-valued variable or expression.

D. Each property returns a single scalar or vector-valued quantity. The latter may be of type integer, logical or real. A method may be used anywhere in the Model where an expression of the corresponding dimensionality and type is allowed.

An example of a Model making use of a physical properties package is shown in the listing below.[1]

```
 1 # model entity "Flash"

 2  PARAMETER
 3    PPP     AS                  FOREIGN_OBJECT "Multiflash"
 4    NoComp  AS                  INTEGER
 5    Mw      AS ARRAY(NoComp) OF REAL

 6  VARIABLE
 7    F, L, V AS                  MolarRate
 8    Hf      AS                  MolarEnergy
 9    Q       AS                  EnergyRate
10    T       AS                  Temperature
11    P       AS                  Pressure
12    x, y, z AS ARRAY(NoComp) OF MoleFraction

13  SET
14    NoComp := PPP.NumberOfComponents ;
15    Mw     := PPP.MolecularWeight ;

16  EQUATION
17    F*z = L*x + V*y ;
18    F*Hf = PPP.LiquidEnthalpy(T,P,L*x) + PPP.VapourEnthalpy(T,P,V*y) + Q ;
19    x*PPP.LiquidFugacityCoefficient(T,P,x) =
20                                 y*PPP.VapourFugacityCoefficient(T,P,y) ;
21    SIGMA(x) = SIGMA(y) = 1 ;
22    ...
```

As we have already seen, line 3 specifies that the Model Flash makes use of a Foreign Object of type Multiflash, corresponding to a software package that is external to gPROMS. For the purposes of defining this Model, this Foreign Object will be called PPP.

Physical property methods are used in several places within the Model. For example:

- At line 14, we make use of the scalar integer method NumberOfComponents to Set the value of the integer parameter NoComp.
- At line 15, we make use of the vector real method MolecularWeight to Set the value of the real array parameter Mw.
- At line 18, we invoke the methods LiquidEnthalpy and VapourEnthalpy to compute the (total) enthalpies of the liquid and vapour product streams respectively.

[1]Note that the line numbers shown in the figures have been included for ease of reference in this document and are not part of the gPROMS language.

- At line 19 and 20, we invoke two more methods, namely LiquidFugacityCoefficient and VapourFugacityCoefficient respectively to compute *arrays* of liquid and vapour phase fugacity coefficients.

It is worth emphasising that method inputs can be expressions rather than simple variables. One example has already been seen at line 18 of the above flash model. A second example is shown below where a liquid phase viscosity is evaluated at the mean of the inlet and exit conditions of a particular unit:

```
PPP.LiquidViscosity((Tin+T)/2, (Pin+P)/2, (xin+x)/2)
```

Finally, no direct reference can be made to an element of a vector-valued method. For instance, it is not possible to access directly the $i^{th}$ element of the vector of liquid fugacity coefficients returned by method LiquidFugacityCoefficient (see line 17 of the listing above). If such access is desired, an equation defining a vector-valued variable as being equal to the method result must first be introduced, e.g.:

```
LFC = PPP.LiquidFugacityCoefficient(T,P,x) ;
```

Other equations can then refer to individual elements of the variable LFC as desired.

**CAUTION!**

It is the model developer's responsibility to check the type (including the correct units of the inputs) and the order of the arguments for each physical property invocation. The current version of gPROMS will perform only a minimal check on these.

## Using Models incorporating physical property calculations

We have seen how physical property calculations can be incorporated within Models, being provided by external physical property packages. However, before these Models can be used to perform, say, a simulation of a given process, some additional information relating to physical properties needs to be specified.

Consider, for example, the Foreign Object PPP used in the example.

```
 1 # model entity "Flash"

 2  PARAMETER
 3    PPP     AS                  FOREIGN_OBJECT "Multiflash"
 4    NoComp  AS                  INTEGER
 5    Mw      AS ARRAY(NoComp) OF REAL

 6  VARIABLE
 7    F, L, V AS                  MolarRate
 8    Hf      AS                  MolarEnergy
 9    Q       AS                  EnergyRate
10    T       AS                  Temperature
11    P       AS                  Pressure
12    x, y, z AS ARRAY(NoComp) OF MoleFraction

13  SET
14    NoComp := PPP.NumberOfComponents ;
15    Mw     := PPP.MolecularWeight ;

16  EQUATION
17    F*z = L*x + V*y ;
18    F*Hf = PPP.LiquidEnthalpy(T,P,L*x) + PPP.VapourEnthalpy(T,P,V*y) + Q ;
19    x*PPP.LiquidFugacityCoefficient(T,P,x) =
20                                 y*PPP.VapourFugacityCoefficient(T,P,y) ;
21    SIGMA(x) = SIGMA(y) = 1 ;
```

```
22    ...
```

We already know that the methods (e.g. LiquidFugacityCoeffient) of this Foreign Object will be provided by an external physical property package called "Multiflash". However, for these methods to be fully defined, we also need to know the set of components that are involved in the mixture under consideration and also the particular type of thermophysical model (e.g. the equation of state or activity coefficient model) that will be used to compute these properties. All of this information is associated with the specific Foreign Object instance PPP. Indeed, two different instances of the same Foreign Object class within the same gPROMS simulation may involve different component sets and/or use different thermophysical property calculation options.

We note that the above information is not actually directly used by gPROMS as it relies entirely on computations provided to it by the external physical property package. In fact, most modern physical property packages provide their own mechanism for the specification of the above information. Typically, this takes the form of a text file conforming to a proprietary format. The file can be constructed and edited using either a text editor or a graphical user interface provided by the package.

As we have seen in: Foreign Object values and their specification, the value of an instance of a Foreign Object is a string of characters that gPROMS passes to the external software to allow it to create this instance. In the case of physical property Foreign Objects, this string typically corresponds to the name of the proprietary file that contains the information mentioned above.

The value of an instance Foreign Object is specified in the Set section of either a Model or a Process entity. Here, we provide some examples of such specifications.

## Example 1: Explicit specification within a primitive Model

Consider the flash model example.

```
 1 # model entity "Flash"

 2  PARAMETER
 3    PPP     AS                  FOREIGN_OBJECT "Multiflash"
 4    NoComp  AS                  INTEGER
 5    Mw      AS ARRAY(NoComp) OF REAL

 6  VARIABLE
 7    F, L, V AS                  MolarRate
 8    Hf      AS                  MolarEnergy
 9    Q       AS                  EnergyRate
10    T       AS                  Temperature
11    P       AS                  Pressure
12    x, y, z AS ARRAY(NoComp) OF MoleFraction

13  SET
14    NoComp := PPP.NumberOfComponents ;
15    Mw     := PPP.MolecularWeight ;

16  EQUATION
17    F*z = L*x + V*y ;
18    F*Hf = PPP.LiquidEnthalpy(T,P,L*x) + PPP.VapourEnthalpy(T,P,V*y) + Q ;
19    x*PPP.LiquidFugacityCoefficient(T,P,x) =
20                                   y*PPP.VapourFugacityCoefficient(T,P,y) ;
21    SIGMA(x) = SIGMA(y) = 1 ;
22    ...
```

In this case, the value of the Foreign Object parameter PPP could be specified by inserting the following line in its Set section (say, after line 13): of the Flash Model :

```
PPP := "Aromatics.mfl" ;
```

This states that the behaviour of Foreign Object PPP is defined by a Multiflash input file (.mfl) called Aromatics.mfl. For this to occur, the file must either be imported into the project file or linked, see the gPROMS ModelDeveloper User Guide. Alternatively, the absolute path to the file can be given.

The obvious disadvantage of the above way of specifying the Foreign Object value is that all instances of Model Flash will be restricted to using the same set of components and thermophysical property calculation options. Thus, the generality and reusability of this Model is severely compromised!

## Example 2: Explicit specification within a higher-level Model

Consider Setting Foreign Object values within composite Models, for example, the Model TwoTankSystem illustrated in the listing below:

```
 1 # model entity "TwoTankSystem"

 2   PARAMETER
 3     LPPP AS FOREIGN_OBJECT "Multiflash"

 4   VARIABLE
 5     TotalVolFlow    AS VolumeRate

 6   UNIT
 7     HighPTank AS Flash
 8     LowPTank  AS Flash

 9   SET
10     HighPTank.PPP := "HPThermo.mfl" ;
11     LowPTank.PPP  := LPPP ;

12   EQUATION
13     TotalVol =
14      HighPTank.PPP.LiquidVolume
15          (HighPTank.T, HighPTank.P, HighPTank.L*HighPTank.x)
16      LowPTank.PPP.LiquidVolume
17          (LowPTank.T,  LowPTank.P,  LowPTank.L*LowPTank.x) ;
```

It contains two instances, HighPTank and LowPTank respectively, of the Model Flash shown in: Explicit specification within a primitive Model. These two flashes operate at significantly different pressures, and consequently make use of different thermophysical property calculation options. On line 10, the Foreign Object PPP of the first of these two flashes is identified with a Multiflash input file HPThermo.mfl in the gPROMS export directory. On the other hand, line 11 Sets the corresponding Foreign Object of the second flash to be equal to a Parameter LPPP declared within TwoTankSystem (see line 3). Obviously, LPPP will have to be Set in an even higher-level Model or in a Process (see below).

As is standard in gPROMS, a composite Model entity can access any Parameter or Variable declared within instances of lower-level Model entities that it refers to. This rule also holds for physical properties Foreign Object entities. For example, lines 13-17 of the above listing compute the combined volumetric flowrate of the liquid streams leaving the two tanks by making use of their individual physical property calculations.

## Example 3: Explicit specification within a Process

Setting a Physical Properties Foreign Object value within a Process is illustrated in the listing below. Lines 2 and 3 declare an instance called Plant of the composite Model defined in the listing in: Explicit specification within a higher-level Model. The value of the Foreign Object parameter LPPP occurring in that Model is then Set in lines 4 and 5.

```
# process entity "PlantSimulation"

  UNIT
    Plant AS TwoTankSystem

  SET
    Plant.LPPP := "LPThermo.mfl" ;

  ...
```

## Example 4: Implicit specification via parameter propagation

Like other gPROMS Parameters, the values of Physical Properties Foreign Objects may be specified implicitly via the gPROMS parameter propagation mechanism.

Parameter propagation is invoked automatically by gPROMS at the start of the execution of a Process if the value of any Foreign Object instance within the entire problem has not been specified explicitly. In such cases, gPROMS will try to determine the missing value by searching for a Foreign Object that:

- is declared within a higher-level Model containing the instance of the Model which has the unspecified Foreign Object, **and**
- has exactly the same name as the unspecified Foreign Object, **and**
- belongs to the same class as the unspecified Foreign Object, **and**
- has already been given a value either explicitly or implicitly.

If a Foreign Object fulfilling *all* of the above criteria is found, its value is also given to the unspecified Foreign Object. If no such Foreign Object is found, the algorithm is applied recursively, searching increasingly higher-level Models and ultimately the Process itself. If, by the end of this procedure, the value of the unspecified Foreign Object remains undetermined, an error occurs and the execution of the Process is aborted.

The main practical implication of the parameter propagation mechanism is that, provided the model developer adopts a standard name (say PPP) for all Physical Properties Foreign Object instances of a certain class (e.g. Multiflash) in all Models that he or she develops, then it is sufficient to specify the value of this Foreign Object once only (usually in the Process section) for it to be adopted automatically by the entire problem.

Alternatively, if a plant has two distinct sections (e.g. a high temperature and a low temperature one) which require the use of two different Foreign Objects, again this can be specified conveniently at the highest appropriate level, as illustrated in the listing below. Here, it is assumed that the Models HighTemperatureSection and LowTemperatureSection referred to in lines 3-4 each contain a Foreign Object called PPP and so do all of their sub-models. In this case, simply Setting appropriate values for the two highest-level Foreign Objects in the Process section (see lines 5-7) achieves the desired specification for the entire problem.

```
1 # model entity "Plant"

2   UNIT
3     HTS AS HighTemperatureSection
4     LTS AS LowTemperatureSection

5   SET
6     HTS.PPP := "HighTThermo.mfl" ;
7     LTS.PPP := "LowTThermo.mfl" ;

8   EQUATION
9     ...
```

## The Multiflash™ physical property interface

Multiflash is a physical property package developed and marketed by Infochem Computer Services Ltd.[2] A gPROMS interface for Multiflash providing the functionality described in the first section in this chapter on the physical properties supported by gPROMS is available and can be licensed together with gPROMS.

To use Multiflash, you need to go through the following steps:

1. In your gPROMS project, ensure that you state the class of your Foreign Objects to be Multiflash (see lines 2 and 3 of the listing in: Explicit specification within a primitive Model).

2. Create a Multiflash input file (.mfl) defining all the components, physical property models, databanks etc. that are to be used by your problem (see also: Constructing a Multiflash input file for more details).

3. In your gPROMS project, Set the value of your Foreign Object to the name of this command file. For example, if the name of the command file is AlkaneMixture.mfl and the Foreign Object is called PhysProp, then the corresponding Set statement in the Model or Process will be:

   ```
   SET
       PhysProp := "AlkaneMixture.mfl" ;
   ```

4. Import the Multiflash command file (e.g. AlkaneMixture.mfl) into the gPROMS Project.

5. Any of the methods listed in the two tables in: Set of Physical Properties supported by gPROMS can now be used in your gPROMS project[3]

## Error handling in the gPROMS/Multiflash interface

Errors may occur at two different two stages when using Multiflash in gPROMS:

- *During the creation of the Multiflash instance at the start of the simulation*

  This type of failure will almost always be caused by a mistake (e.g. syntax error) in the Multiflash input file. Any error at this stage will lead to an immediate termination of the simulation. Specific attention should be given to the syntax of component names according to the specifications described in the Multiflash manual.

  At least one message and an error number will be reported on the screen and a line-by-line analysis of the input file(s) will be dumped in the file MultiflashCommandFileErrors.txt residing in the Results sub-directory of the Case. Please view this file to detect the source of the errors. In some cases, you may need to consult the Multiflash manual (using the number of the error reported) for more information. Note that the error message:

  ```
  No information available under this error number
  ```

  very often indicates that the configfile (i.e. the file which tells Multiflash where to locate its databanks and error files) could not be found. In such cases, you should check the path specification for the configfile as described in the Multiflash manual.

  If the creation of the Multiflash instance is successful, a message will appear on the screen to confirm this. gPROMS will then proceed with the numerical solution.

- *During the numerical solution*

  Errors reported by Multiflash as a result of calls issued to it by gPROMS during the numerical solution phase are captured by the interface. If a *fatal* error occurs:

  - A message will be printed on the screen pointing out:

[2]See http://www.infochemuk.com.
[3]with the exception of the LiquidExcessEnthalpy which is not currently supported.

  - the particular method that was being called (e.g. the enthalpy calculation routine);
  - the values of the inputs (e.g. temperature, pressure, composition) of this method;
  - the number used by Multiflash to identify this type of error; use this to consult the Multiflash manual for more details.

- An error file will automatically be generated. The name of the error file will be xxx-multiflash.error, where xxx is the name of the .mfl file, and it will be located in the Results directory of the Case. For instance, if the name of the error file is AlkaneMixture.mfl, then the name of the error file will be AlkaneMixture-multiflash.error. The error file will contain details of the input and output values of the Multiflash routine called as well as the full listing of the command file. A screen message also appears confirming the creation of the error file. The error file should be included in all correspondence with support.gPROMS@psenterprise.com.

Any non-fatal errors reported by Multiflash are ignored. For example, during a particular iteration in gPROMS, the values of inputs may temporary go outside the recommended bounds of a correlation or the simulation may enter supercritical regions. The numerical solution continues and, when appropriate, screen messages are printed.

## Units of measurement

"Units of measurements" refers to the quantities listed in the two tables in: Set of Physical Properties supported by gPROMS. The units for the input quantities are:

- temperature *T*: Kelvin
- pressure *P*: Pa
- amount of substance, *n*: mol. This is the default unit. The amount of substance can alternatively be expressed on a mass basis, in kg. To indicate this, the keyword, Mass, should be used when Setting the value of the Foreign Object, as follows:

  ```
  SET
    PhysProp1 := "AlkaneMixture.mfl" ;      #molar basis
    PhysProp2 := "MASS:AlkaneMixture.mfl" ; #mass basis
  ```

  Note that it is permissible to have both types of specifications in the same gPROMS Model or Process. These will correspond to two different Foreign Objects.

- total enthalpy of mixture, *H*: Joule
- total volume of the mixture, *V*: $m^3$
- total internal energy, *U*: Joule

The units of the returned quantities are given in the table below:

**Table 2.3. Units of the properties returned by the Multiflash interface**

| Property | Units of measurement |
|---|---|
| ActivityCoefficients | - |
| BoilingPoint | K |
| BubbleTemperature | K |
| BubblePressure | Pa |
| Conductivity | W/(m.K) |
| CompressibilityFactor | - |

| Property | Units of measurement |
|---|---|
| CpCv | - |
| CriticalPressure | Pa |
| CriticalTemperature | K |
| CriticalVolume | $m^3$/mol |
| Enthalpy | J |
| Density | kg/$m^3$ |
| DewTemperature | K |
| DewPressure | Pa |
| Entropy | J/K |
| ExcessGibbsEnergy | J |
| FugacityCoefficients | - |
| GibbsEnergy | J |
| HeatCapacity | J/K |
| IdealGasEnthalpyOfFormationAt25C | J/mol (molar basis) or J/kg (mass basis) |
| InternalEnergy | J |
| MeltingPoint | K |
| MolecularWeight | g/mol |
| NumberOfComponents | - |
| SurfaceTension | N/m |
| VapourPhaseFraction | - |
| VapourPressure | Pa |
| Viscosity | Pa.s |
| Volume | $m^3$ |

## Equilibrium flash methods in Multiflash

All the flash methods listed in the second table in: Set of Physical Properties supported by gPROMS (repeated below) are made available by the Multiflash interface.

### Table 2.4. Equilibrium flash physical property functions and their arguments

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| TPFlash | *T,P,n* | Equilibrium temperature and pressure flash | Vector |
| TVFlash | *T,P,n* | Equilibrium temperature and volume flash | Vector |
| PHFlash | *P,H,n* | Equilibrium pressure and enthalpy flash | Vector |
| UVFlash | *U,V,n* | Equilibrium internal energy and volume flash | Vector |
| Enthalpy | *T,P,n* | Total enthalpy of the mixture | Scalar |
| Entropy | *T,P,n* | Total entropy of the mixture | Scalar |
| Volume | *T,P,n* | Total volume of the mixture | Scalar |
| Density | *T,P,n* | Total density of the mixture | Scalar |
| CompressibilityFactor | *T,P,n* | Total compressibility factor of the mixture | Scalar |
| Energy | *T,P,n* | Total internal energy of the mixture | Scalar |
| HeatCapacity | *T,P,n* | Total heat capacity of the mixture | Scalar |

| Property Name | Inputs | Description | Type |
|---|---|---|---|
| GibbsFreeEnergy | *T*,*P*,*n* | Total Gibbs energy of the mixture | Scalar |
| BubbleTemperature | *P*,*n* | Bubble point temperature of the mixture | Scalar |
| BubblePressure | *T*,*n* | Bubble point pressure of the mixture | Scalar |
| DewTemperature | *P*,*n* | Dew point temperature of the mixture | Scalar |
| DewPressure | *T*,*n* | Dew point pressure of the mixture | Scalar |
| IsLiquidStable | *T*,*P*,*n* | Stability indicator for the liquid phase | Scalar |
| IsVapourStable | *T*,*P*,*n* | Stability indicator for the vapour phase | Scalar |
| VapourPhaseFraction | *T*,*P*,*n* | Vapour phase fraction of the mixture | Scalar |

The results vector for the first four methods listed in the table above is of length 11 + 3 * NoComp. The information contained in it is organised as detailed in the table below.

An important point to note is that up to two liquid phases are assumed to be present at equilibrium. Consequently, the flash methods and results can handle a VLLE system. If a particular phase does not exist under the specified conditions, then its amount and all its intensive properties are returned as zero.

**Table 2.5. Results vector organisation for the Multiflash equilibrium flash methods**

| Variable | Length | Position in Results vector | |
|---|---|---|---|
| | | From | To |
| Temperature | 1 | 1 | 1 |
| Pressure | 1 | 2 | 2 |
| Vapour molar fractions | NoComp | 3 | 2+NoComp |
| Vapour enthalpy | 1 | 3+NoComp | 3+NoComp |
| Vapour molar amount | 1 | 4+NoComp | 4+NoComp |
| Vapour volume | 1 | 5+NoComp | 5+NoComp |
| 1st liquid molar fractions | NoComp | 6+NoComp | 5+2*NoComp |
| 1st liquid enthalpy | 1 | 6+2*NoComp | 6+2*NoComp |
| 1st liquid molar amount | 1 | 7+2*NoComp | 7+2*NoComp |
| 1st liquid volume | 1 | 8+2*NoComp | 8+2*NoComp |
| 2nd Liquid molar fractions | NoComp | 9+2*NoComp | 8+3*NoComp |
| 2nd Liquid enthalpy | 1 | 9+3*NoComp | 9+3*NoComp |
| 2nd Liquid molar amount | 1 | 10+3*NoComp | 10+3*NoComp |
| 2nd Liquid volume | 1 | 11+3*NoComp | 11+3*NoComp |

## Constructing a Multiflash input file

The Multiflash input file is a text file containing the specification of the thermodynamic models, data sources for pure components, binary interaction parameters and components that occur in the mixture of interest[4]

By convention, Multiflash input files have the extension .mfl.

[4]Although Multiflash input files may also contain the mixture conditions (e.g. temperature, pressure and composition), this is not necessary (or meaningful) for the use of these files within the context of gPROMS.

There are two ways of constructing Multiflash input files:

- Define the information using the graphical interface of Multiflash for Windows[5] and then "export" this information to create the input file automatically, which can then be imported into ModelBuilder. This is the preffered method of constructing the input file.

- Alternatively, create a Miscellaneous file in ModelBuilder and use the built-in text editor to enter all the required information. (Alternatively any text editor can be used, with the file imported into ModelBuilder).

Full information on the use of the Multiflash for Windows package is provided with the Multiflash user manual which is part of the gPROMS distribution.

## Creating the Multiflash input file using the graphical interface

The Multiflash input file can be created quite easily using Multiflash for Windows. The following procedure is all that is required to generate a working Multiflash input file and include it in a gPROMS project.

1. Launch Multiflash for Windows by left clicking on the Start Menu and selecting Programs → Process Systems Enterprise → Multiflash 3.9 for Windows (assuming the default Start Menu shortcuts were installed).

   - The Multiflash window should now appear (see The Multiflash Window).

   - If you would like to follow the procedure in the Multiflash documentation, then select Programs → Process Systems Enterprise → Documentation → Multiflash → Multiflash for Windows from the Start Menu. Navigate to page 9 (which is the 19th page of the pdf document).

2. Define all of the components in the problem by clicking on the button or by selecting Components... from the Select menu.

   a. Select the required databank from the Data source listbox (see Components Dialog).

   b. Ensure the Name radio button is enabled.

   c. Type in the name of a component in the Enter name: textbox and press return.

   d. Repeat step 3 until all components have been added.

   e. The components are shown on the left hand side and can be changed or removed using the Edit and Delete buttons below.

   f. Press the Close button when all components have been added.

   - See page 12 of the Multiflash documentation for further details.

3. Define the physical property models to use by selecting Model set... from the Select menu.

   a. When the Select Model Set dialog appears, choose the required models and press the Define Model button.

   b. Press OK when the dialog appears indicating that the model was successfully created.

   - This is described on page 14 of the Multiflash documentation and detailed information about the models available can be found on page 27.

4. Save the Multiflash input file by selecting Save Problem Setup (or pressing **CTRL+s**) or Save Problem Setup As... from the File menu, or by pressing the save button: .

   - Choose a suitable location and name for the file. Exactly where it is saved is not important because it will be imported into the gPROMS project in the next step.

[5]This is a Microsoft Windows-based program, the use of which is covered by your Multiflash license.

- This step is described on page 16 of the Multiflash documentation. The steps that we have skipped are not important because the gPROMS activities performed will determine the values of the inputs to Multiflash.

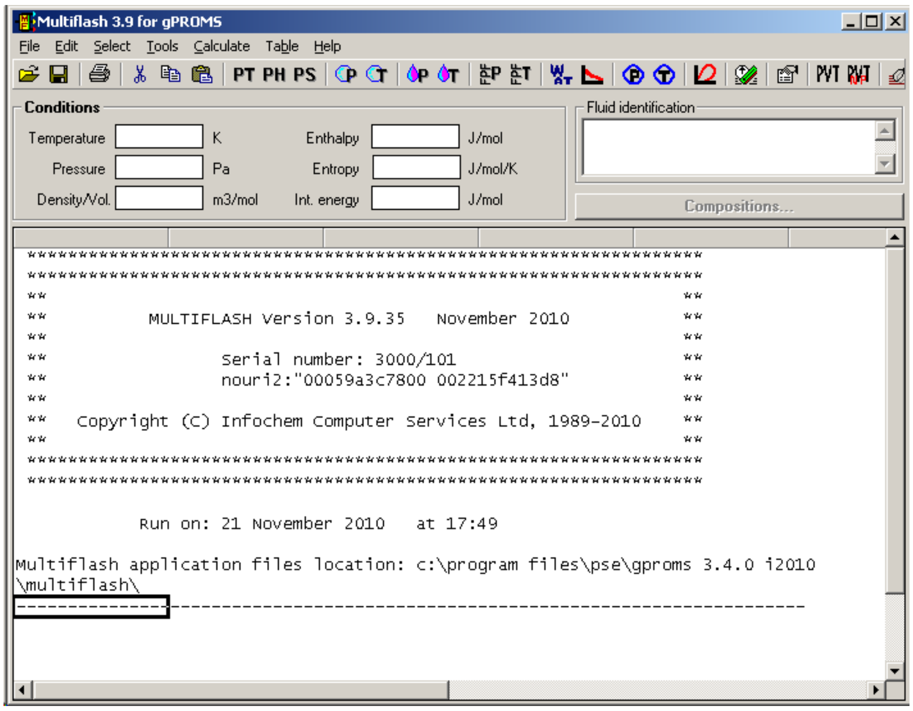
5. Finally, import the Multiflash input file to the gPROMS project.

   a. In ModelBuilder select Import files... from the Tools menu.

   b. Use the file browser to locate the Multiflash file that you saved in the previous step and press the Import button.

   c. The Multiflash input file will now appear in the Miscellaneous Files section of the project tree and can be used in any gPROMS Models by following the procedures described in Incorporating physical properties in Models.

**Figure 2.1. The Multiflash Window**

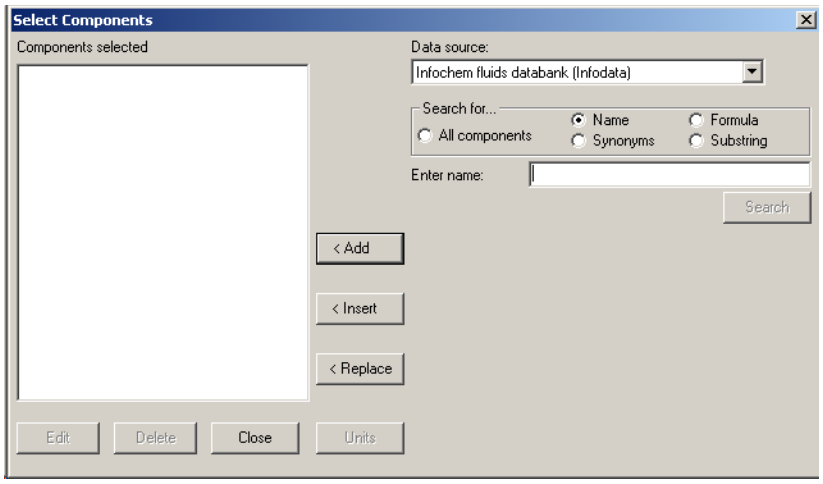


```
**********************************************************************
**********************************************************************
**                                                                  **
**          MULTIFLASH Version 3.9.35    November 2010              **
**                                                                  **
**                 Serial number: 3000/101                          **
**                 nouri2:"00059a3c7800 002215f413d8"               **
**                                                                  **
**   Copyright (C) Infochem Computer Services Ltd, 1989-2010        **
**                                                                  **
**********************************************************************
**********************************************************************

          Run on: 21 November 2010   at 17:49

Multiflash application files location: c:\program files\pse\gproms 3.4.0 i2010
\multiflash\
------------------------------------------------------------------------------
```

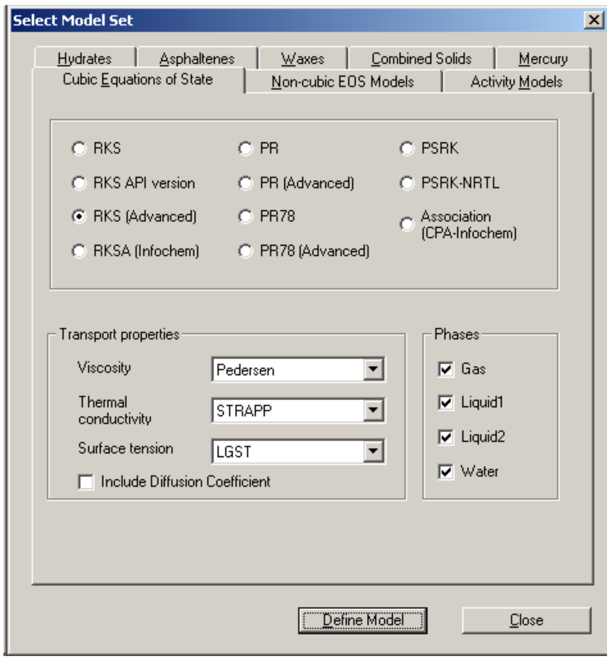
**Figure 2.2. The Components Dialog**



Data source: Infochem fluids databank (Infodata)

Search for...: [✓ Name]

**Figure 2.3. The Select Model Set Dialog**



RKS | PR | PSRK
RKS API version | PR (Advanced) | PSRK-NRTL
[✓ RKS (Advanced)] | PR78 | Association (CPA-Infochem)
RKSA (Infochem) | PR78 (Advanced)

Transport properties

| | |
|---|---|
| Viscosity | Pedersen |
| Thermal conductivity | STRAPP |
| Surface tension | LGST |

☐ Include Diffusion Coefficient

Phases

☑ Gas
☑ Liquid1
☑ Liquid2
☑ Water

## Creating the Multiflash input file by hand

In this section, we describe how to construct a Multiflash input file by hand, using a text editor. It may be helpful to note that a number of sample problem input files covering typical problem types are supplied with the Multiflash manual. It may be best to use one of these as the starting point for creating your own files.

As an illustration, suppose we are interested in a mixture consisting of four components: methyl-acetate, methanol, water and toluene. We will use a text editor to construct the corresponding Multiflash input file, specifying the information listed below in sequence. Note that each command line is terminated with a semicolon, and that comment lines can be inserted anywhere using the # symbol.

### Source of Pure Component Data

The data source (databank) for pure components is set using the command Puredata followed by the databank name:

```
PUREDATA    databank_name ;
```

where the databank name can be either Dippr (from AIChE) or the standard Infochem fluids databank Infodata.

### Components In a Mixture

Components are added to the mixture using the command Components followed by a list of components' names. If a particular component's name includes punctuation or spaces, then it should be enclosed in double quotation marks. Each name must be a valid component name for the databank considered. A list of components supported by Multiflash is given in the Multiflash manual. For the specific mixture considered here, the command takes the following form:

```
COMPONENTS    methylacetate methanol water toluene ;
```

### Source of Binary Interaction Parameters Data

Binary Interaction Parameters (BIPs) are required by most of the mixture models in Multiflash for use by the thermodynamic and/or transport properties models for mixtures. BIP data may be taken from a databank or entered directly on the command line. The command has the following form:

```
BIPDATA    databank_name ;
```

The oilandgas databank is the most frequently used one; it provides BIP values for typical compounds in oil and gas mixtures.

## Model Definition

The thermodynamic and transport property models to be used are specified through the Model command which has the following general syntax:

```
MODEL    model_id  MF_model_name  Model_options ;
```

where:

- *model_id* is a user-defined name that will be used to refer to the particular combination of the property model and options specified;
- *MF_model_name* is the Multiflash name for the basic model - the list of recognised models is given in the Multiflash manual;
- *Model_options* are additional keywords that describe model variants, references to other previously described models or references to the source of binary interaction parameters.

For example, to define a model called Model1 which involves the use of the Redlich-Kwong equation of state with a Soave modification, the command has the following form:

```
MODEL    model1 RKS ;
```

Any number of models may be defined in the same input file. For example, the command:

```
MODEL    model2 UNIFAC VLE model1 ;
```

defines a second model, called model2, that makes use of the UNIFAC liquid-phase activity model[6] Note that model2 specifies that the previously defined model1 is to be used for the calculation of gas phase properties.

Transport property models are optional and need only be defined if the computation of a transport property is required by the model. For example, in the common case where the Lohrenz-Bray-Clark (LBC) viscosity model is to be used, taking into account the model identified by model1 for the calculation of densities, the following command is added:

```
MODEL    model3 LBC * model1
```

where model3 is the user defined model name and the asterisk indicates that the LBC method should use the default option for matching the reference viscosities.

Similarly, if the Chung-Lee-Starling model (CLS) for the calculation of thermal conductivity is to be used based on the user defined model1 model (for the calculation of the required thermodynamic properties) the syntax has the format:

```
MODEL    model4 CLS model1 ;
```

Other models based on simple mixing rules are also available.

The models that have been mentioned above are the ones that are most often used for separations involving oil and gas mixtures. A complete list of all available models can be found in the Multiflash manual.

## Phase Descriptors

A phase descriptor is a user defined name that it is used to refer to a phase. It is necessary to define a phase descriptor for every phase that Multiflash is to consider when performing a calculation.

The general format of the Pd command used for defining a phase descriptor is as follows:

[6]This is used in many applications since it is completely predictive and does not require any BIPs.

```
PD    PD_ID  PHASE_TYPE  MODEL_IDENTIFIERS ;
```

where *PD_ID* is a user defined name that will be used to refer to the particular instance of phase type and associated models. *PHASE_TYPE* is a keyword that defines the phase type. Valid settings are Gas, Liquid, Solidsolution and Condensed (pure solid phase).

*MODEL_IDENTIFIERS* is a list of up to six models. These must have already been defined as described in: Model Definition. The same model may appear more than once in the list. The models specified determine the approach to be used for the computation of the thermodynamic and transport properties of the phase **in the following order**:

1. model to be used for fugacity (K-values)
2. model to be used for volume/density (optional)
3. model to be used for enthalpy/entropy (optional)
4. model to be used for viscosity (optional)
5. model to be used for thermal conductivity (optional)
6. model to be used for surface tension (optional)

For example, the command:

```
PD    PDLiquid LIQUID model2 model1 model1 ;
```

defines a phase descriptor PDLiquid that corresponds to a liquid phase. Model model2 will be used for the calculation of fugacities in this phase, while model model1 will be employed for the computation of both volume/densities and enthalpies.

On the other hand, the command:

```
PD    PDVapour VAPOUR model1 model1 model1 ;
```

defines a phase descriptor PDVapour corresponding to a vapour phase and using model model1 model for the calculation of all thermodynamic properties (fugacity, volume, enthalpy).

Note that transport properties are not needed for the example, considered here. Therefore, only the first three (of the six) models are specified against each phase descriptor.

### The complete Multiflash input file

The complete Multiflash input file for the example considered here will have the following form:

```
PUREDATA    DIPPR
COMPONENTS  methylacetate methanol water toluene ;
BIPDATA     oilandgas ;
MODEL       model1 RKS ;
MODEL       model2 UNIFAC VLE model1 ;
PD          PDLiquid LIQUID model2 model1 model1 ;
PD          PDVapour VAPOUR model1 model1 model1 ;
```

## Using Multiflash stream types

A Multiflash input file, of the kind described in: Constructing a Multiflash input file, specifies the information that is necessary for the computation of the thermophysical and transport properties of a particular type of material.

However, many processes involve more than one type of material. For instance, in addition to the main material being processed (which, say, is generally a mixture of several components), the process may also involve other materials used in auxiliary functions (e.g. cooling water).

Even when the set of components appears everywhere in the process, large variations in operating conditions may mean that it is expedient to use different thermodynamic and transport models in different sections of the plant

- thereby, at least from the modelling point of view, treating the matter in each of those sections as a different material.

For all these reasons, we need to be able to handle more than one type of material in our model. One way of doing this is by using separate Multiflash input files, one for each material, and creating a different instance for each one of them using the mechanisms described in: Models that involve Physical Property calculations.

A different, and computationally more efficient way, is by making use of a single Multiflash input file containing multiple "stream types"[7]

A stream type is a collection of components and phase descriptors, typically corresponding to the chemical species and the phases that may be present in a particular part of the plant being studied. Thus, these are generally subsets of, respectively, all the components and all the phase descriptors that are defined in a Multiflash input file.

## Defining stream types in the Multiflash input file

A stream may be introduced to a Multiflash input file using the Streamtype command (often abbreviated to ST). An extended version of the earlier input file is shown below:

```
PUREDATA    DIPPR
COMPONENTS  methylacetate methanol water toluene ;
BIPDATA     oilandgas ;
MODEL       model1 RKS ;
MODEL       model2 UNIFAC VLE model1 ;
PD          PDLiquid LIQUID model2 model1 model1 ;
PD          PDVapour VAPOUR model1 model1 model1 ;
ST Liq1 COMPONENTS methylacetate methanol; PDS PDLiquid PDVapour ;;
ST Wat  COMPONENTS water; PDS PDLiquid ;;
```

The last two lines of the extended file define two separate stream types called Liq1 and Wat respectively. The first one involves two of the four components declared in the input file while the second one contains only one of these components. Moreover, stream type Liq1 involves both vapour and liquid phases while Wat is just liquid.

Note that a semi-colon (;) is used to terminate each field of the stream type, as well as the stream type command itself - hence the double semi-colons at the end of each of the above commands. If the Components keyword is omitted, all the components defined in the input file are included in the stream type. Similarly, omitting the PDS keyword will include all the defined phase descriptors.

## Referring to stream types in gPROMS entities

Consider, for example, a gPROMS model of a plant which includes a water-cooled condenser .The latter makes use of two physical property Foreign Objects - one (called ProcessPPP) to compute the properties of the process stream being condensed and a second one (called CoolantPPP) to provide the properties of the coolant.

The condenser is situated in a part of the plant where no toluene exists. However, the reaction section of the plant needs to handle all four components declared in the Multiflash input file in both the vapour and the liquid phases. It does this by using another physical property Foreign Object, called GenericPPP.

Assuming that the Multiflash input file is called Organics.mfl, we can then specify the following values for the physical property Foreign Objects in the condenser:

```
SET
     Condenser.ProcessPPP  := "Organics.mfl<Liq1>" ;
     Condenser.CoolantPPP  := "Organics.mfl<Wat>"  ;
     RxnSection.GenericPPP := "Organics.mfl"       ;
```

We note that the first two specified values comprise the name of the Multiflash input file followed by the name of the stream type enclosed in angled brackets. As usual, each value is enclosed within double quotes. On the other hand, the last specification comprises simply the name of the Multiflash input file.

[7]These have no direct relationship with the concept of gPROMS stream types - refer to the gPROMS ModelDeveloper User Guide.

In this case, an invocation of the method ProcessPPP.LiquidEnthalpy within the condenser model would return the liquid enthalpy for the methyl-acetate/methanol mixture while CoolantPPP.LiquidEnthalpy will return the liquid phase enthalpy for the water coolant. Also, GenericPPP.LiquidDensity will return the liquid-phase density for the four-component system. On the other hand, any attempt to invoke CoolantPPP.Vapour-Enthalpy will result in a failure because no vapour phase descriptor has been defined for the Wat stream.

As the above example shows, as far as gPROMS is concerned, using multiple stream types within the same Multiflash input file is little different to using multiple Multiflash instances, each being described by a separate Multiflash input file. However, the use of streams within a single input file may result in significant benefits in computational efficiency during the actual computation.

It is also possible to combine mass basis (see also: Units of measurement) and stream type specifications as follows:

```
SET
    Condenser.CoolantPPP1  := "Organics.mfl<Wat>" ;      #molar basis
    Condenser.CoolantPPP2  := "MASS:Organics.mfl<Wat>" ; #mass basis
```

## The IK-CAPE physical property interface

The IK-CAPE physical property package has been developed by the IK-CAPE consortium of German chemical companies[8] with the aim of standardising the usage of physical properties across these companies.

If you wish to use the IK-CAPE package, you need to go through the following steps:

1. Check whether your installation is licensed to use IK-CAPE. You can do this by checking for the existence of file IKCAPE.so in the fo/IKCAPE sub-directory of the gPROMS installation directory on your computer system.

2. In your gPROMS project, ensure that you state the class of your Foreign Objects to be Ikcape (see lines 2 and 3 of the listing in: Incorporating Physical Properties in Models).

3. Create the IK-CAPE neutral file you would like to use. The form of this file is described in detail in the document "*Thermodynamik-Schnittstelle fur CAPE-Anwendungen: User's Guide*". Your neutral file should contain at least one material system[9]

4. In your gPROMS project, Set the value of your Foreign Object to the name of this neutral file. For example, if the name of the neutral file is BenzeneToluene.dat located in the *Miscellaneous Files* folder of the gPROMS project and the Foreign Object is called PhysProp, then the corresponding Set statement in the Model or Process will be:

```
SET
      PhysProp := "BenzeneToluene.dat" ;
```

5. Any of the methods listed in the first table in: Set of Physical Properties supported by gPROMS can now be used in your gPROMS input file with the exception of the BoilingPoint, LiquidEntropy, VapourEntropy, LiquidGibbsEnergy and VapourGibbsEnergy properties that are not currently supported by the IK-CAPE package.

## The CAPE-OPEN physical property interface (COThermoFO)

gPROMS comes as standard (on Microsoft Windows) with the *COThermoFO* Foreign Object which allows gPROMS models to make use of external thermodynamic and physical properties software that comply with

[8]Current membership (June 1998): BASF, BAYER, Degussa, DOW Chemical, Hoechst and Huls.

[9]The current implementation of the IK-CAPE physical property interface allows only one material system in each neutral file; if your file contains more than one such system, all but the first one will be ignored. If you wish to use multiple material systems with the IK-CAPE package, simply insert them in separate neutral files and treat the latter as different instances of class Ikcape in your project.

version 1.0 of the CAPE-OPEN Thermodynamic and Physical Properties specification (see http://www.colan.org/index-33.html).

## Introduction to property packages and systems

The CAPE-OPEN standard for thermodynamic and physical properties[10] is based on the concept of a *Property Package*. This is a software component that represents a particular mixture of interest; it involves a combination of:

- a specification of the species appearing in the mixture;
- a specification of the set of thermodynamic models that will be used for the computation of the physical properties of this mixture;
- a set of calculation routines for performing the above computations;
- all fundamental data required for the above computations (e.g. critical constants for pure components, binary interaction coefficients for pairs of components, and so on).

A collection of property packages is called a *Property System*.

Generally a property package will be created using a 3rd party CAPE-OPEN compatible physical properties software tool. Depending on the tool used any created property package may exist independently, or form part of a property system.

Before a CAPE-OPEN property package/system can be used in gPROMS (or any other CAPE-OPEN compliant process modelling environment), it has to have an appropriate entry in the Microsoft Windows registry of the computer on which it will be executing. How this is achieved will be specific to the 3rd party tool used, but in most cases the tool will automatically register the property package/system on the computer it was created on. If you need to register the property package/system on a different computer then you should consult the 3rd party documentation for the tool in question.

## Using COThermoFO in a gPROMS model

Once a CAPE-OPEN properties package/system has been installed, it can be used via the standard gPROMS mechanisms for physical properties packages. Thus, the user will typically:

- Introduce a Foreign Object for thermophysical property calculations in each MODEL entity that needs such calculations, (see also: Incorporating physical properties in Models).

```
PARAMETER
    PhysProps AS FOREIGN_OBJECT "Thermo"
VARIABLE
    ...
EQUATION
    F * h_in = L *  PhysProps.LiquidEnthalpy (T, P, x) +
               V *  PhysProps.VapourEnthalpy (T, P, y) ;
    ...
```

- Use standard gPROMS physical property calculation methods in the MODEL entity's equations (see also: The set of physical properties supported by gPROMS). Theoretically all the standard properties are supported[11] as a consequence of which it should be possible to use a CAPE-OPEN property package/system or other simple physical properties interface (e.g. Multiflash) interchangeably without the need for any modification to gPROMS models that make use of them.

  In addition to the standard properties, CAPE-OPEN property packages/systems may support additional properties. The full list and a more detailed discussion is given in: List of supported methods/properties.

[10]Full details are given in the document "CAPE-OPEN Open Interface Specification: Thermodynamic and Physical Properties Version 1.0" available at http://www.colan.org/index-33.html.
[11]Some CAPE-OPEN property packages/systems may not provide all the standard properties.

- In the PROCESS entity, declare the Foreign Object to belong to class *COThermoFO*. The "value" of the particular instance of this Foreign Object is set to the ProgId under which the CAPE-OPEN property package/ system has been registered in the Microsoft Windows registry (see also: Obtaining a list of available property packages/systems).

  For example, the following specification:

  ```
  UNIT
      F AS Flash
  SET
      F.PhysProps := "COThermoFO::ThermoCo.AroPack.1" ;
      ...
  ```

  specifies that the Foreign Object PhysProps appearing in instance F of MODEL Flash is a properties package that has been registered under the ProgId "ThermoCo.AroPack.1".

  The specification:

  ```
  SET
      F.PhysProps := "COThermoFO::ThermoCo.PropSys.1<AroPack>" ;
  ```

  specifies that PhysProps is a properties package named "AroPack" contained within a property system registered unded the ProgId "ThermoCo.PropSys.1".

## Obtaining a list of available property packages/ systems

A list of the available property packages/systems can be obtained by specifying an empty initialisation string to COThermoFO. e.g.

```
 SET
     F.PhysProps := "COThermoFO::" ;
```

When the resulting model is simulated it will fail, but the execution output will contain a list of all the available property packages/systems. e.g.

```
PROPERTY PACKAGES
-----------------
 0) ProgID:      PPDS.CapeSteamPackage.1
    CLSID:       {8E9B4FC1-439C-11D5-8E2D-00D0590F7D4D}
    Name:        PPDS CO Steam Package
    Description: PPDS CO Package for properties of steam (IAPS84)

PROPERTY SYSTEMS
----------------
 0) ProgID:      PPDS.CapeThermoSystem.1
    CLSID:       {032F7643-2F57-11D5-B7A8-0000E812B8B1}
    Name:        PPDS CO ThermoSystem
    Description: PPDS set of CO Packages
    Packages:    Hydrocarbon_test_package
                 Chemical_test_package
                 INDISS_test_package
                 MethanolSynthesis

 1) ProgID:      OATS.ThermoSystem.1
    CLSID:       {4CCF55DB-E332-42F8-B685-188989F8E1EC}
```

```
    Name:        OATS (CAPE-OPEN 1.0)
    Description: Out-of-proc Application Thermo Server: Thermo System
    Packages:    Multiflash Thermo System/METHANOLSYNTHESIS

 2) ProgID:      MFCOThermoSys.MFCOSys.1
    CLSID:       {653CE81C-DAD9-434B-B878-D5947CB16AD4}
    Name:        Multiflash Thermo System
    Description: Multiflash CAPE-OPEN v1.0 Thermo System
    Packages:    Air
                 BENZENEWATER
                 flash
                 FluentCSTR
                 FuelAir
                 HEXENE
                 HIDIC
                 METHANOLSYNTHESIS
                 METHANOLWATER
                 WATER

 3) ProgID:      COCO_TEA.ThermoPack.1
    CLSID:       {90DAC7FA-E0E4-40B5-A903-E0B12774D52B}
    Name:        TEA (CAPE-OPEN 1.0)
    Description: COCO Thermodynamics for Engineering Applications
    Packages:    C1_C2
                 C1_C2 (EOS)
                 n-depropropanizer
                 alkanes
                 HDA
                 Water-nButanol-UNIQUAC
                 MethanolSynthesis
                 MethanolWater
```

The same output can also be obtained at the Windows CMD line by executing the following command:

```
  SimplePME.exe -pp
```

## Optional flags to COThermoFO

The behaviour of COThermoFO is controlled by a number of optional flags that may be specified in the initialisation string after the property package/system identifier. e.g.

```
  SET
      F.PhysProps := "COThermoFO::ThermoCo.PropSys.1<AroPack> -mass -debug" ;
```

The available options are:

| | |
|---|---|
| `-mass` | the default behaviour of COThermoFO is to work on *mole* basis. Specifying this flag switches this to *mass* basis. |
| `-safe` | causes the underlying CAPE-OPEN material object to be cleared before each calculation.<br><br>This can make COThermoFO up to 10% slower and should only be specified if you have reasons to suspect that the property package/system or material object implementation is misbehaving. |
| `-debug` | causes COThermoFO to write a file called "output\CAPEOPEN.log" to the working directory. This file contains a log of all the property calculations made by COThermoFO and the corresponding interactions with the underlying CAPE-OPEN material object. |

| | |
|---|---|
| | Using this option can slow down execution considerably depending on the number of property calculations performed.<br><br>If you are running from within ModeBuilder the "CAPEOPEN.log" will automatically be imported into the case's 'Results' group. However note that if this file is over 2000 Kb in length it will be imported as a binary file and in order to read it you will need to export it and load it into a seperate text editor. |
| `-report` | causes COThermoFO to write a file called "output\COPropertyPackageReport.txt" to the working directory. This file contains a summary of the components, phases and properties supported by the specifed CAPE-OPEN property package(s).<br><br>If you are running from within ModeBuilder the report will automatically be imported into the case's 'Results' group. |
| `-subhf` | causes COThermoFO to subtract the enthalpy of formation from all enthalpies received from the property package and add the enthalpy of formation to all enthalpies sent to the property package.<br><br>This flag should be used when using the PML reactor models with a property package that includes the enthalpy of formation (e.g. Aspen Properties), otherwise the enthalpy of formation will be double counted by the model. |
| `-noexptransform` | Disables the so-called *exponential transformation* that is applied by default to near 0 and -ve component fractions in order to avoid sending -ve values to the underlying physical properties package and also to improve numerical stability.<br><br>Unfortunately this transformation can occasionally cause its own numerical problems so this flag is provided to disable it. |
| `-fasteqmprops` | Enables an optimisation for equilibrium property calculations. Not supported by all physical property packages. |

## Overriding the values returned by the COMPONENTS() method

The default behaviour of the COThermoFO COMPONENTS() method is to return the list of component id's supplied by the property package. This can prove inconvenient in 2 cases:

1. a given component id is rather unwieldy to use as an ORDERED_SET value in a gPROMS model, e.g. 'DIMETHYL-1,4-CYCLOHEXANEDICARBOXYLATE'

2. a gPROMS model might have been written expecting a particular component id to be 'H2O', but it is then switched to using a physical properties package that uses the component id 'WATER', or '7732-18-5' (the CAS No. of water).

To deal with these problems the component id's returned by COThermoFO can be overridden by providing a "input \COThermoFOAliases.txt" file of the following form:

```
# Lines beginning with a # character are ignored
# All component id's should be double quoted ""
"CanonicalName1" "IdA" "IdB" "IdC"
"CanonicalName2" "IdD"
"CanonicalName3" "IdE" "IdF"
# etc.
```

This tells COThermoFO that if the physical properties package contains:

- a component called "IdA", "IdB" or "IdC" then the COMPONENTS() method should return "CanonicalName1"
- a component called "IdD" then the COMPONENTS() method should return "CanonicalName2"
- a component called "IdE" or "IdF" then the COMPONENTS() method should return "CanonicalName3"

## List of supported methods/properties

The full lists of methods/properties supported by COThermoFO are given in the tables below[12].

To construct the name of the Foreign Object method corresponding to a given property it is sometimes necessary to prefix the property name with a phase name ("Vapor", "Liquid" or "Solid"). e.g. to calculate the density of a liquid at given conditions the Foreign Object method name will be "LiquidDensity". In contrast, the pure component vapour pressures are obtained from gPROMS by calling the method "VaporPressure" which does not need any additional prefix.

### Note

The CAPE-OPEN standard does not mandate a fixed list of properties that any given property package must support. As a result of this the methods usable by COThermoFO are package dependent. A list of the properties a given package supports can be obtained by using the '-report' option flag to COThermoFO, see Optional Flags to COThermoFO. In some cases a method corresponding to a property that is not directly supported by a package, e.g. "LogKValues" is still usable because COThermoFO can calculate it from one that is available, e.g. "KValues".

**Table 2.6. Methods corresponding to CAPE-OPEN universal constants [4.12.2]**

| CAPE-OPEN Property | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| StandardAccelerationOfGravity | $m/s^2$ | None | 9.80665 | |
| AvogadroConstant | 1/mol | None | 6.02214199(47) x $10^{23}$ | |
| BoltzmannConstant | J/K | None | 1.3806503(24) x $10^{-23}$ | |
| MolarGasConstant | J/(mol.K) | None | 8.314472(15) | |

**Table 2.7. Methods corresponding to CAPE-OPEN constant properties [4.12.1]**

| CAPE-OPEN Property | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| AcentricFactor | - | none | Vector | |
| AssociationParameter | - | none | Vector | |
| Born Radius | m | none | Vector | |
| Charge | - | none | Vector | |
| CriticalCompressibilityFactor | - | none | Vector | |
| CriticalDensity | $mol/m^3$ | none | Vector | |
| CriticalPressure | Pa | none | Vector | |
| CriticalTemperature | K | none | Vector | |
| CriticalVolume | $m^3/mol$ | none | Vector | b |
| DiffusionVolume | $m^3/mol$ | none | Vector | |
| DipoleMoment | C.m | none | Vector | |

[12]The numbered references in the table titles refer to the corresponding table in the document "CAPE-OPEN Open Interface Specification: Thermodynamic and Physical Properties Version 1.0" available at http://www.colan.org/index-33.html.

| CAPE-OPEN Property | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| EnergyLennardJones | K | none | Vector | |
| GyrationRadius | m | none | Vector | |
| HeatOfFusionAtNormalFreezingPoint | J/mol | none | Vector | b |
| HeatOfVaporizationAtNormalBoilingPoint | J/mol | none | Vector | b |
| IdealGasEnthalpyOfFormationAt25C | J/mol | none | Vector | b |
| IdealGasGibbsFreeEnergyOfFormationAt25C | J/mol | none | Vector | b |
| LengthLennardJones | m | none | Vector | |
| LiquidDensityAt25C | mol/m$^3$ | none | Vector | b |
| LiquidVolumeAt25C | m$^3$/mol | none | Vector | b |
| MolecularWeight | g/mol | none | Vector | |
| NormalBoilingPoint | K | none | Vector | |
| NormalFreezingPoint | K | none | Vector | |
| Parachor | m$^3$kg$^{0.25}$/ s$^{0.5}$mol | none | Vector | |
| RefractiveIndex | - | none | Vector | |
| SolubilityParameter | - | none | Vector | |
| SpecificGravity | - | none | Vector | |
| StandardEnthalpyAqueousDilution | J/mol | none | Vector | b |
| StandardEntropyGas | J/(mol.K) | none | Vector | b |
| StandardEntropyLiquid | J/(mol.K) | none | Vector | b |
| StandardEntropySolid | J/(mol.K) | none | Vector | b |
| StandardFormationEnthalpyGas | J/mol | none | Vector | b |
| StandardFormationEnthalpyLiquid | J/mol | none | Vector | b |
| StandardFormationEnthalpySolid | J/mol | none | Vector | b |
| StandardFreeFormationEnthalpyGas | J/mol | none | Vector | b |
| StandardFreeFormationEnthalpyLiquid | J/mol | none | Vector | b |
| StandardFreeFormationEnthalpySolid | J/mol | none | Vector | b |
| StandardGibbsAqueousDilution | J/mol | none | Vector | b |
| TriplePointPressure | Pa | none | Vector | |
| TriplePointTemperature | K | none | Vector | |
| VanderwaalsArea | m$^2$/mol | none | Vector | b |
| VanderwaalsVolume | m$^3$/mol | none | Vector | b |

## Table 2.8. Methods corresponding to CAPE-OPEN non-constant properties [4.12.3]

| Method | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| CompressibilityFactor | - | T,P,n | Scalar | p? |
| Density | kg/m3 | T,P,n | Scalar | d, p? |
| Energy | J | T,P,n | Scalar | b, e, p? |
| Enthalpy | J | T,P,n | Scalar | b, p? |
| Entropy | J/K | T,P,n | Scalar | b, p? |
| Expansivity | 1/K | T | Vector | |

| Method | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| FreeEnergy | J | T,P,n | Scalar | b, p? |
| Fugacity | Pa | T,P,n | Vector | p! |
| FugacityCoefficient | - | T,P,n | Vector | p! |
| GibbsFreeEnergy | J | T,P,n | Scalar | b, p? |
| GlassTransitionTemperature | K | P | Vector | |
| HeatCapacity | J/K | T,P,n | Scalar | b, p? |
| HeatCapacityCV | J/K | T,P,n | Scalar | b, p? |
| HeatOfFusion | J/mol | T | Vector | b |
| HeatOfSolidSolidPhaseTransition | J/mol | T | Vector | b |
| HeatOfSublimation | J/mol | T | Vector | b |
| HeatOfVaporization | J/mol | T | Vector | b |
| HelmholtzFreeEnergy | J | T,P,n | Scalar | b, p? |
| IdealGasEnthalpy | J/mol | T | Vector | b |
| IdealGasHeatCapacity | J/mol | T | Vector | b |
| InternalEnergy | J | T,P,n | Scalar | b, e, p?, e |
| Kvalues | - | T,P,nl,nv | Vector | |
| LiquidActivity | - | T,P,n | Vector | |
| LiquidActivityCoefficient | - | T,P,n | Vector | |
| LiquidExcessEnthalpy | J | T,P,n | Scalar | b |
| LogFugacityCoefficient | - | T,P,n | Vector | p! |
| LogKvalues | - | T,P,nl,nv | Vector | |
| MeltingPressure | Pa | T | Vector | |
| PhaseFraction | mol/mol | T,P,n | Scalar | b, p! |
| SolidSolidPhaseTransitionTemperature | K | P | Vector | |
| SolidSolidPhaseTransitionPressure | Pa | T | Vector | |
| SpeedOfSound | m/s | T,P,n | Scalar | p! |
| SublimationPressure | Pa | T | Vector | |
| SurfaceTension | N/m | T,P,nl,nv | Vector | |
| ThermalConductivity | W/(m.K) | T,P,n | Scalar | p! |
| VaporPressure | Pa | T | Vector | |
| VirialCoefficient | $m^3$/mol | T | Vector | b |
| Viscosity | Pa.s | T,P,n | Scalar | p! |
| Volume | $m^3$ | T,P,n | Scalar | b, p? |
| VolumeChangeUponMelting | $m^3$/mol | T | Vector | b |
| VolumeChangeUponSolidSolidPhaseTransition | $m^3$/mol | T | Vector | b |
| VolumeChangeUponSublimation | $m^3$/mol | T | Vector | b |
| VolumeChangeUponVaporization | $m^3$/mol | T | Vector | b |

## Table 2.9. Methods corresponding to CAPE-OPEN flash calculations

| Method | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| HSFlash | - | H,S,n | Vector | b |

| Method | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| PHFlash | - | P,H,n | Vector | b |
| PSFlash | - | P,S,n | Vector | b |
| PTFlash | - | P,T,n | Vector | b |
| PVFlash | - | P,V,n | Vector | b |
| PVFFlash | - | P,VF,n | Vector | b |
| SVFlash | - | S,V,n | Vector | b |
| THFlash | - | T,H,n | Vector | b |
| TPFlash | - | T,P,n | Vector | b |
| TSFlash | - | T,S,n | Vector | b |
| TVFlash | - | T,V,n | Vector | b |
| TVFFlash | - | T,VF,n | Vector | b |
| UVFlash | - | U,V,n | Vector | b |

## Table 2.10. Other supported methods

| Method | Units | Inputs | Result | Notes |
|---|---|---|---|---|
| BubblePressure | Pa | T,P,n | Scalar | |
| BubblePressureCompositions | mol/mol | T,n | Vector | b |
| BubbleTemperature | K | P,n | Scalar | |
| BubbleTemperatureCompositions | mol/mol | P,n | Vector | b |
| Components | - | None | Vector of strings | |
| CpCv | - | T,P,n | Scalar | p! |
| DewPressure | Pa | T,P,n | Scalar | |
| DewPressureCompositions | mol/mol | T,n | Vector | b |
| DewTemperature | K | P,n | Scalar | |
| DewTemperatureCompositions | mol/mol | P,n | Vector | b |
| NumberOfComponents | - | None | Scalar | |

## Table 2.11. Notes:

| | |
|---|---|
| 1. | COThermoFO does not necessarily provide methods corresponding to all of the CAPE-OPEN 1.0 properties. If we have missed one that you require (and is supported by your physical properties package) then let us know. |
| 2. | Scalar methods like "Enthalpy" return a single value, typically corresponding to a property of the mixture. |
| 3. | Vector methods like "MolecularWeight" return a vector of results, each element of which corresponds to a different component in the mixture.<br><br>The exception are the "Flash" methods these return a vector of `11 + 3 * NumberOfComponents` elements, arranged like so: |

**Table 2.12.**

| Element(s) | Property |
|---|---|
| 1 | Temperature |
| 2 | Pressure |
| 3 .. 2 + nocomp | Vapor molar fractions |
| 3 + nocomp | Vapor enthalpy |
| 4 + nocomp | Vapor molar amount |
| 5 + nocomp | Vapor volume |
| 6 + nocomp .. 5 + 2 * nocomp | Liquid_1 molar fractions |
| 6 + 2 * nocomp | Liquid_1 enthalpy |
| 7 + 2 * nocomp | Liquid_1 molar amount |
| 8 + 2 * nocomp | Liquid_1 volume |
| 9 + 2 * nocomp .. 8 + 3 * nocomp | Liquid_2 molar fractions |
| 9 + 3 * nocomp | Liquid_2 enthalpy |
| 10 + 3 * nocomp | Liquid_2 molar amount |
| 11 + 3 * nocomp | Liquid_2 volume |

However currently only a single liquid phase is supported so the last (nocomp + 3) elements of this vector will always be set to 0.0.

4. COThermoFO (like all gPROMS physical properties foreign objects) deals in total quantities, not specific quantities:

i.e. if you call "Enthalpy" with `n` where `SUM(n) = 2` then you get the enthalpy of 2 mol (or kg) of the mixture. If `SUM(n) = 1` then you get the enthalpy of 1 mol (or kg) which of course corresponds to the specific quantity.

5. The inputs are:

**Table 2.13.**

| | |
|---|---|
| E | Enthalpy (J) |
| nl | Amount in liquid phase, a vector of "NumberOfComponents" elements |
| nv | Amount in vapour phase, a vector of "NumberOfComponents" elements |
| P | Pressure (Pa) |
| S | Entropy (J) |
| T | Temperature (K) |
| U | Internal Energy (J) |

| | | |
|---|---|---|
| | V | Volume ($m^3$) |
| | VF | Vapor phase fraction (mol/mol or kg/kg) |

| | |
|---|---|
| 6. | The current version of COThermoFO does not make use of partial derivatives exposed by the underlying property package. Where partial derivatives are required by gPROMS they will be calculated by finite differences. |
| b | Though the units of measurement of this property are described in mole terms they and the returned value depend on whether mole or mass basis is specified for COThermoFO. |
| d | The units of "Density" are always kg/m3 even when you specify mole basis for COThermoFO. |
| e | "Energy" and "InternalEnergy" are the same property, the CAPE-OPEN standard calls it "Energy", but Multiflash provides "InternalEnergy" as a synonym. |
| p? | The name of this method *may* be prefixed with a phase name ("Vapor", "Liquid" or "Solid"). If no prefix is specified then the overall quantity is calculated. |
| p! | The name of this method *must* be prefixed with a phase name. |

## The COThermoFO Test Tool

To help users and PSE support staff diagnose interoperability issues between COThermoFO and CAPE-OPEN compliant property packages/systems the Windows version of gPROMS comes with a command-line utility called COThermoFOTestTool.exe.

This utility uses the COThermoFO to perform a set of physical property calculations on a CAPE-OPEN property package/system and generates a report of the successes and failures. The calculations to perform, and the expected results are described by an XML text file.

```
COThermoFOTestTool.exe [-ignoreresults] xmlFile
```

| | |
|---|---|
| `-ignoreresults` | only test that the FO method calls are succeeding, not that they are returning the expected value |
| *`xmlFile`* | file containing an XML description of the tests to perform, the format of which is described below |

**Example 2.1. Example COThermoFOTestTool XML file**

```
<?xml version="1.0"?>
<ForeignObject id='COThermoFO::MFCOThermoSys.MFCOSys&lt;methanolwater.mfl&gt;>  ❶
<Method name="TPFLASH" checkmethod="true" evalmethod="true"> ❷
  <Inputs> ❸
    <Input type="real" name="temperature">
      <Value>347</Value>
    </Input>
    <Input type="real" name="pressure">
      <Value>100000</Value>
    </Input>
    <Input type="real" name="amount">
      <Value>0.7</Value>
      <Value>0.3</Value>
    </Input>
  </Inputs>
  <Output type="real"> ❹
    <Value delta="0.1">347</Value>
    <Value delta="1">100000</Value>
    <Value delta="1e-005">0.761842</Value>
    <Value delta="1e-005">0.238158</Value>
    <Value delta="0.01">1562.04</Value>
    <Value delta="1e-005">0.797261</Value>
    <Value delta="1e-007">0.0226528</Value>
    <Value delta="1e-005">0.456811</Value>
    <Value delta="1e-005">0.543189</Value>
    <Value delta="0.1">-7416.28</Value>
    <Value delta="1e-005">0.202739</Value>
    <Value delta="1e-009">6.05012e-006</Value>
    <Value delta="1e-006">0</Value>
    <Value delta="1e-006">0</Value>
    <Value delta="1e-006">0</Value>
    <Value delta="1e-006">0</Value>
    <Value delta="1e-006">0</Value>
  </Output>
</Method>
<Method name="NUMBEROFCOMPONENTS" checkmethod="true" evalmethod="true">
  <Inputs> ❺
  </Inputs>
  <Output type="real">
    <Value delta="1e-006">2</Value>
  </Output>
</Method>
<!-- Not implemented by Multiflash -->
<Method name="STANDARDFORMATIONENTHALPYSOLID" checkmethod="false" evalmethod="false">
  <Inputs>
  </Inputs>
  <Output type="real">
  </Output>
</Method>
</ForeignObject>
```

❶ The COThermoFO initialisation string; the format is the same as when using COThermoFO within gPROMS except the < and > symbols are replaced by &lt; and &gt;.

❷ The name of the method and whether to test the method signature (checkmethod) and/or the method evaluation (evalmethod).

❸ The methods inputs; each input has a name, type (always real) and one or more values.

❹ The method output; this has a type (always real) and one or more values. Each value includes a delta by which the returned value is allowed to differ from the expected value without an error being reported.

❺ Methods such as `NUMBEROFCOMPONENTS()` have no inputs and so an empty <Inputs> element is specified.

❻ Tests can also be included for methods/properties which you do not expect the property package/system to support. An error is then reported if the call succeeds.

# Chapter 3. Using Physical Properties for Complex Materials

The state of the simple materials considered in the previous chapter on using physical properties for simple materials can be described in terms of their thermodynamic phase and their temperature, pressure, and molecular composition (e.g. a vector of molar fractions). This chapter is concerned with more complex materials that need to be characterised by additional properties or attributes. The latter may include size, shape, electrical charge, porosity and surface roughness. Examples of complex materials include electrolytes, polymers, petroleum fractions and particulate solids.

gPROMS provides a generalised interface to physical properties packages for electrolytic systems. To make the most of this chapter, you should already be familiar with the concepts introduced in the previous chapter on using physical properties for simple materials. You should then read:

- General concepts for electrolyte system modelling. This describes the general concepts that underpin all electrolytic physical property packages interfaced to gPROMS. It is important to understand these concepts to be able to make effective use of the facilities provided by these packages in your models.
- Electrolytic physical property methods. This describes the methods provided by all physical property packages interfaced to gPROMS via the standard electrolytic physical properties interface.
- The OLI physical property interface. This describes how to use one particular interface, namely that to the OLI[1] package for aqueous electrolytic systems.

## General concepts for electrolyte system modelling

A description of the general concepts that underpin the gPROMS electrolytic physical properties interface is given here.

### Species and phases

An electrolytic material may involve several different classes of species, including:

- molecular species (e.g. $H_2O, Cl, O_2$)
- ionic species (e.g. $Cl^-, OH^-, H^+, Na^+$)
- solid components (e.g. $NaCl, Na_2SO_4$)
- hydrates (e.g. $NaOH.1H_2O$)

In general, the above species may occur in a number of thermodynamic *phases*. The electrolytic physical property interface assumes that the following phases may appear in a system:

- a single vapour phase comprising a mixture of molecular species;
- a single liquid phase comprising a mixture of molecular species and ions;
- one or more solid phases, each comprising a single solid component or hydrate.

The interface makes the following additional assumptions:

- All species that exist in the vapour phase will also exist in the liquid phase.

[1]OLI Systems Inc. (http://www.olisystems.com)

- Species in the solid phase(s) do not exist in either the vapour or liquid phases.
- If the same chemical component exists in both solid and liquid phases, it will be modelled as two different species.
- The species are ordered as follows:
  - species that occur in both vapour and liquid phases, followed by,
  - species that occur in the liquid phase only, followed by,
  - species that occur in the solid phase only.

As an example, the following species may be present in a system comprising sodium chloride and water:

Vapour: $H_2O(vap), HCl(vap)$

Liquid: $H_2O(liq), HCl(aq), OH^-, H^+, Na^+, Cl^-$

Solid: $NaCl(sol), NaOH(sol), NaOH.1H_2O(sol)$

Note that the solid species include hydrates and precipitates.

## Vapour-liquid equilibrium

The vapour-liquid equilibrium (VLE) relationships pertain to species that exist in both liquid and vapour phases. For the sodium chloride example considered earlier, the following VLE equilibria may be established:

$$H_2O(vap) \rightleftharpoons H_2O(liq)$$

$$HCl(vap) \rightleftharpoons HCl(aq)$$

The VLE equations are given by:

$$K_i^{vl}(T,P) \;\; f_i^v(T,P,y) = f_i^l(T,P,x) \qquad i = 1,\ldots,N^v$$

where $K_i^{vl}$ is the vapour-liquid equilibrium constant for species *i*, $f_i^v$ and $f_i^l$ are, respectively, vapour and liquid-phase fugacities of component *i*, and $N^v$ is the number of species present in the vapour phase[2] Here, *T* and *P* are the system temperature and pressure while *x* and *y* denote molar fractions in the liquid and vapour phases respectively.

## Liquid-phase reactions

Liquid-phase reactions may take place among the molecular and ionic species in the liquid phase. For example, the following reactions may be considered for the sodium chloride system introduced earlier:

$$H_2O \rightleftharpoons H^+ + OH^-$$

$$HCl(aq) \rightleftharpoons H^+ + Cl^-$$

The equilibria corresponding to the above reactions are generally described by equations of the form:

$$K_j^{lr}(T,P) = \prod_{i=1}^{N^l} (f_i^l)^{\nu_{ji}^l} \qquad j = 1,\ldots,M^l$$

where $K_j^{lr}$ is the equilibrium constant for reaction *j*, $\nu_{ji}^l$ are the stoichiometric coefficients for the liquid phase reactions, $M^l$ is the number of liquid-phase reactions, and $N^l$ is the number of species in the liquid phase.

[2]As has already been mentioned, all of these are assumed also to exist in the liquid phase.

## Liquid-solid equilibrium

Each solid species in the system is assumed to undergo dissociation to form species in the liquid phase. For the sodium chloride example, the solid dissociation reactions are:

$$NaCl(sol) \rightleftharpoons Na^+ + Cl^-$$

$$NaOH.1H_2O \rightleftharpoons Na^+ + OH^- + H_2O(liq)$$

$$NaOH(sol) \rightleftharpoons Na^+ + OH^-$$

The equilibria of solid dissociation reactions are generally described by equations of the form:

$$K_j^{sr}(T,P) = \prod_{i=1}^{N^l} (f_i^l)^{\nu_{ji}^s} \qquad j = 1, \ldots, N^s$$

where $K_j^{sr}$ is the equilibrium constant (the "solubility product") for species *j*, $\nu_{ji}^s$ is the stoichiometric matrix for the solid dissociation reactions[3] and $N^s$ is the number of solid species.

## Fugacity and activity coefficients

The equations presented on Vapour-liquid equilibrium, Liquid-phase reactions and Liquid-solid equilibrium have been expressed in terms of the vapour and liquid-phase fugacities, $f_i^v$ and $f_i^l$ respectively.

Alternative quantities, such as the vapour-phase fugacity coefficients$\phi_i^v$ and the (liquid-phase) activity coefficients $\gamma_i$, are often used in electrolytic system modelling. The relation between vapour-phase fugacity and fugacity coefficient is given by:

$$f_i^v = \phi_i^v(T,P,y)\, y_i\, P \qquad i = 1, \ldots, N^v$$

As far as the liquid phase is concerned, the gPROMS electrolytic physical property interface treats the *first* species in the system as the (main) solvent (e.g. water). The liquid-phase fugacities for the *other* species are related to the corresponding activity coefficients via:

$$f_i^l = \gamma_i(T,P,x) m_i \qquad i = 2, \ldots, N^l$$

Here $m_i$ is the *molality* of species *i* in the liquid phase defined as:

$$m_i = \frac{x_i}{x_1\ MW_1} \qquad i = 2, \ldots, N^l$$

where $MW_1$ is the molecular weight of species 1 (in kg/mol).

# Electrolytic physical property methods

The gPROMS Electrolytic Physical Properties interface specifies that all electrolytic physical property packages interfaced to gPROMS need to provide the Thermophysical property methods and their arguments listed in the first table below.

The form of the methods described here has been chosen so as to maximise the reliability and accuracy of any gPROMS within which they are incorporated. This is particularly important in view of the extremely low

[3]Note that dissociation reactions are always written with the solid species appearing on the *left* hand side of the reaction. Hence, all coefficients $\nu_{ji}^s$ used in the above equation are non-negative.

concentrations of some species in many electrolytic applications. We recommend that you make use of these methods in any models that you develop[4] In this manner, you will also ensure that your models will be usable with any electrolytic physical property package interfaced to gPROMS.

The following points should be noted about the tables below:

1. The inputs of the methods in the first table are as follows:
   - T : temperature (K)
   - P : pressure (Pa)
   - nx : **natural logarithm** of the amounts of each species in the liquid phase (in mol)
   - ny : **natural logarithm** of the amounts of each species in the vapour phase (in mol)
   - ns : **natural logarithm** of the amounts of each species in the solid phase (in mol)
   - nz : **natural logarithm** of the amounts of each molecular species (in mol)
2. he following methods return the **natural logarithms** of the corresponding quantities:
   - VapourFugacity : $\ln(f_i^v)$
   - LiquidFugacity : $\ln(f_i^l)$
   - VapourFugacityCoefficients : $\ln(\phi_i^v)$
   - LiquidActivityCoefficients : $\ln(\gamma_i)$
   - VapourLiquidEquilibriumConstants : $\ln(K_j^{vl})$
   - LiquidReactionEquilibriumConstants : $\ln(K_j^{lr})$
   - SolidDissociationEquilibriumConstants : $\ln(K_j^{sr})$
3. The datum for all enthalpy calculations is the elemental species at standard conditions. One implication of this is that "heat generation" terms must *not* be included in energy conservation equations to account for chemical reactions.
4. The results of the two equilibrium flash methods (MolecularTPEquilibrium and TrueSpeciesTPEquilibrium) are organised as detailed in the third table. The quantities returned in the results vector of the methods in the third table include the **natural logarithms** of molar fractions and amounts.
5. For convenience in the construction of some complex models, three additional quantities are defined:
   - *Vapour Liquid Sum* (*VLS*)

     $$VLS_i = \ln(f_i^l) - \ln(f_i^v) - \ln(K_i^{vl}) \qquad j = 1, \ldots, N^v$$

     Thus, the *VLS* is the logarithm of the ratio of the right hand side of equation

     $$K_i^{vl}(T,P) \;\; f_i^v(T,P,y) = f_i^l(T,P,x) \qquad i = 1, \ldots, N^v$$

     to its left hand side. Under conditions of vapour/liquid equilibrium, its value should be zero.

[4]The details of how precisely to refer to Foreign Object methods in your Models are described in: Using Foreign Objects in gPROMS Models and Incorporating Physical Properties in Models.

- *Liquid Reaction Sum* (*LRS*)

  $LRS_j = \sum_{i=1}^{N^l} \nu_{ji}^l \ln(f_i^l) - \ln(K_j^{lr}) \qquad j = 1, \ldots, N^l$

  Thus, the *LRS* is the logarithm of the ratio of the right hand side of equation

  $K_j^{lr}(T, P) = \prod_{i=1}^{N^l} (f_i^l)^{\nu_{ji}^l} \qquad j = 1, \ldots, M^l$

  to its left hand side. Under conditions of reaction equilibrium, its value should be zero.

- Solid Dissociation Sum (*SDS*)

  $SDS_j = \sum_{i=1}^{N^l} \nu_{ji}^s \ln(f_i^l) - \ln(K_j^{sr}) \qquad j = 1, \ldots, N^s$

  Thus, the *SDS* is the logarithm of the ratio of the right hand side of equation

  $K_j^{sr}(T, P) = \prod_{i=1}^{N^l} (f_i^l)^{\nu_{ji}^s} \qquad j = 1, \ldots, N^s$

  to its left hand side. If the solid species *j* is in equilibrium with the liquid phase, its value should be zero.

The three above quantities are returned by methods VapourLiquidSum, LiquidReactionSum and SolidDissociationSum respectively.

**Table 3.1. Thermophysical property methods and their arguments**

| Property Name | Inputs | Description | Type | Units |
|---|---|---|---|---|
| NumberOfSpecies | - | Total number of species in the system. | Scalar | - |
| NumberOfVapourSpecies | - | Number of vapour species. | Scalar | - |
| NumberOfLiquidSpecies | - | Number of liquid species. | Scalar | - |
| NumberOfSolidSpecies | - | Number of solid species. | Scalar | - |
| NumberOfMolecularSpecies | - | Number of whole molecular species. | Scalar | - |
| MolecularWeight | - | Molecular weights of the species. | Vector | g/mol |
| SpeciesCharge | - | Electric charges of the species. | Vector | - |
| NumberOfLiquidReactions | - | Number of liquid phase reactions. | Scalar | - |
| LiquidReactionMatrix | - | Stoichiometric coefficient matrix for the liquid-phase reactions. | Vector | - |
| SolidDissociationMatrix | - | Stoichiometric coefficient matrix for the solid dissociation reactions. | Vector | - |
| LiquidSolidReactionMatrix | - | Stoichiometric coefficient matrix for the liquid-phase and solid dissociation reactions. | Vector | - |
| VapourLiquidEquilibriumConstants | T,P | Equilibrium constants for the vapour-liquid equilibria. | Vector | - |
| LiquidReactionEquilibriumConstants | T,P | Equilibrium constants for the liquid phase reactions. | Vector | - |

| Property Name | Inputs | Description | Type | Units |
|---|---|---|---|---|
| SolidDissociationEquilibriumConstants | T,P | Equilibrium constants for the solid dissociation reactions. | Vector | - |
| VapourLiquidSum | T,P,ny,nx | Vapour-liquid equilibrium sums for vapour species. | Vector | - |
| LiquidReactionSum | T,P,nx | Liquid reaction equilibrium sums for liquid-phase reactions. | Vector | - |
| SolidDissociationSum | T,P,nx | Solid dissociation sums for solid species. | Vector | - |
| VapourEnthalpy | T,P,ny | Total enthalpy of the vapour phase. | Scalar | J |
| VapourVolume | T,P,ny | Total volume of the vapour phase. | Scalar | $m^3$ |
| VapourDensity | T,P,ny | Density of the vapour phase. | Scalar | $kg/m^3$ |
| VapourFugacity | T,P,ny | Vapour fugacity of all species in the vapour phase. | Vector | - |
| VapourFugacityCoefficients | T,P,ny | Vapour fugacity coefficients of all species in the vapour phase. | Vector | - |
| LiquidEnthalpy | T,P,nx | Total enthalpy of the liquid phase. | Scalar | J |
| LiquidVolume | T,P,nx | Total volume of the liquid phase. | Scalar | $m^3$ |
| LiquidDensity | T,P,nx | Density of the liquid phase. | Scalar | $kg/m^3$ |
| LiquidFugacity | T,P,nx | Liquid fugacity of all species in the liquid phase. | Vector | - |
| LiquidActivityCoefficients | T,P,nx | Activity coefficients of all the species in the liquid phase. | Vector | - |
| SolidEnthalpy | T,P,ns | Total enthalpy of the solid phase. | Scalar | J |
| SolidVolume | T,P,ns | Total volume of the solid phase. | Scalar | $m^3$ |
| SolidDensity | T,P,ns | Density of the solid phase. | Scalar | $kg/m^3$ |
| pH | T,P,nx | pH of the liquid phase. | Scalar | - |
| MolecularTPEquilibrium | T,P,nz | Equilibrium flash calculation (at given T, P, and molecular species amounts, nz). | Vector | - |
| TrueSpeciesTPEquilibrium | T,P,ny,nx,ns | Equilibrium flash calculation (at given T, P, and true species amounts). | Vector | - |
| BubbleTemperature | P,nx | Bubble point temperature. | Scalar | K |
| BubblePressure | T,nx | Bubble point pressure. | Scalar | Pa |
| DewTemperature | P,ny | Dew point temperature. | Scalar | K |
| DewPressure | T,ny | Dew point pressure. | Scalar | Pa |

Table 3.2. Details of the vector methods in the interface

| | |
|---|---|
| T | Temperature, K. |

| | |
|---|---|
| P | Pressure, Pa. |
| Nvap | Number of vapour species. |
| Nliq | Number of liquid species. |
| Nsol | Number of solid species. |
| NoS | Number of species = Nliq+Nsol. |
| Ninp | Number of molecular species in feed. |
| NliqR | Number of liquid equilibrium constants (reactions). |
| nx | molar amounts of all true species in liquid phase(size=NoS). |
| ny | molar amounts of all true species in vapour phase(size=NoS). |
| ns | molar amounts of all true species in solid phase(size=NoS). |
| nz | molar amounts of the molecular species in feed(size=Ninp). |
| Name of vector property | Size |
| MolecularWeight | NoS |
| SpeciesCharge | NoS |
| ReactionType | Nvap+NliqR+Nsol |
| ReactionNumberToSolidSpecies | NliqR |
| VapourLiquidEquilibriumConstants | Nvap |
| VapourFugacity | Nvap |
| VapourFugacityCoefficients | Nvap |
| LiquidReactionEquilibriumConstants | NliqR |
| SolidDissociationEquilibriumConstants | Nsol |
| LiquidActivityCoefficients | Nliq |
| LiquidFugacity | Nliq |
| MolecularTPEquilibrium | 14+Nvap+Nliq+Nsol |
| TrueSpeciesTPEquilibrium | 14+Nvap+Nliq+Nsol |
| LiquidReactionMatrix | NliqR*Nliq |
| SolidDissociationMatrix | Nsol*Nliq |
| LiquidSolidReactionMatrix | (NliqR+Nsol)*NoS |
| VapourLiquidSum | Nvap |
| LiquidReactionSum | NliqR |
| SolidDissociationSum | Nsol |

**Table 3.3. Organisation of results vector for methods MolecularTPEquilibrium and TrueSpeciesTPEquilibrium**

| Variable | Length | Position in results vector | | Units |
|---|---|---|---|---|
| | | From | To | |
| Temperature | 1 | 1 | 1 | K |
| Pressure | 1 | 2 | 2 | Pa |

| Variable | Length | Position in results vector | | Units |
|---|---|---|---|---|
| Vapour molar fractions | Nvap | 3 | 2+Nvap | - |
| Vapour enthalpy | 1 | 3+Nvap | 3+Nvap | J |
| Vapour molar amount | 1 | 4+Nvap | 4+Nvap | mol |
| Vapour volume | 1 | 5+Nvap | 5+Nvap | $m^3$ |
| Vapour density | 1 | 6+Nvap | 6+Nvap | $kg/m^3$ |
| Liquid molar fractions | Nliq | 7+Nvap | 6+Nvap+Nliq | - |
| Liquid enthalpy | 1 | 7+Nvap+Nliq | 7+Nvap+Nliq | J |
| Liquid molar amount | 1 | 8+Nvap+Nliq | 8+Nvap+Nliq | mol |
| Liquid volume | 1 | 9+Nvap+Nliq | 9+Nvap+Nliq | $m^3$ |
| Liquid density | 1 | 10+Nvap+Nliq | 10+Nvap+Nliq | $kg/m^3$ |
| Solid molar fractions | Nsol | 11+Nvap+Nliq | 10+Nvap+Nliq+Nsol | - |
| Solid enthalpy | 1 | 11+Nvap+Nliq+Nsol | 11+Nvap+Nliq+Nsol | J |
| Solid molar amount | 1 | 12+Nvap+Nliq+Nsol | 12+Nvap+Nliq+Nsol | mol |
| Solid volume | 1 | 13+Nvap+Nliq+Nsol | 13+Nvap+Nliq+Nsol | $m^3$ |
| Solid density | 1 | 14+Nvap+Nliq+Nsol | 14+Nvap+Nliq+Nsol | $kg/m^3$ |

## The OLI physical property interface

OLI is a physical property package for aqueous electrolytic systems developed and marketed by OLI Systems, Inc.[5]

To use OLI with gPROMS, you need to go through the following steps:

1. Check whether your installation is licensed to use OLI. You can do this by checking for the existence of either the file OLI.so or OLI.dll in the fo/OLI subdirectory of the gPROMS installation directory on your computer system.
2. In your gPROMS project, ensure that you state the class of your Foreign Objects to be OLI (see lines 2 and 3 of the listing below).

```
2  PARAMETER
3    PPP     AS                    FOREIGN_OBJECT "OLI"
```

3. Specify the system chemistry by creating the chemistry model or data file. This file encapsulates all the thermodynamic details of the system to be studied and will be used internally by the interface to perform all the calculations. The chemistry model will contain information on speciation, all the equilibrium equations, species charges and stoichiometry, pure component constants, various correlation coefficients, etc.

   The best way to create the chemistry model file is via the OLI Toolkit. In choosing the phases present, it is advisable to specify the vapour and aqueous (liquid) phases; include the solid phase only if there is a possibility of solid formation (e.g. as a result of salt precipitation or hydrate formation).

   At the end of the chemistry model definition, a text file with extension .dbs will be created. This is the only file that the gPROMS-OLI interface will need from the OLI package.

4. In your gPROMS project, Set the value of your Foreign Object to the name of the chemistry model file including the .dbs extension. For example, if the name of the chemistry model file is SodiumChloride.dbs located in

[5]http://www.olisystems.com.

the Miscellaneous Files of the gPROMS project tree and the Foreign Object is called PhysProp, then the corresponding Set statement in the Model or Process will be:

```
SET
   PhysProp := "SodiumChloride.dbs" ;
```

5. Any of the methods listed in first table in: electrolytic physical property methods can now be used in your gPROMS project. Two points are worth noting in this context:

   - The first component in the list of species will always be water.
   - The first element of the vector returned by the method LiquidActivityCoefficients will contain the activity of water rather than its activity *coefficient*. Subsequent elements will contain the activity coefficients of the remaining species.

## Electrolyte system report file

At the start of the gPROMS simulation run, the interface will automatically initialise the chemistry model that you have specified (see above). If this initialisation is successful, an Electrolytic System Report (ESR) file will be produced in the *Results* folder of the Case.

The ESR file will be placed in the Output sub-directory of your gPROMS working directory. It will have the extension .esr and the same base name as the .dbs file. For example, if the chemistry model file is SodiumChloride.dbs, then the ESR file will be called SodiumChloride.esr.

Example of an Electrolyte System Report File

```
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +                                                                    +
 +                                                                    +
 +                                                                    +
 +               gPROMS ELECTROLYTE SYSTEM REPORT                     +
 +                                                                    +
 +                                                                    +
 +       Created from OLI chemistry model file: nacl.dbs              +
 +                                                                    +
 +                                                                    +
 +                                                                    +
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


SPECIES                                 PHASES IN WHICH PRESENT
=======                                 =======================

                        Vapour                 Liquid                 Solid
                        ------                 ------                 -----
 1  H2O                    *                      *
 2  HCL                    *                      *
 3  OH(-)                                         *
 4  H(+)                                          *
 5  NA(+)                                         *
 6  CL(-)                                         *
 7  NACL(ppt)                                                            *
 8  NAOH(ppt)                                                            *
 9  NAOH.1H2O                                                            *

 Number of vapour species = 2
 Number of liquid species = 6
 Number of solid  species = 3
 Total  number of species = 9


VAPOUR-LIQUID EQUILIBRIA
========================
[VL  1]   H2O(vap) <=> H2O(aq)
[VL  2]   HCL(vap) <=> HCL(aq)


IONIC EQUILIBRIA
================
[IE  1]   HCL(aq) <=>  H(+) +  CL(-)
[IE  2]   H2O <=>  OH(-) +  H(+)


SOLID-LIQUID (DISSOCIATION) EQUILIBRIA
======================================
[SD  1]   NACL(ppt) <=>  NA(+) +  CL(-)
[SD  2]   NAOH.1H2O <=>  H2O +  OH(-) +  NA(+)
[SD  3]   NAOH(ppt) <=>  OH(-) +  NA(+)
```

An example of an ESR file is given in the listing above. As can be seen, it contains the details of the species present (names and phases) and the vapour-liquid equilibria, liquid phase reactions and solid dissociation reactions that take place in the system.

### Caution

Always study the information in the ESR file carefully to ensure that gPROMS' interpretation of your chemistry file is indeed what you intended to specify!

## Error handling in the OLI interface

In general, errors occur in two phases of the simulation in gPROMS: initialisation and calculations.

The initialisation of an OLI Foreign Object consists of loading the chemistry model file (.dbs file) and checking all the Foreign Object methods used in your gPROMS Models. At the end of a successful loading of the chemistry model file, an electrolyte system report file will be generated. Any error messages that arise from this step will be printed on the screen. A common error will be the inability to locate the chemistry model file specified. Please check that either the full pathname of the file or the pathname relative to the gPROMS export directory is given.

The initialisation step will also involve checking all the methods used in the gPROMS Models. Any errors detected here will also be printed to the screen. Please use the ESR file generated to help in locating errors. Also, check that the names of the methods are exactly as given in first table in Electrolytic physical property methods.

The second type of errors arise during the gPROMS Process initialisation and integration. All error messages will be printed on the screen.