Thermoptim reference manual Volume 3

1 Foreword

Thermoptim documentation

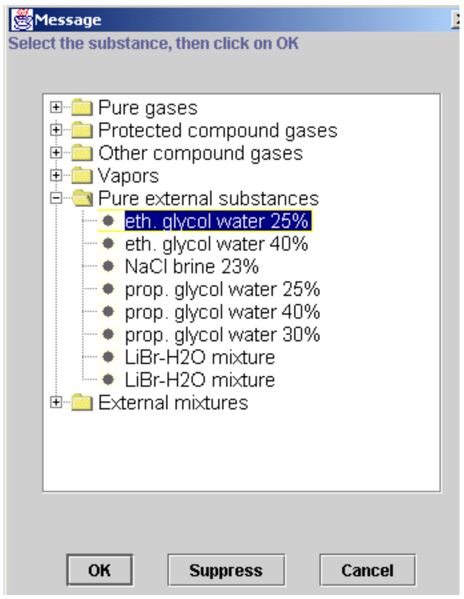
Thermoptim documentation is comprised of several complementary parts:

- a short documentation named Quick Reference available through menu Help: it gives access to tab frames introducing the main concepts used
- a printable documentation, mainly as pdf files
- on line e-learning modules with sound tracks named Diapason
- guided explorations of Thermoptim models

The documentation available is presented in Volumes 1 and 2 of the reference manual.

This is the third volume. It is dedicated to the design of external classes. After a quick introduction to the extension mechanism that has been adopted, it explains how to use and program external classes, and introduces a freely distributed development environment.

2 Extension system for Thermoptim by adding external classes



Select the substance, then click on OK

- Pure gases
- Protected compound gases
- Other compound gases
- Vapors
- Pure external substances
  - eth. glycol water 25%
  - eth. glycol water 40%
  - NaCl brine 23%
  - prop. glycol water 25%
  - prop. glycol water 40%
  - prop. glycol water 30%
  - LiBr-H2O mixture
  - LiBr-H2O mixture
- External mixtures

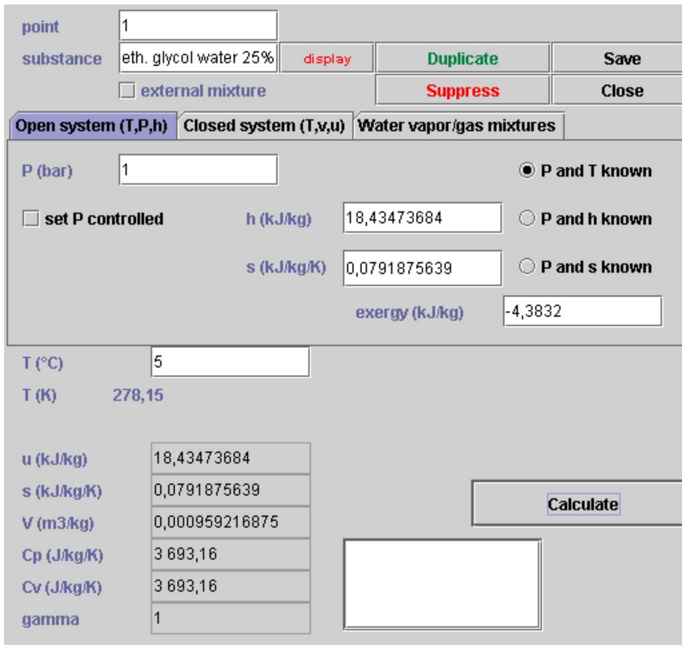
One great advantage of Thermoptim is that its graphic environment can be used to visually build models for a large number of energy systems, from the simplest refrigerator to complex integrated gasification combined cycle electric power plants using several hundred elements.

Not only does this greatly simplify the modeling process and facilitate subsequent use and maintenance of the model, but it also makes the models more reliable. The connections between the different elements are made automatically, thus ensuring consistency.

Through version 1.4, only the components available in the Thermoptim primitive type set could be assembled in this manner, which limited the potential of the software. A number of users wished to be able to define their own elements and/or their own substances.

Thermoptim interface with external classes (Java code elements) provides the solution and facilitates the interoperability of the software with the outside world, especially with other applications developed using Java.

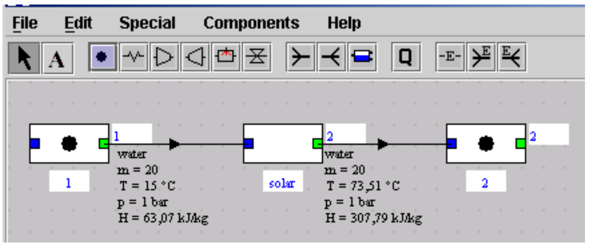
The benefits are two-fold:



point	1
substance	eth. glycol water 25%
external mixture	☐
P (bar)	1
set P controlled	☐
h (kJ/kg)	18,43473684
s (kJ/kg/K)	0,0791875639
exergy (kJ/kg)	-4,3832
P and T known	●
P and h known	○
P and s known	○
T (°C)	5
T (K)	278,15
u (kJ/kg)	18,43473684
s (kJ/kg/K)	0,0791875639
V (m3/kg)	0,000959216875
Cp (J/kg/K)	3 693,16
Cv (J/kg/K)	3 693,16
gamma	1

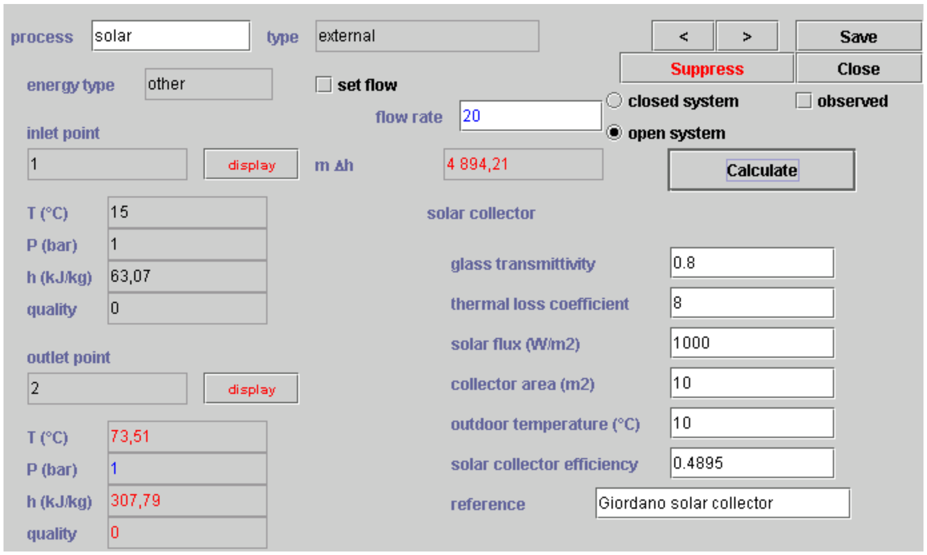
- create Thermoptim extensions from common primitive type set, by adding external modules that define the elements that automatically appear on the screens in a seamless fashion. Thus users can add their own substances or components not available in the basic set.
- drive Thermoptim from another application, either to guide a user (smart tutorial) or to check the code (driver or regulation, access to thermodynamic libraries).

The previous and opposite figures show how the external substances are added to Thermoptim list of substances, and then replace an internal substance on the point screen. They are just as easy to use as if they were part of the software.



The figure opposite shows how an external component representing a solar collector appears in the diagram editor, and the figure below shows the screen of the corresponding process, composed partly of Thermoptim internal code and partly of external code (lower third on the right).

2.1 Software implementation



process	solar
type	external
energy type	other
set flow	☐
flow rate	20
closed system	○
open system	●
observed	☐
inlet point	1
m Δh	4 894,21
T (°C)	15
P (bar)	1
h (kJ/kg)	63,07
quality	0
outlet point	2
T (°C)	73,51
P (bar)	1
h (kJ/kg)	307,79
quality	0

solar collector

glass transmittivity	0.8
thermal loss coefficient	8
solar flux (W/m2)	1000
collector area (m2)	10
outdoor temperature (°C)	10
solar collector efficiency	0.4895
reference	Giordano solar collector

Practically speaking, adding a new external process is quite easy. Simply create a specific class, which inherits from the abstract parent class extThopt.ExtProcess. The interaction with Thermoptim is ensured on two levels:

by general methods for performing the required calculations;

by a JPanel that is built in to the external process screen. Thus, the class designer can create his own graphic interface, which is then inserted into the Thermoptim screen.

2.2 Programming issues

To protect the Thermoptim code, its classes are “obfuscated”, i.e. all of its elements are renamed, making it extremely difficult to decompile. However, this makes it impossible to access the Thermoptim content from the outside. The solution is not to obfuscate the methods that we want to be accessible. Thus there we must find a compromise between accessibility and code protection, which means providing as few accesses as possible. For the moment, only two Thermoptim packages out of five are partially accessible: rg.thopt and rg.corps. The first contains the simulator classes and the second the substance classes.

The solution consists of adding a non-obfuscated package (extThopt), in which there are classes (for example extThopt.CorpsExterne), recognized by Thermoptim, which are subclasses of Thermoptim classes, through interface classes (extThopt.CorpsExterne is a subclass of rg.corps.CorpsExt, which in turn is a subclass of rg.corps.Corps). These classes have methods that establish the correspondence with their obfuscated internal equivalent. We recommend that you refer to the API of the package extThopt to learn about the syntax and functions of all of the methods available. This API is part of the development environment of the external classes available (in the folder api_extThopt).

2.3 Using the external classes

In version 1.5, there are four types of external classes: substances, processes, dividers, and external nodes. The same principle applies to all: in the external package (extThopt), an input class is simply an extension of one of the Thermoptim classes (CorpsExterne for substances, TransfExterne for processes, etc.). All of the external overloaded Thermoptim methods are located in this class. ExtThopt simply acts as the input point in its package. Its role is to instantiate the various classes representing the elements added to Thermoptim and to interface with the software using overloaded methods.

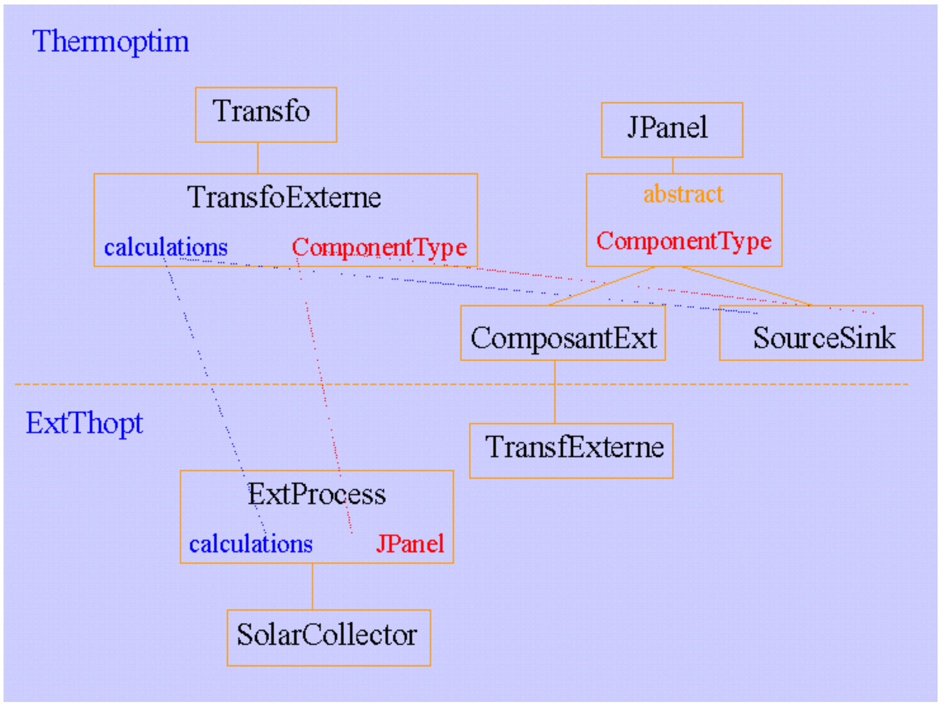
External classes inherit from an abstract parent class. For example, ExtSubstance for substances, ExtProcess for components. In practice, the interaction with Thermoptim is ensured at one or two levels, depending:

- parent classes define all of the general methods useful for performing the required calculations
- with the exception of ExtSubstance, they also define a JPanel that is incorporated into the screen of the external component. In this way, the designer can create his own graphic interface, which then appears on the Thermoptim screen. The designer can define his own graphic elements (buttons, fields, labels, etc.) and process specific actions in his methods. For the components, two methods, saveCompParameters() and readCompParameters(String ligne_data), save the parameter settings in the Thermoptim project file, for use when the project is loaded.

In short, external classes, such as CorpsExterne or TransfExterne, inherit directly from Thermoptim classes and ensure the interface with the software. They call abstract classes, such as ExtSubstance or ExtProcess, whose subclasses may be freely designed by the users via interface methods and JPanel.

The diagram below illustrates the structure used for external processes. For substances, the diagram is the same, but simpler. Since there is no graphic interface, there is no need for the right-hand part of the diagram. People familiar with UML notation will find the class diagrams for external substances and processes in Appendix 5.

The top part of the diagram corresponds to classes present in Thermoptim basic set, and the lower part corresponds to the extThopt package that can be modified by the external class designer. SourceSink is a default implementation of ComponentType to be able to instantiate a TransfoExterne, even in the absence of external classes of this type, and SolarCollector and LiBrAbsorption are examples of external classes. The dotted blue and red lines symbolize the relays of the parts of the code relating to the calculations and the graphic interface, made from TransfoExterne to the other classes, either internal classes (SourceSink) or external classes (subclasses of ExtProcess).



Thermoptim selects directly from the external classes as follows: When it is launched, the software analyzes the archives extThopt2.zip and extUser2.zip, which contain all of the external classes. It finds all of the classes added by default and by the users and sorts them by parent class, loads them in the various types of arrays, and adds the external elements to its own lists so that they are seamlessly accessible. Subsequently, if one of these elements is selected, Thermoptim sends its class to the appropriate constructor, which instantiates it. The line External Class Viewer in the Special menu shows the external classes available (see section 3.6).

Once an external component is instantiated, it is possible to change it by double-clicking on its type as it appears on the screen.

For now, there is no consistency test on the class names and element types, but this feature will be added shortly. For the development of external classes, an emulation system makes it possible to launch Thermoptim from an external application, and to dynamically load the classes under development, providing access to the debugging tools in the usual working environment (see section 6).

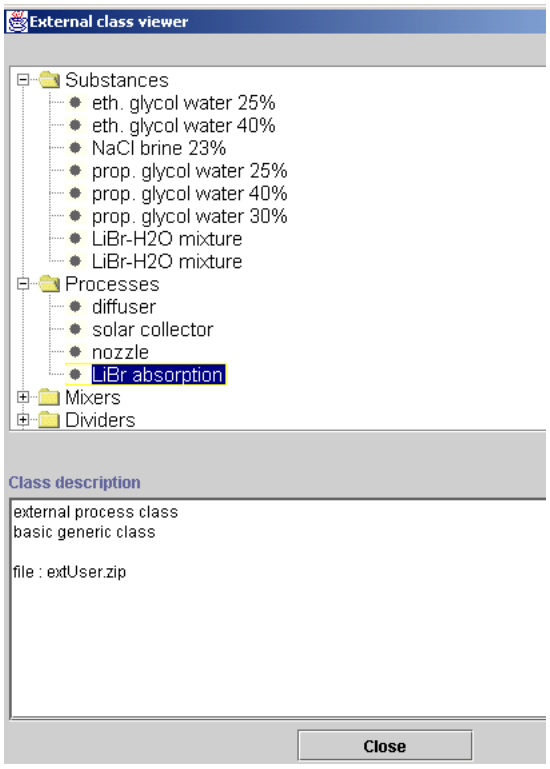
Moreover, the package extThopt includes a Util class that provides a number of utility methods for formatting numbers, reading them onscreen, saving and reading values, finding roots using the bisection method, etc. (see Appendix 3).

2.4 Three categories of interface methods

The classes used in Thermoptim, CorpsExterne, TransfExterne and the other, define the interface methods, which fall into three additional categories:

- Internal Thermoptim methods designed to be obfuscated
- Non-obfuscated internal methods that are directly used by the external classes: These methods are used to execute the Thermoptim methods with external parameter settings, and **must not be overloaded.** Their signature must be strictly respected: in particular, the structure of the Vectors of Objects used as arguments must correspond exactly to what Thermoptim expects, or a ClassCastException will be generated.
- External methods corresponding to Thermoptim obfuscated methods: These methods are used to execute the external methods from within Thermoptim, and **must be overloaded.**

The non-obfuscated methods falling into the last two categories are listed in Appendix 1, for all of the accessible classes. For more information, please see the html documentation for these classes provided in the folder api_Thermoptim.



- Substances
  - eth. glycol water 25%
  - eth. glycol water 40%
  - NaCl brine 23%
  - prop. glycol water 25%
  - prop. glycol water 40%
  - prop. glycol water 30%
  - LiBr-H2O mixture
  - LiBr-H2O mixture
- Processes
  - diffuser
  - solar collector
  - nozzle
  - LiBr absorption
- Mixers
- Dividers

Class description

```
external process class
basic generic class

file : extUser.zip
```

2.5 Using external components

2.5.1 Viewing the available external classes

To help you use and manage the external classes, the line External Class Viewer from the Special menu displays all of the external classes available. They are sorted by type (substances, processes, mixers, dividers, drivers) with a short description of the class selected and where it comes from (extThopt2.zip and extUser2.zip archives as well as classes under development).

This screen can be consulted while you are developing your model.

2.5.2 Representation of an external component in the diagram editor

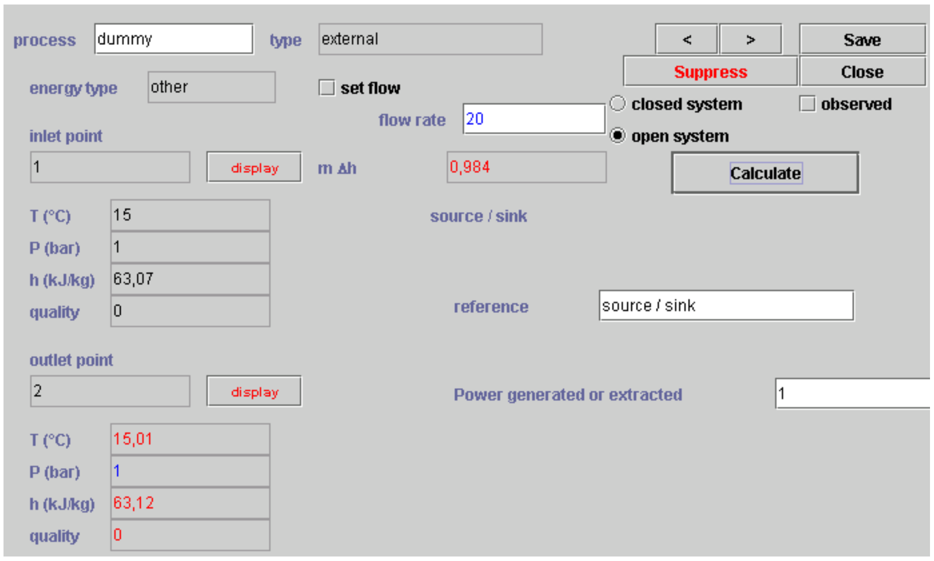
Specific icons were added to represent the external components (

for processes,

for mixers, and

for dividers). The external component is then selected when the simulator is updated from the diagram as indicated below.

2.5.3 Loading an external class



process: dummy type: external

energy type: other set flow flow rate: 20

inlet point: 1 m Δh: 0,984

closed system observed (●) open system

T (°C)	15
P (bar)	1
h (kJ/kg)	63,07
quality	0

source / sink

reference: source / sink

outlet point: 2

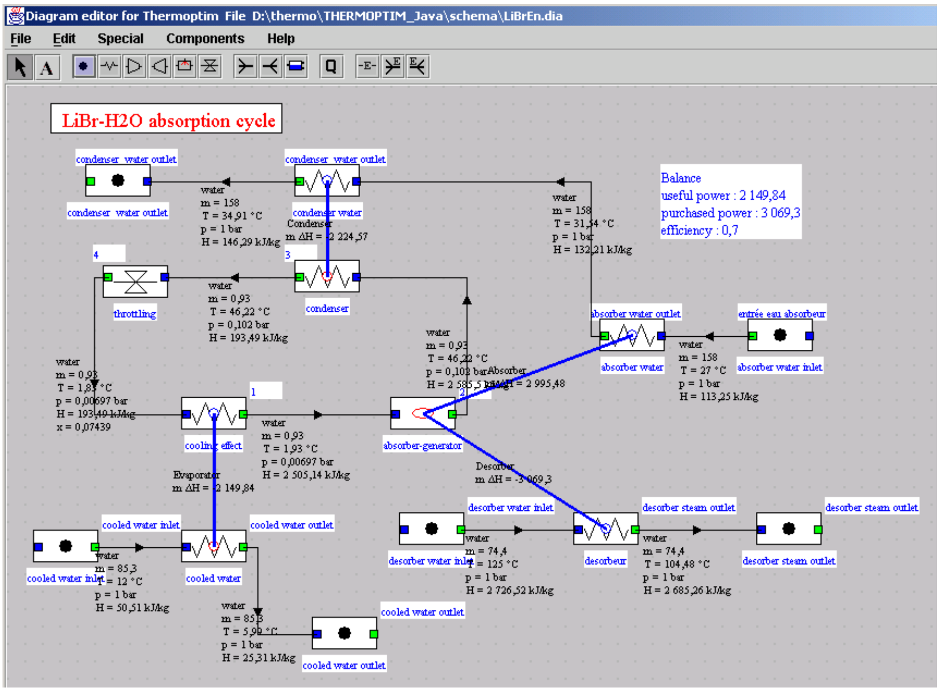
Power generated or extracted: 1

T (°C)	15,01
P (bar)	1
h (kJ/kg)	63,12
quality	0

To load an external process (for an external node, the process is the same), you can either:

- from the simulator screen click on the column header of the process array, then choose External and finally select the type of external process you want from the list;
- Or, from the diagram editor, build the external component graphically then update the simulator from the diagram. In the case of an external process, by default it is a “heat source / sink” type, as shown in the screen opposite.

Once this default process is created, double click on the label “source / sink” to access the list of all external processes available. Choose the one you want and it will be loaded.

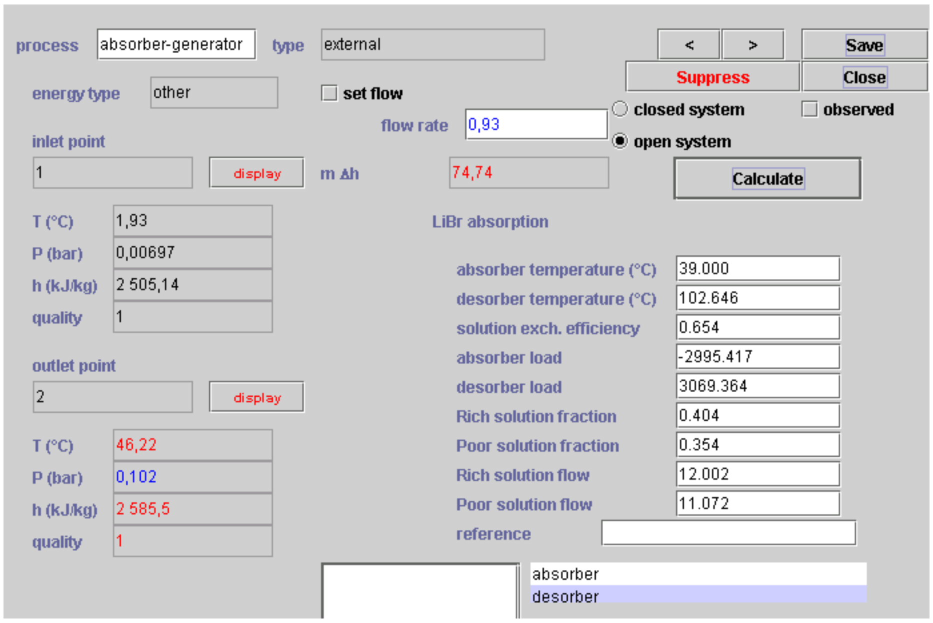


2.5.4 Thermocouplers

The thermocoupler system completes the heat exchanger system by allowing components other than exchange processes to connect to one or more exchange processes to represent a thermal coupling. This system does not encompass the exchanger system: two exchange processes cannot be connected by a thermocoupler.

This system has a number of benefits, because it can be used to represent many thermocouplers that do not constitute a heat exchange in the traditional sense, like for example cooling the walls of the combustion chamber of a reciprocating engine, cooled compression, and above all supply or removal of heat from the multi-functional external components.

The figure above is an illustration of this: An absorption refrigeration cycle, whose absorption-desorption system is defined and integrated in an external process, is supplied with the steam that exits the evaporator then enters the condenser. This cycle involves the mixture LiBr-H2O, whose properties are modeled either directly in the external process, or in an external substance, and requires high temperature heat supply to the desorber and medium temperature heat removal from the absorber. The representation of these thermocouplers is possible thanks to the thermocoupler system: the external process calculates the thermal energies that must be exchanged, and the thermocoupler recalculates the corresponding “exchange” process, which updates its downstream point.



process: absorber-generator
type: external
energy type: other
set flow [ ]
flow rate: 0,93
closed system ○
open system ●
observed [ ]
inlet point: 1
m Δh: 74,74

T (°C)	1,93
P (bar)	0,00697
h (kJ/kg)	2 505,14
quality	1

outlet point: 2

T (°C)	46,22
P (bar)	0,102
h (kJ/kg)	2 585,5
quality	1

LiBr absorption

absorber temperature (°C)	39.000
desorber temperature (°C)	102.646
solution exch. efficiency	0.654
absorber load	-2995.417
desorber load	3069.364
Rich solution fraction	0.404
Poor solution fraction	0.354
Rich solution flow	12.002
Poor solution flow	11.072
reference	

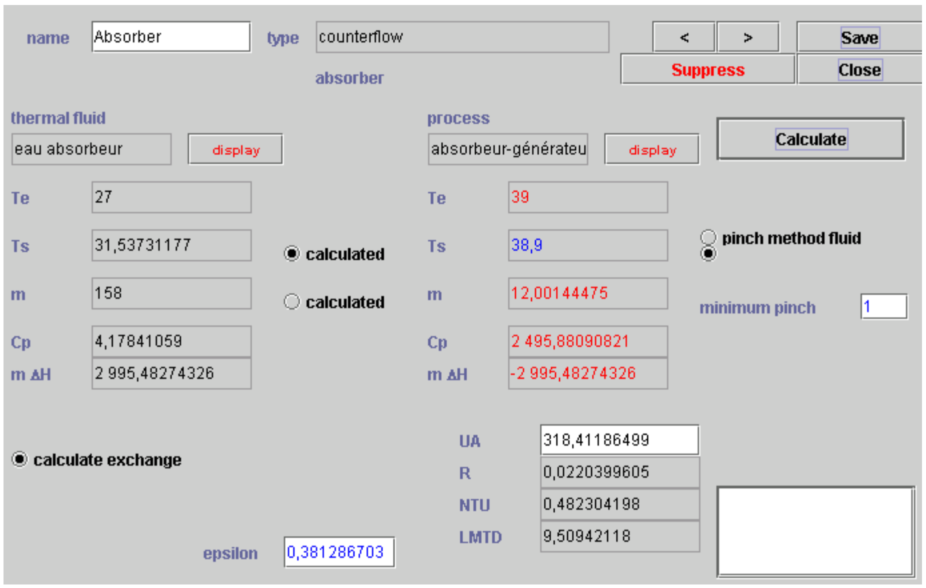
absorber
desorber

The types of thermocouplers used by an external component appear in the lower right hand corner of the screen. Double click on one of the types to open the screen of the corresponding thermocoupler.

Given that thermocouplers are a type of heat exchanger, it is valuable to define them by values such as effectiveness ε, UA, NTU or LMTD, that can be calculated using similar equations. The component sends to each of its thermocouplers the equivalent values for flow rates, inlet and outlet temperature and thermal energy transferred, which they must take into account in their calculations. Specific methods are provided in the external class code and are not user-modifiable.

However, there are limits to the similarities with exchangers: for example, temperature crossovers unacceptable in an exchanger may occur in a thermocoupler, leading to absurd values.

So it is best to transmit values that are unlikely to lead to this type of situation. One possible solution is to assume that the thermocoupler is isothermal for calculations of characteristics that are similar to exchanger characteristics. For example, a combustion chamber may be assumed to be at mean temperature between its upstream and downstream points when calculating cooling. This assumption is not absurd and may prevent a temperature crossover between the cooling fluid and the gases that cross the component.



name: Absorber | type: counterflow | absorber

thermal fluid: eau absorbeur | process: absorbeur-générateu

	thermal fluid		process
Te	27	Te	39
Ts	31,53731177	Ts	38,9
m	158	m	12,00144475
Cp	4,17841059	Cp	2 495,88090821
m ΔH	2 995,48274326	m ΔH	-2 995,48274326

● calculated (Ts)
○ calculated (m)
○ pinch method fluid ●
minimum pinch: 1

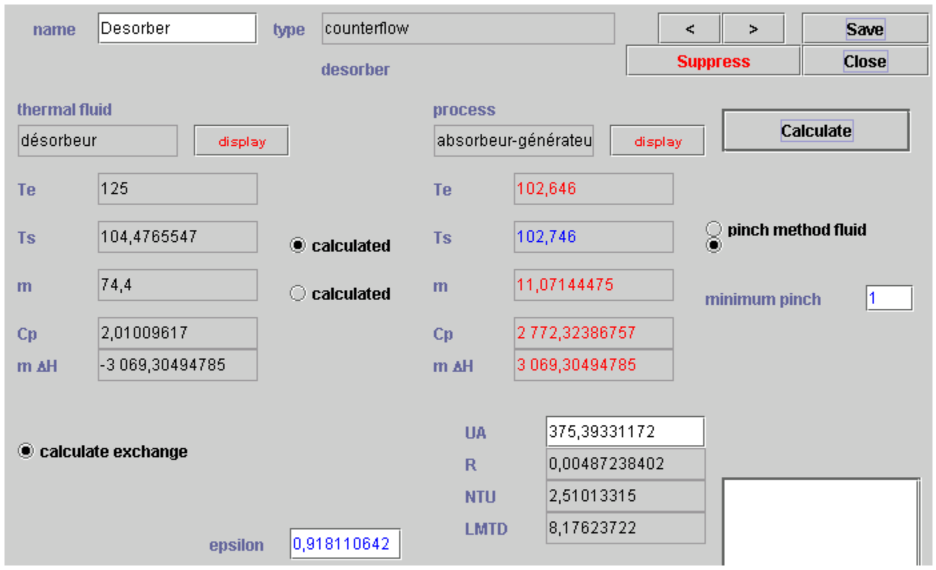
● calculate exchange
epsilon: 0,381286703

UA	318,41186499
R	0,0220399605
NTU	0,482304198
LMTD	9,50942118

In the case of the absorption machine presented above, we assumed that the absorber and desorber were isothermal.

Both lead to the screens below. If we had not taken the temperature of the absorber as a reference for the exchange calculations, keeping the temperatures of the steam entering and exiting the external process, we would have ended up with a temperature crossover.

For external processes that accept several thermocouplers and for external nodes, the potential complexity of the calculations prevents the exchange process from driving the thermocoupler. Its load is always set by the external component. This is why there are fewer options for calculating a thermocoupler than for a heat exchanger: The user can only choose between calculating the outlet temperature of the exchange process (at a given flow rate) and the flow rate, when the temperature is known.



name: Desorber | type: counterflow | desorber

thermal fluid: désorbeur | process: absorbeur-générateu

	thermal fluid		process
Te	125	Te	102,646
Ts	104,4765547	Ts	102,746
m	74,4	m	11,07144475
Cp	2,01009617	Cp	2 772,32386757
m ΔH	-3 069,30494785	m ΔH	3 069,30494785

● calculated (Ts)
○ calculated (m)
○ pinch method fluid ●
minimum pinch: 1

● calculate exchange
epsilon: 0,918110642

UA	375,39331172
R	0,00487238402
NTU	2,51013315
LMTD	8,17623722

Note that on the thermocoupler screen, the external component fluid can be selected as a pinch fluid and a minimum pinch value can be entered (see optimization method, volume 1).

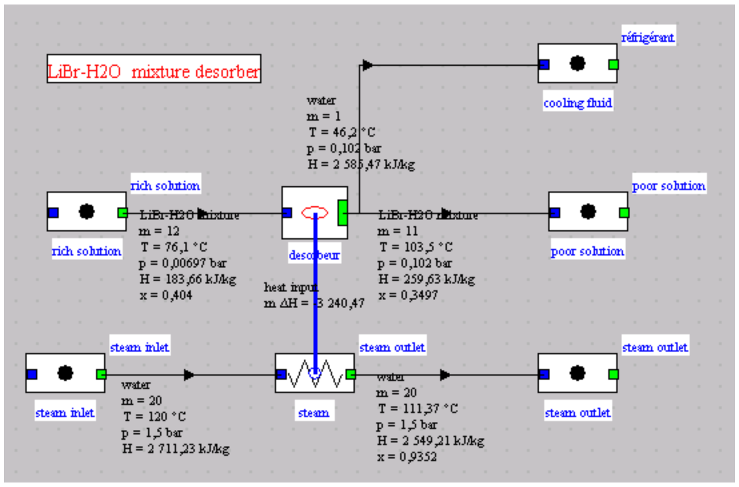
2.5.5 External Nodes

The nodes of Thermoptim basic set are extremely simple components used to complete the description of the fluid circuits. They are always adiabatic, and they ensure the conservation of the mass flow rate and enthalpy

However, there is one somewhat special component, considered as a process, but which is in fact a node in most cases: combustion, which receives an oxidizer and a fuel, and from which burned gases exit.

In the LiBr-H2O absorption example, a number of energy systems are involved, as well as components of varying complexity, which can have a number of input and outlet fluids, after various internal calculations, with or without thermocoupling with external heat sources.

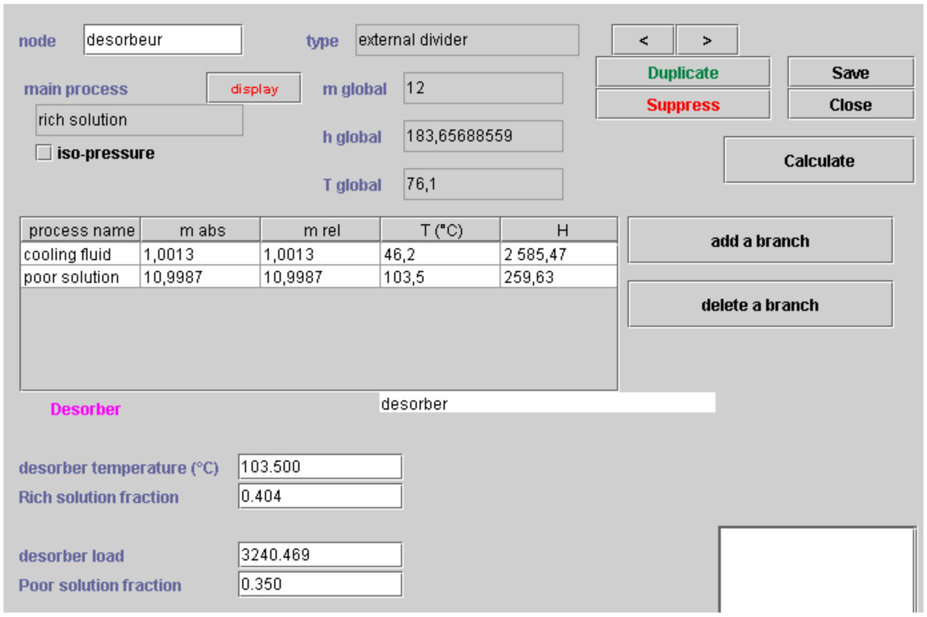
External nodes allow a user to define these components. Only external mixers and dividers are defined: no component simultaneously performs both functions (receiving and emitting several fluids), but you need only couple an external mixer with an external diffuser to do so.



In many respects, we encounter the same problems in implementing these external nodes as with thermocouplers: the potential complexity of the calculations to be made in the node makes it necessary for the node to take over and control both the main vein and the branches, since no default calculation is possible.

The verification and consistency problems are even more critical than for thermocouplers: only the node designer knows which processes it must be connected to in inlet and outlet. The user must refer to the documentation of the class to know how to use it.

The figure above shows the diagram of a desorber for an absorption machine using the LiBr-H2O mixture whose properties must in this case be provided by an external substance.



node: desorbeur

type: external divider

main process: rich solution

iso-pressure [ ]

m global: 12

h global: 183,65688559

T global: 76,1

process name	m abs	m rel	T (°C)	H
cooling fluid	1,0013	1,0013	46,2	2 585,47
poor solution	10,9987	10,9987	103,5	259,63

Desorber: desorber

desorber temperature (°C): 103.500

Rich solution fraction: 0.404

desorber load: 3240.469

Poor solution fraction: 0.350

The external node screen is shown opposite. As with external processes, it contains a general part defined in the Thermoptim basic set, completed by a part defined by the user (here the lower left zone).

In the model shown here, the only parameter defined in the node is the temperature of the desorber. The properties of the rich solution (mass fraction and flow rate) and the state of the refrigerant are obtained from their respective processes.

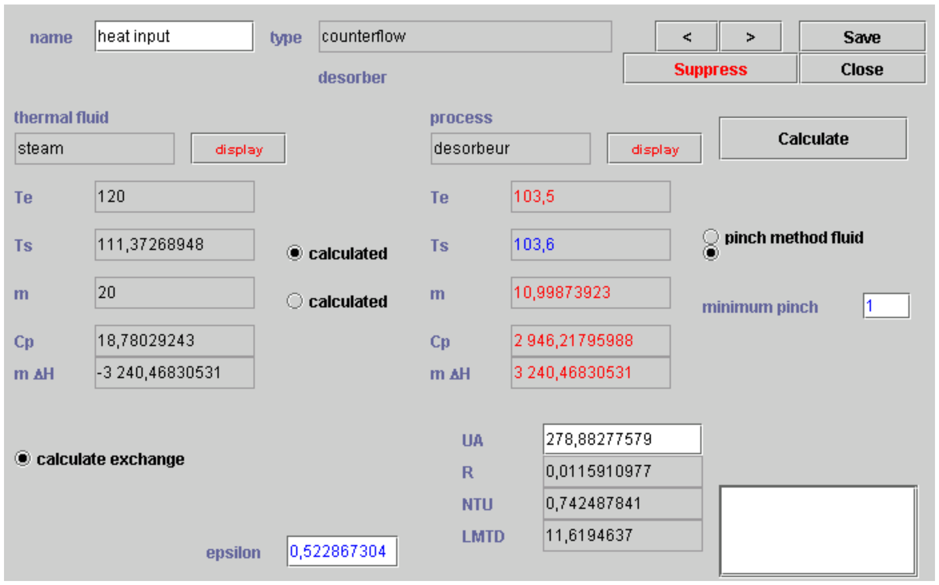
The node calculates the flow rates of the refrigerant and the poor solution, its state, and heat supply required.

The thermocoupler then calculates the final state of the steam used.

Before each recalculation, the node checks that its structure is correct and loads the values that it needs to be calculated. Then it updates the thermocoupler that can in turn be recalculated. In this example, we assumed that the desorber was isothermal, and we took the flow rate of the poor solution as the reference flow rate.

2.5.6 Miscellaneous comments

You may have noticed that the screens of the external processes represented here are composed partly of Thermoptim internal code and partly of external code (lower right section in the processes, lower left in the nodes.). This is because the character strings used in the external components have not been “internationalized” the way the Thermoptim basic set has.



name: heat input type: counterflow desorber

< > Save Suppress Close

thermal fluid: steam (display) process: desorbeur (display) Calculate

	thermal fluid			process
Te	120		Te	103,5
Ts	111,37268948	[✓ calculated]	Ts	103,6
m	20	[calculated]	m	10,99873923
Cp	18,78029243		Cp	2 946,21795988
m ΔH	-3 240,46830531		m ΔH	3 240,46830531

[pinch method fluid] [✓]

minimum pinch: 1

[✓ calculate exchange]

epsilon: 0,522867304

UA	278,88277579
R	0,0115910977
NTU	0,742487841
LMTD	11,6194637

But this does not affect their use in any way. However, you can see that the number display has not been internationalized either, so the decimal separator is not the same: it is a period and not a comma. Of course, this can be modified, but it has not yet been done. This means that the figures must be entered with a point in the part defined in the external component, and with a comma in the rest of the screen. It is important to pay close attention to this issue, otherwise number formatting errors will be detected by Java.

2.6 Representation of real fluid mixtures in Thermoptim

Until 2005, the only Thermoptim substances whose composition could be defined by the user were compound gases. It was impossible to represent real fluid mixtures. The external substance system was reworked to be able to do so, via the introduction of a new type of substance, called “**external mixture**”, and substances formerly known as external substances were renamed “**pure external substances**”.

An external mixture is made from a **system**, i.e. from a given set of **pure substances,** and its composition is specified in a new editor.

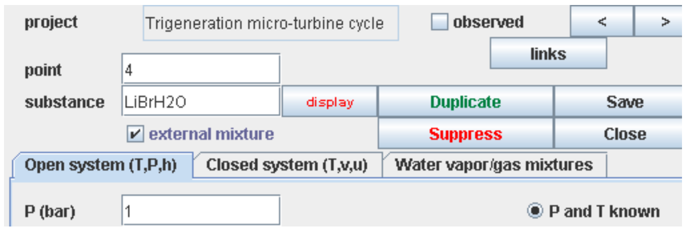
The distinctive characteristic of these external mixtures is that they can be used to generate a set of substances from the same **system** of pure substances. In this way, they are similar to compound gases. The difference is that the interactions among real fluids are much more complex than among ideal gases, so you must specify not only the pure substances used, but also their models, mixture rules and a number of additional parameters.

2.6.1 Link between Thermoptim and thermodynamic properties servers

The system makes it possible not only to define external mixtures by completely writing their external classes, but also to calculate those using **thermodynamic properties servers (TPS)**, software packages that have specialized libraries, such as TEP ThermoSoft developed by the Centre Energétique et Procédés of Ecole des Mines de Paris and Simulis Thermodynamics made by ProSim.

This link is a very valuable extension of Thermoptim, since it means that many new fluids can be used in the software, with extremely accurate models. The trade-off is that it is somewhat complex and especially, calculation times can be long.

2.6.2 Creating an external mixture

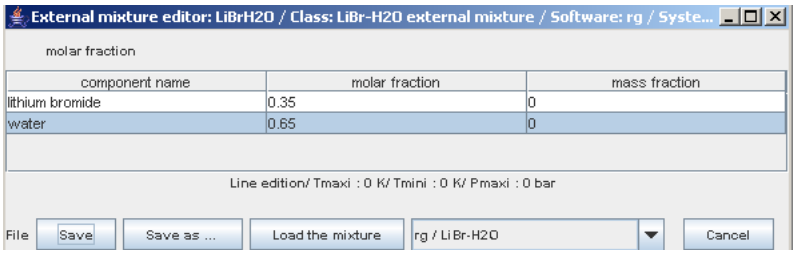


project: Trigeneration micro-turbine cycle
point: 4
substance: LiBrH2O
[✓ external mixture]
P (bar): 1
[● P and T known]

The point calculation screen has been modified slightly: an “external mixture” option has been added.

To create a new external mixture, select this option then enter a new substance name and press Enter.

The external mixture editor can then be opened by clicking on “display”. By default, the external mixture created is of the type "LiBr-H2O external mixture" whose class is presented as an example in the programming documentation. If this is the desired substance, enter its composition, then click “Save”. The expected composition type (molar or mass fraction) is displayed above the component names column, and corresponds to the first column of figures from the left.



External mixture editor: LiBrH2O / Class: LiBr-H2O external mixture / Software: rg / Syste...

molar fraction

component name	molar fraction	mass fraction
lithium bromide	0.35	0
water	0.65	0

Line edition/ Tmaxi : 0 K/ Tmini : 0 K/ Pmaxi : 0 bar

rg / LiBr-H2O

In the title of the external mixture editor the following items appear:

- the mixture name (here LiBrH2O)
- the corresponding external class identifier (here LiBr-H2O external mixture)
- the name of the software used (either a TPS, or as in this example (rg) a family of external classes).
- the system used (here LiBr-H2O)

This information is used to characterize the mixture by referring to the documentation. The same TPS may propose several systems, and may be used in several external classes.

From the pop-up menu located in the lower right of the editor screen you can display all of the systems available. If the one proposed by default is not the one you want, select one from the list and click “load the mixture”. The editor is then updated, and you can enter the composition and click “save”.

In a Thermoptim project file (.prj), the data for external mixtures are saved in the same way as for compound gases.

COMPOUND SUBSTANCES 2

Name of gas / Components molar fraction mass fraction

gaz_brulés 5

CO2 0.0309766336 0.0477375785

H2O 0.0619532671 0.0390824699

O2 0.14154164 0.158596897

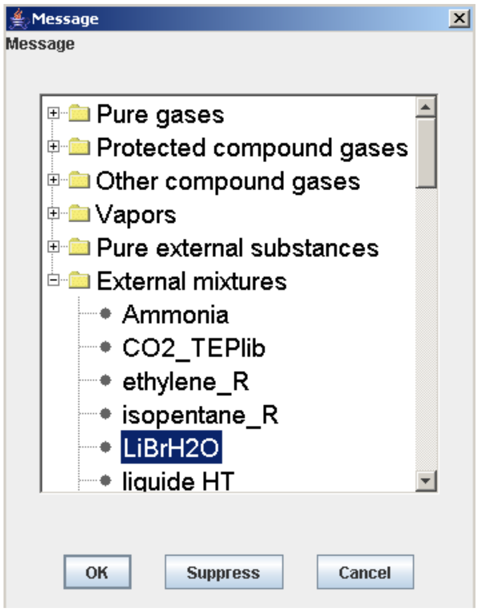
N2 0.756807249 0.74238276

Ar 0.0087212103 0.0122002945

LiBrH2O 2 extMixture=true classe=LiBr-H2O external mixture syst=LiBr-H2O

lithium bromide 0.35 0

water 0.65 0



extMixture=true indicates that it is an external mixture, and the rest of the line specifies the class and the system selected. The next lines give the molar or mass compositions (for the time being, for a given mixture, the input can be done only in the manner selected by the designer of the external class, either in molar fractions, or in mass fractions).

Note that the order in which the components appears is not random: it is defined in the corresponding external class. Consequently, if you edit a project file manually, you must keep the same order for the system components.

The substance selection screen has been modified to distinguish between two categories of external substances. In this example, the three external mixtures on the bottom of the list correspond to different compositions of the same system.

Once the external mixture has been defined or loaded from the substance selection screen, its composition can be modified in the editor, and as we saw above, the system can also be changed by selecting one from the available libraries (pop-up menu on the lower right).

3 Programming external components

This section presents the basic notions needed to program external components, with some illustrations taken from the examples provided with the development environment. Once again, however, we would like to stress how important it is to properly prepare the documentation of your classes. The documentation must include at least two sections: the programming file and the user documentation. A poorly documented class will be difficult to use or modify, and the easiest time to document an external class is when it is created. Unfortunately experience has shown that this is only rarely the case…

Since the external class represents a component model that has a physical sense, it is important to write a note describing the model, both for the programming file and the user documentation. The code itself must include enough comments to make the links to the model as clear as possible (in particular it is best if the notations are uniform). Finally, the user documentation must specifically indicate the component constraints and requirements, with a thorough description of which inlet and outlet fluids can be connected to it, (name, nature, etc.) and how the thermocouplers, if any, must be constructed. The method getClassDescription () can be used to enter a brief description of the class, which can be consulted from a special Thermoptim screen. We recommend including the documentation references and the main constraints and requirements of the class in this method.

Practically speaking, you can proceed as follows: gather the following elements in an archive that has the same name as the class: the java code of the component, its class file, a modeling note, instructions for use with a short example of how it is used, and the .prj and .dia files for the example. A model library containing all of these archives can easily be created and updated.

3.1 External Substances

3.1.1 Construction of pure external substances

The structure of external classes was defined above. The class extThopt.CorpsExterne inherits from the rg.corps.CorpsExt class of Thermoptim and performs the interface, relaying the calculations to an abstract class (extThopt.ExtSubstance). This class defines the basic methods and is subclassed by the different classes introduced (for example extThopt.GlycolWater25, extThopt.GlycolWater40, etc.).é We recommend that you refer to the API of extThopt.CorpsExterne and extThopt.ExtSubstance to learn about the syntax and functions of all of the methods available. This API is part of the development environment of the external classes available.

Since Thermoptim knows only the external interface class extThopt.CorpsExterne and the abstract class extThopt.ExtSubstance, the instantiations can use only these two classes. The synchronization between the internal and external methods must be done carefully, but it is the only way to make sure that the classes used are working properly.

We will begin by explaining the construction procedure implemented in Thermoptim, then we will show how to create a new external class, by subclassing extThopt.ExtSubstance.

3.1.1.1 Construction of a pure external substance in Thermoptim

The construction procedure is as follows:

1. the user selects an external substance from the list, which shows the index in the arrays of external classes loaded in Thermoptim.
2. you load this class, which you transtype in ExtSubstance, and encapsulate in an Object.

```
Class c=(Class)rg.util.Util.substClasses[i];//load the class of the external substance
Constructor ct=c.getConstructor(null);//carefully instantiate it with its constructor without an argument.
extThopt.ExtSubstance ec=(extThopt.ExtSubstance)ct.newInstance(null);
Object ob=ec;//encapsulate it in an Object (otherwise the argument will not go through!)
corps=(Corps)new CorpsExterne(ob);//instantiate the external substance
```

which is done by the following constructor:

```
public CorpsExterne(Object obj){
subst=(ExtSubstance)obj;
setNom(subst.getType());
setComment(subst.chemForm);
initCorpsExt(subst.M, subst.PC, subst.TC, subst.VC, subst.Tmini, subst.Tmaxi,
subst.Pmini, subst.Pmaxi, subst.typeCorps);
}
```

3.1.1.2 Creating a pure external substance

To create an external substance, simply subclass extThopt.ExtSubstance. Let us take the example of the class DowThermA.

The constructor is as follows:

```
public DowThermA (){
super();
type=getType();//type of substance
M=166; PC=31.34; TC=752.15; //initializations (molar mass, critical pressure and temperatures)
Tmini=293.15; Tmaxi=523.15; //minimum and maximum temperatures of the substance definition
chemForm="(C6H5)2O (73.5% vol), (C6H5)2 (26.5% vol)"; //chemical composition of the substance
}
```

The method getType() sends the type of substance as it appears on the Thermoptim screens:

```
public String getType(){
return "Dowtherm A";
}
```

The method getClassDescription () can be used to enter a brief description of the class, which will appear in the external class viewer (see section 3.6). Provide a short explanation of the model, and if possible cross-reference to more detailed documentation.

```
public String getClassDescription(){
return "data from FLUIDFILE software by Dow Chemical\n\nauthor : R. Gicquel April 2003";
}
```

The methods for calculating the properties of the substance are presented later in the manual.

3.1.2 Calculations on the pure external substances

3.1.2.1 Principles of correspondence

For the calculations, the arguments are passed by the non-obfuscated methods. Please see Appendix 1 for details on existing methods.

For example, the enthalpy equation is inverted to solve for T in extThopt.CorpsExterne by:

```
public double getT_from_hP(double hv,double P){
return subst.getT_from_hP(hv,P);
}
```

The interface class rg.corps.CorpsExt ensures the correspondence between inv_hp_T (obfuscated) and getT_from_hP (non-obfuscated) :

```
public double inv_hp_T(double hv,double P){
return getT_from_hP(hv, P);
}
```

Thus, any time inv_hp_T by is called by a method internal to Thermoptim for an external substance, the call is relayed to getT_from_hP by rg.corps.CorpsExt.

Going in the other direction, rg.corps.Corps or rg.corps.CorpsExt contains the following method:

```
public double getT_from_hP(double hv,double P){
return inv_hp_T(hv, P);
}
```

When getT_from_hP is called by an external method for an internal substance, the call is relayed to inv_hp_T.

3.1.2.2 Examples of implementation

The class DowThermA defines a heat transfer fluid used in the exchangers. The substance remains in the liquid state at all times, and the calculations are simple: For example enthalpy, which is a function only of temperature (and its inversion).

```
public double calcH (double T, double P,double x) {
double a=0.00281914712153519,b=1.51937611940298;
double TT=T-273.15;
return b*TT+a/2.*TT*TT;
}
```

because h is given by a second degree polynomial to solve for T, its inversion is explicit:

```
public double getT_from_hP(double $hv,double $P){
double a=0.00281914712153519,b=1.51937611940298;
double c=-$hv;
a=a/2.;
double delta=b*b-4.*a*c;
double TT = (-b+Math.pow(delta,0.5))/2./a;
return TT+273.15;
}
```

The calculations here are very simple. If we were dealing with a condensable vapor, they would be much more complex. The class LiBrH2Omixture is a somewhat more complicated example.

3.1.3 Constructing external mixtures

3.1.3.1 Special issues

Like pure external substances, external mixtures are extThopt. ExtSubstance type, even if their calculations are performed in the TPS. To differentiate them, external mixtures must implement the interface extThopt.ExternalMixture, which specifies which methods they must define (see below).

The names of the pure external substances are listed on the substance selection screen, where they are used to instantiate classes.

For external mixtures, things are much more complex, because the same external class can define several systems, and each system generates as many compositions as the user wants, except of course if you are dealing with a pure substance, in which case the molar fraction and the mass fraction are both 1. Each external class defines all of the systems it proposes, and these systems cannot be modified by the user from within Thermoptim.

The class rg.corps.ExtMixture is used to instantiate the mixtures. It has a Substance called refExternalSubstance, which corresponds to the external class where the calculations are done, and with which it exchanges the system name and the selected compositions, using the method CorpsExterne (which must not be obfuscated, of course):

```
public void CalcPropCorps (double T, double p, double x, String systType, double[] fract_mol) {
//we start by updating the system composition
subst.updateComp(systType, fract_mol);
double U=20., H=10., S=0.5, V=0.01, Cv=0., Xh=0.;
if((T>=subst.Tmini)&&(T<=subst.Tmaxi)){
//we call the method CalcPropCorps of the external mixture
double[]val=subst.CalcPropCorps (T, p, x);
//we load the values calculated
H=val[0];
V=val[1];
Cv=val[2];
x=val[3];
U=val[4];
S=val[5];
}
else{
```

```
String message=resIntTh_fr.getString("mess_116");
JOptionPane.showMessageDialog(new JFrame(),message);
}
//update of the substance variables (hidden)
setState(p, T, x,//in arguments
U, H, S, V, Cv, Xh);//to be recalculated
}
```

This method relays the system selected (systType) and its molar or mass fraction (fract_mol) to the external substance (subst), then calculates the point.

In order to return the value of the vapor quality or the composition at liquid-vapor equilibrium, we defined the method getQuality(), which returns an array of doubles corresponding to the variable X of ExtSubstance, which must be correctly updated when inversion calculations are performed.

Any external mixture must return three methods defined in the interface extThopt.ExternalMixture, used to initialize ExtMixture when it is instantiated:

- public String getSoftware(), which defines the thermodynamic properties server used
- public Vector getMixtures(), which defines the names of the different systems, and the substances they contain
- public boolean isMolarFraction(), which is true if the composition must be entered in molar variables, and otherwise it is false.

The structure of the Vector vMixtures is as follows:

```
String[] system={"lithium bromide","water"};
vMixtures= new Vector();
Object[]obj=new Object[2];
obj[0]="LiBr-H2O";
obj[1]= system;
vMixtures.addElement(obj);
```

Note that isMolarFraction() serves only to define the mixture editor display: In all cases the composition transits via the array fract_mol.

3.1.3.2 Critical or pseudo-critical values: getCriticalParameters() method

Critical values of a mixture depend on its composition and cannot be initialized once and for all. To transfer them in Thermoptim the external mixture uses getCriticalParameters(), whose implementation for CTPLib (actually pseudo-critical values) is given by:

```
public double [] getCriticalParameters () {
double [] props = new double [4];
if (nbComponents == 1) {
props [0] = compProp [0] .TC;
props [1] = compProp [0] .PC / 1.e5;
props [2] = compProp [0] .VC;
props [3] = compProp [0] .M;
}dropoff window
else {
props [0] = CalcTcMix ();
props [1] = CalcPcMix () / 1.e5;
props [2] = CalcVcMix ();
props [3] = CalcMMix ();
```

}dropoff window

return props;

}

The values are transferred when the substance is calculated.

For an external mixture, the calculation of the state of the substance from Thermoptim is relayed to void rg.corps.ExtMixture.etat_complet (double TT, double pp, double x), which calculates from the class by:

refExternalSubstance.CalcPropCorps (TT, p, x, fractType, systType, fract_mol); // modRG 23/01

and then retrieves the values v = refExternalSubstance.getSubstProperties Vector ();

and updates all values, including the critical coordinates

When calculating refExternalSubstance.CalcPropCorps, critical values are determined by:

double [] getCriticalParameters prop = ();

subst.TC prop = [0];

subst.PC prop = [1];

subst.VC = prop [2];

subst.M = prop [3];

getCriticalParameters()returns the correct values.

3.1.3.3 Files associated with external mixtures

As in the case of compound gases, the list of external mixtures is saved in a file called "mel_ext.txt", which is updated each time a new mixture is created, and is used to initialize the substance selection screen.

external mixtures file

CO2_TEP TEPThermoSoft mixtures CO2 CO2;1.0;0.0

liqu pauvre TEPThermoSoft mixtures NH3-H2O ammonia;0.25;0.0 water;0.75;0.0

vap riche TEPThermoSoft mixtures NH3-H2O ammonia;0.75;0.0 water;0.25;0.0

mel pauvre TEPThermoSoft mixtures NH3-H2O ammonia;0.1;0.0 water;0.9;0.0

LiBrH2O LiBr-H2O external mixture LiBr-H2O lithium bromide;0.35;0.0 water;0.65;0.0

Fin

Fin

3.1.3.3 Example of external mixture: the system LiBr-H2O

The system provides a simple example of how to define an external mixture. Later in the manual we will give a more complex example on linking up with the Thermosoft TEP thermodynamic properties server, taking into account the characteristics of the ThermoSoft database. Another example is the coupling with the TEP Lib SPT which is detailed on the **Thermoptim UNIT portal**. As the latter is written in Java, the coupling is particularly simple.

Warning: this system is modeled in Thermoptim external classes in two ways: by the H2OLiBrMixture class as an external mixture, and by the LiBrH2OMixture class as a simple external substance. In the latter case, the mass fraction of LiBr X in the mixture uses the field usually devoted to the quality x.

In machines using the LiBr-H2O mixture, the difference in the vapor pressure between the solvent (LiBr) and the solute (H2O) is such that we can set aside the mass fraction of the solvent in the vapor phase, which simplifies the calculations. Note that it is common practice to define the parameters of the diagram of the LiBr-H2O mixture as a function of the mass concentration in solvent (LiBr) and not in solute. Since water is likely to crystallize at low temperatures, the mixture crystallization curve is often shown on the diagram, corresponding to a lower operating limit for the machines.

For this pair, the ASHRAE proposes equations (1) and (2), set up by generalizing the refrigerant (water) saturation pressure law to the mixture, in which the water temperature t' (°C) is replaced by a linear function of the solution temperature t (°C). P, expressed based on a decimal logarithm, is the pressure in kPa, and X is the mass fraction of the mixture in LiBr. These equations are valid in the following value intervals: $-15 < t' < 110$ °C, $5 < t < 175$ °C, $45 < X < 70$ %.

ASHRAE, Fundamentals Handbook (SI), Thermophysical properties of refrigerants, 2001

$$log(P) = C + \frac{D}{t' + 273.15} + \frac{E}{(t' + 273.15)^2} \qquad (1)$$

$$t' = \frac{t - \sum_{i=0}^{3} B_i X^i}{\sum_{i=0}^{3} A_i X^i} \qquad (2)$$

Table 1 coefficients of equations 1 and 2

ASHRAE	0	1	2	3	4
A0	-2.00755	B0	124.937	C	7.05
A1	0.16976	B1	-7.71649	D	-1596.49
A2	-0.00313	B2	0.152286	E	-104095.5
A3	0.0000198	B3	-0.000795		

The constructor initializes the validity limits of the class and the Vector vMixtures, which in this case includes just one system:

```
public H2OLiBrMixture (){
super();
type=getType();
M=29;PC=10;TC=350; //Attention: initialisations without physical sense
Tmini=278.15; Tmaxi=448.15;
chemForm="LiBr-H2O mixture";
typeCorps=6;//external susbstance type isMixture
vMixtures= new Vector();
Object[]obj=new Object[2];
obj[0]="LiBr-H2O";
obj[1]=system;
vMixtures.addElement(obj);
}
String[] system={"lithium bromide","water"};
```

The following methods define the class description, the software used (since it is not a TPS, the name is not important in this case), the unit to be used to express the composition (in this case mass), the system proposed, and the identifier of the class.

```
public String getClassDescription(){
return "external mixture class\nWatch out! the LiBr composition is to be expressed as mass fractions";
}
public String getSoftware(){
return "rg";
}
public boolean isMolarFraction(){
return false;
}
public Vector getMixtures(){
return vMixtures;
}
public String getType(){
return "LiBr-H2O external mixture";
}
```

The composition is updated based on the following method, which uses the intermediate variable x, equal to X/100.

```
public void updateComp(String systType, double[] fract_mass){
//attention: x represents the mass fraction, even if it goes through fract_mol in ExtMixture
selectedSyst=systType;
x=fract_mass[0];
}
```

The other methods (not presented here) define the calculations to be performed to solve equations (1) and (2).

3.1.4 Instantiation from external classes

It is possible to directly instantiate an external mixture from the external classes, for example in a driver. The syntax is provided below. You simply have to indicate the system selected (here "LiBr-H2O"), and update the composition (particularly simple in this case). The calculation and inversion functions can then be used directly, and the results can be written in the output file, for example.

```
H2OLiBrMixture monCorps=new H2OLiBrMixture ();//instantiation of the substance
double[]fractmass={0.35,0.65};
monCorps.updateComp("LiBr-H2O",fractmass);
System.out.println("Enthalpie du mélange externe LiBr-H2O pour la composition massique : "+fractmass[0]+" LiBr");
for(int i=0;i<10;i++){
double T=280+10*i;
double[] val=monCorps.CalcPropCorps(T, 5, 0);
System.out.println("T :\t"+T+" h : \t"+val[0]);
}
```

which gives:

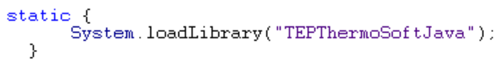
```
Enthalpy of the external misture LiBr-H2O for the mass composition: 0.35 LiBr
T : 280.0 h : -5.2873148226108775
T : 290.0 h : 21.108948694345447
T : 300.0 h : 47.56170686221427
T : 310.0 h : 74.07095968099561
T : 320.0 h : 100.63670715068943
T : 330.0 h : 127.25894927129576
T : 340.0 h : 153.9376860428146
T : 350.0 h : 180.67291746524594
T : 360.0 h : 207.46464353858977
T : 370.0 h : 234.3128642628461
```

3.1.5 Coupling class with the properties server TEP ThermoSoft

Coupling with a thermodynamic properties server (TPS) such as TEP ThermoSoft is much more complex, on the one hand because this TPS is composed of calculation modules that are separate from Thermoptim, and what’s more they are developed in another language, and on the other hand because the calculable thermodynamic systems can vary greatly from case to case. We will illustrate the procedure using the example of the class TEPThermoSoft.java.

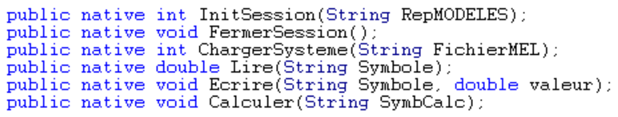
3.1.5.1 Interface between Java and Delphi

The following method is used to load the library "TEPThermoSoftJava.dll" which constitutes the interface with the Delphi environment in which TEP ThermoSoft was developed:



```
static {
      System.loadLibrary("TEPThermoSoftJava");
  }
```

Next you must declare the Delphi methods callable from Java that serve to dialog between the two environments (please refer to Appendix 1 for details on their function and syntax)



```
public native int InitSession(String RepMODELES);
public native void FermerSession();
public native int ChargerSysteme(String FichierMEL);
public native double Lire(String Symbole);
public native void Ecrire(String Symbole, double valeur);
public native void Calculer(String SymbCalc);
```

3.1.5.2 Definition of mixtures proposed by the TPS

Each TPS should define a list of the systems proposed, which is placed in the “mixtures” folder of the installation directory. The standard format is as follows:

External mixture file

NH3-H2O NH3-H2O.mel 2 ammonia water M=44 PC=200 TC=404.128 Tmaxi=1100 Tmini=216.7 T0=293.15 P0=1.

CO2 CO2.mel 1 CO2 M=44 PC=73.77 TC=304.128 Tmaxi=1100 Tmini=216.7 T0=273.15 P0=36.59027

Fin

Fin

Each line contains:

- the name of the system as it appears on the Thermoptim selection screens (for example NH3-H2O)
- the mixture definition file, in the format of the properties server (for example NH3-H2O.mel)
- the number of components
- the name of the components as they appear on the Thermoptim selection screens
- the molar mass, the critical temperature and pressure, maximum and minimum temperatures for the fluid, and the reference temperature and pressure values (for which h=u=s=0)

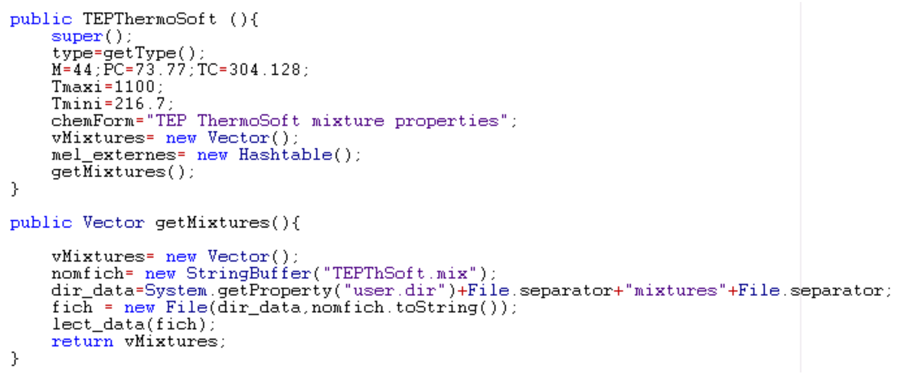
It is basically via the mixture definition file that the properties server is initialized. Thermoptim simply modifies the composition and the state variables.

Note that this list contains the system “CO2” with just one component, i.e., a pure substance. The external mixture mechanism also makes it possible to emulate a TPS to represent a pure substance by a more precise model than the one used in Thermoptim. In this specific case, it is a dedicated equation that can be used in the neighborhood of the critical point.

Wagner, R. Span, A new equation of state for carbon dioxide covering the fluid region from the triple-point temperature to 1100 K at pressures up to 800 MPa. J. Phys. Chem. Ref. Data, 25(6):1509, 1996.

The arborescence of the properties server directories can be set up at the user’s discretion. The only requirements are that the dll links be placed in the installation directory and the list of systems proposed in the “mixtures” folder. In the case of TEP ThermoSoft, this list in contained in the file “TEPThSoft.mix”.

The constructor of the class TEPThermoSoft.java is very similar to the class considered above:

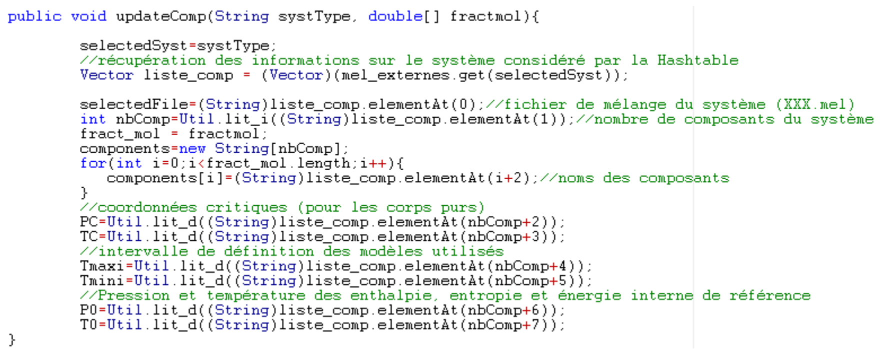


```java
public TEPThermoSoft (){
    super();
    type=getType();
    M=44;PC=73.77;TC=304.128;
    Tmaxi=1100;
    Tmini=216.7;
    chemForm="TEP ThermoSoft mixture properties";
    vMixtures= new Vector();
    mel_externes= new Hashtable();
    getMixtures();
}

public Vector getMixtures(){

    vMixtures= new Vector();
    nomfich= new StringBuffer("TEPThSoft.mix");
    dir_data=System.getProperty("user.dir")+File.separator+"mixtures"+File.separator;
    fich = new File(dir_data,nomfich.toString());
    lect_data(fich);
    return vMixtures;
}
```

It introduces a Hashtable to be able to easily reference the existing external mixtures, and executes the method getMixtures() which constructs the Vector of existing mixtures and the Hashtable, by analyzing the mixture file "TEPThSoft.mix" using the method lect_data(). The same class can be used with various sets of mixtures depending on the applications.

The three methods of the ExternalMixture interface are then defined, which does not pose any particular problem.

The method updateComp must be able both to initialize the system correctly and update its composition. To do so, it calls the Hashtable created by the constructor.

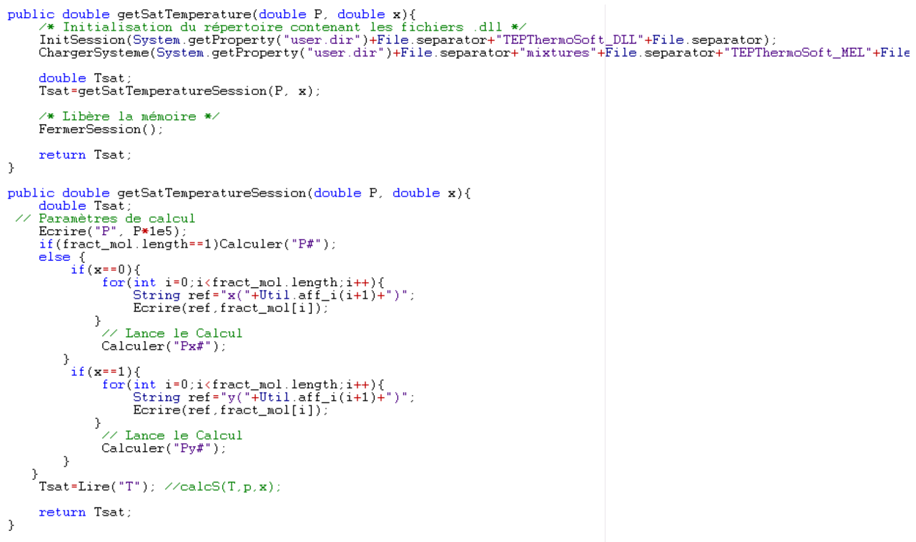


```java
public void updateComp(String systType, double[] fractmol){

        selectedSyst=systType;
        //récupération des informations sur le système considéré par la Hashtable
        Vector liste_comp = (Vector)(mel_externes.get(selectedSyst));

        selectedFile=(String)liste_comp.elementAt(0);//fichier de mélange du système (XXX.mel)
        int nbComp=Util.lit_i((String)liste_comp.elementAt(1));//nombre de composants du système
        fract_mol = fractmol;
        components=new String[nbComp];
        for(int i=0;i<fract_mol.length;i++){
           components[i]=(String)liste_comp.elementAt(i+2);//noms des composants
        }
        //coordonnées critiques (pour les corps purs)
        PC=Util.lit_d((String)liste_comp.elementAt(nbComp+2));
        TC=Util.lit_d((String)liste_comp.elementAt(nbComp+3));
        //intervalle de définition des modèles utilisés
        Tmaxi=Util.lit_d((String)liste_comp.elementAt(nbComp+4));
        Tmini=Util.lit_d((String)liste_comp.elementAt(nbComp+5));
        //Pression et température des enthalpie, entropie et énergie interne de référence
        P0=Util.lit_d((String)liste_comp.elementAt(nbComp+6));
        T0=Util.lit_d((String)liste_comp.elementAt(nbComp+7));
}
```

An example of how to calculate the saturation temperature is given below (bubble if x = 0, dew if x = 1). It provides a good illustration of the call sequence to the TEP ThermoSoft methods documented in Appendix 1. Schematically, we start by initializing the session and we load in the system in question, then we initialize the pressure, converted from bars to Pascals, and the composition of the mixture. Depending on whether it is a pure substance or a true mixture, the execution of the calculation differs slightly. The work session is closed to free up resources, and the saturation temperature is returned to Thermoptim.

In order to accelerate the calculations, it is better to avoid opening and closing a session whenever possible. Consequently, variants of certain methods utilizable only when a session is open have been implemented. They differ from the others in that the term Session is added to their name.

In both cases, we have created two sister methods: getSatTemperature() and getSatTemperatureSession(). The first manages the opening and closing of sessions, and second performs the calculations. This makes it possible to directly call the second without reinitializing the system.



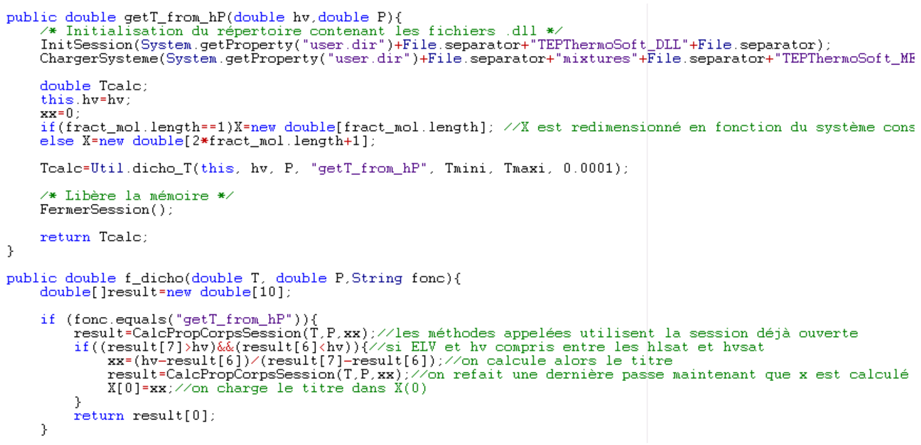
```java
public double getSatTemperature(double P, double x){
    /* Initialisation du répertoire contenant les fichiers .dll */
    InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);
    ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File

    double Tsat;
    Tsat=getSatTemperatureSession(P, x);

    /* Libère la mémoire */
    FermerSession();

    return Tsat;
}

public double getSatTemperatureSession(double P, double x){
    double Tsat;
 // Paramètres de calcul
    Ecrire("P", P*1e5);
    if(fract_mol.length==1)Calculer("P#");
    else {
        if(x==0){
            for(int i=0;i<fract_mol.length;i++){
                String ref="x("+Util.aff_i(i+1)+")";
                Ecrire(ref,fract_mol[i]);
            }
            // Lance le Calcul
            Calculer("Px#");
        }
        if(x==1){
            for(int i=0;i<fract_mol.length;i++){
                String ref="y("+Util.aff_i(i+1)+")";
                Ecrire(ref,fract_mol[i]);
            }
            // Lance le Calcul
            Calculer("Py#");
        }
    }
    Tsat=Lire("T"); //calcS(T,p,x);

    return Tsat;
}
```

The other calculation methods implemented are more complex, specifically those that use functions involving finding roots using the bisection method, but the principle is the same. We will refer to the comments documenting the class for additional information.

3.1.5.3 Inversion of functions

The example below is for the method inverting the enthalpy equation to solve for T, when pressure is known.

Overall the procedure is similar: Upon initialization, a session is opened, and the system is loaded. The inversion calculation can then be performed in this session, since the function f_dicho calls the version of CalcPropCorps that does not open its own session. When the calculations are done, the session is closed.

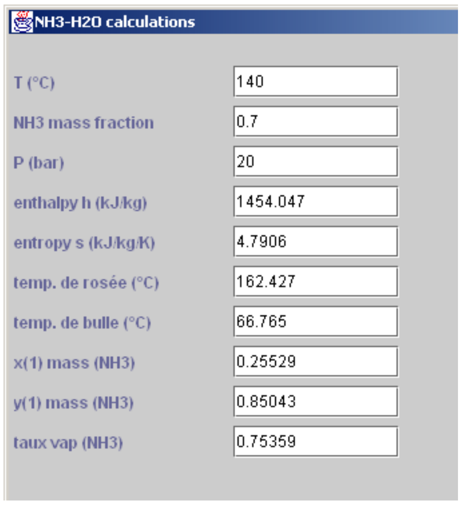


```java
public double getT_from_hP(double hv,double P){
    /* Initialisation du répertoire contenant les fichiers .dll */
    InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);
    ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_ME

    double Tcalc;
    this.hv=hv;
    xx=0;
    if(fract_mol.length==1)X=new double[fract_mol.length]; //X est redimensionné en fonction du système cons
    else X=new double[2*fract_mol.length+1];

    Tcalc=Util.dicho_T(this, hv, P, "getT_from_hP", Tmini, Tmaxi, 0.0001);

    /* Libère la mémoire */
    FermerSession();

    return Tcalc;
}

public double f_dicho(double T, double P,String fonc){
    double[]result=new double[10];

    if (fonc.equals("getT_from_hP")){
        result=CalcPropCorpsSession(T,P,xx);//les méthodes appelées utilisent la session déjà ouverte
        if((result[7]>hv)&&(result[6]<hv)){//si ELV et hv compris entre les hlsat et hvsat
            xx=(hv-result[6])/(result[7]-result[6]);//on calcule alors le titre
            result=CalcPropCorpsSession(T,P,xx);//on refait une dernière passe maintenant que x est calculé
            X[0]=xx;//on charge le titre dans X(0)
        }
        return result[0];
    }
```

There is one particularity of this inversion method: during a liquid-vapor equilibrium, for a pure substance, the temperature determination is not enough. The vapor quality x must also be determined. It is calculated in f_dicho, then loaded in X[0], which Thermoptim can access using the method getQuality() of ExtSubstance.

For the time being, only the vapor quality of pure substances is used by the Thermoptim internal classes, but, as shown in the following section, the mechanism implemented in TEPThermoSoft.java is generic and allows the external classes to access the composition of multi-component mixtures.

3.1.5.4 Example of calculating the external mixture (NH3-H2O)



NH3-H2O calculations

T (°C)	140
NH3 mass fraction	0.7
P (bar)	20
enthalpy h (kJ/kg)	1454.047
entropy s (kJ/kg/K)	4.7906
temp. de rosée (°C)	162.427
temp. de bulle (°C)	66.765
x(1) mass (NH3)	0.25529
y(1) mass (NH3)	0.85043
taux vap (NH3)	0.75359

TEP ThermoSoft proposes the pair NH3-H2O as a mixture. In order to illustrate the calculations possible, we have created a small class called NH3_Absorption.java, which displays a screen of the principal properties useful for calculating absorption systems that use this mixture.

As usual for this application, the composition is entered in mass variables.

In the example opposite, at 140° C for a mass fraction of NH3 equal to 0.7, the point is located inside the isobaric lens at 20 bars, the bubble and dew temperatures are respectively 66.8 °C and 162.4 °C.

TEP ThermoSoft sends the liquid and vapor molar fractions of NH3, as well as the mean vaporization rate. Their values are sent by the array double[] X, which, in the case of a pure substance, contains only the vapor quality.

The code for performing the calculations in the class NH3_Absorption.java illustrates the use of the array X by an external class:

```java
void bCalcul_actionPerformed(java.awt.event.ActionEvent event)
{
	TEPThermoSoft monCorps=new TEPThermoSoft();//instanciation du corps
	double x1Mass=Util.lit_d(fractmassNH3.getText());
	double x1Mol=18.0153*x1Mass/(17.0306+x1Mass);//fraction molaire
	double[]fractmol={x1Mol,1.-x1Mol};
	monCorps.updateComp("NH3-H2O",fractmol);//choix du système et mise à jour de la composition

	double T=Util.lit_d(T_value_C.getText())+273.15;
	double P=Util.lit_d(P_value.getText());
	System.out.println("\n\nt  :  "+T_value_C.getText()+"  P : "+P+"  x1(mol) : "+x1Mol+"  x1(mass) : "+x1Mass);
	double Trosee=monCorps.getSatTemperature(P, 1)-273.15;
	Trosee_value.setText(Util.aff_d(Trosee,3));

	double Tbulle=monCorps.getSatTemperature(P, 0)-273.15;
	Tbulle_value.setText(Util.aff_d(Tbulle,3));

	double[]result=monCorps.CalcPropCorps(T,P,1);
	h_value.setText(Util.aff_d(result[0],3));
	s_value.setText(Util.aff_d(result[5],4));
	x1_value.setText(Util.aff_d((17.0306*monCorps.X[1])/(18.0153-monCorps.X[1]),5));
	y1_value.setText(Util.aff_d((17.0306*monCorps.X[3])/(18.0153-monCorps.X[3]),5));
	x_value.setText(Util.aff_d(result[3],5));
}
```

3.1.6 Thermodynamic charts of external mixtures

External mixtures not being included in Thermoptim, the software does not provide their thermodynamic charts. To overcome this limitation, we added a new chart type, called external mixture chart, which allows the use of simplified entropy and (h,P) charts.

The preparation of the chart background can be made thanks to a special external class called CreateMixtureCharts.java. You should refer to its documentation for further details on this topic.

The new charts are simplified compared to the others in that they show only the bubble and dew curves, as well as a single set of isovalues, i.e. the isobars for the entropy chart, and the isotherms for the (h, P) chart.

These charts being a variation of vapor charts, their use is explained in the documentation of these.

3.2 External Processes

The structure of external classes was defined earlier in this manual: The class extThopt.TransfExterne inherits from rg.corps.TransfoExt of Thermoptim and performs the interface, relaying the calculations at the level of an abstract class (extThopt.ExtProcess) defining the basic methods, and subclassed by the different classes introduced (for example extThopt.SolarCollector). We recommend that you refer to the API of extThopt.TransfExterne and extThopt. ExtProcess to learn the syntax and function of the methods available. This API is part of the development environment of the external classes available (in the folder api_extThopt).

We will begin by explaining the construction procedure implemented in Thermoptim, then we will show how to create a new external component, by subclassing extThopt. ExtProcess.

3.2.1 Construction

3.2.1.1 Construction of an external process in Thermoptim

The construction procedure is as follows:

1. the user selects an external process from the list, which shows the index in the arrays of external classes loaded in Thermoptim.
2. you load this class, which you transtype in ExtProcess, and encapsulate in an Object.

Class c=(Class)rg.util.Util.componentClasses[i];//you load the class of the external component

Constructor ct=c.getConstructor(null);//you carefully instantiate it with its constructor without an argument.

extThopt.ExtProcess ec=(extThopt.ExtProcess)ct.newInstance(null);

Object ob=ec;//you encapsulated it in an Object

1. you then instantiate the class TransfExterne, which inherits from ComposantExt, i.e., it is in fact a JPanel containing the user interface defined in the external process.

JPanel1.remove(cType);

cType=new TransfExterne((Projet)Proj,ob, this);//we instantiate the external component

setupCType();//sets up the external user interface

which is done by the following constructor:

public TransfExterne(Projet proj, Object obj, TransfoExterne te){

this.proj=proj;

this.te=te;//we pass the ference to the TransfoExterne internal to Thermoptim

ep= (ExtProcess)obj;//retrieves the external class instantiated in TransfoExterne

ep.proj=proj;//retrieves the reference of the project

Vector vSetUp=new Vector();

vSetUp.addElement(te);//sends the reference to the TransfoExterne

vSetUp.addElement(ep.JPanel1);//loads the interface

vSetUp.addElement(ep.thermoCouplerTypes);//definition of the types of thermocouplers required

setUpFrame(vSetUp);//execution of setCompFrame();

ep.tfe=this;

}

1. if the construction is done while the project file is being read, the process parameters are updated
2. when a project is completely loaded, Thermoptim executes in each external component a special method called init() which performs initializations by referencing other instantiated external components (see section 3.4). This makes it possible to synchronize their methods.

3.2.1.2 Creating an external process

To create an external process, simply subclass extThopt.ExtProcess. Let us look at the example of the class SolarCollector (we have provided only part of the construction of the graphic interface here).

public SolarCollector (){

```
super();
JPanel1.setLayout(null);//Layout of the JPanel
JPanel1.setBounds(0,0,400,300);//dimensions of the JPanel (generally standard)
JLabel1.setText("glass transmittivity");//definition of the first label
JPanel1.add(JLabel1);
JLabel1.setBounds(0,0,164,24);
JPanel1.add(tau_value);//definition of the first editable text field for entering the glass transmittivity
tau_value.setBounds(164,0,124,24);
tau_value.setText("0.8");
type=getType();//type of process
thermoCouplerTypes=new String[0];//no thermocoupler connected
}
public String getType(){
return "solar collector";
}
public String getClassDescription(){
```

> return "flat plate solar collector (without thermocoupler)\n\nauthor : R. Gicquel january 2003\n\nRef : note MODÉLISATION D'UN CAPTEUR SOLAIRE THERMIQUE";

```
}
```

3.2.2 Updating and calculating the process

Here we refer to the example of the class SolarCollector, whose model is presented in the note “SolarCollector.doc”

The sequence of operations is as follows:

1. update the component before calculation by loading the values of the process and the upstream point
2. read the parameters on the external component screen
3. calculate the power used and the state of the downstream point
4. calculate the thermal loads of the thermocouplers
5. update the external component screen

Let us now look at the practical problems encountered during each of these steps. A few portions of the code are provided below, but we recommend reading the rest of this note while referring to the entire class SolarCollector.java.

1) update the component by loading the values of the process and the upstream point

The problem here is that an external component does not have direct access to the simulator variables: these values are obtained by very general methods, which construct Vectors with different structures depending on the desired object.

The procedure is not complicated, but it must be followed carefully:

```
String[] args=new String[2]; //array of arguments
args[0]="process";//type of element desired (a process in this case)
args[1]=tfe.getCompName();//name of the process (obtained by the reference tfe)
Vector vProp=proj.getProperties(args);//Project method given in Appendix 2
Double f=(Double)vProp.elementAt(3);
double flow=f.doubleValue();//flow rate value, automatically propagated from the upstream process
String amont=(String)vProp.elementAt(1);//name of the upstream point
```

```
getPointProperties(amont);//automatic decoding of the Vector (method of the ExtProcess class)
Tamont=Tpoint;//here T1
```

The method getPointProperties() of ExtProcess automatically loads the state of a point in easily manipulable values, called Tpoint, Point, lecorps… The method is given below.

```
public void getPointProperties(String nom){
String[] args=new String[2];
args[0]="point";//type of the element (see method getProperties(String[] args))
args[1]=nom;//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
lecorps=(Corps)vProp.elementAt(0);
nomCorps=(String)vProp.elementAt(1);
Double y=(Double)vProp.elementAt(2);
Tpoint=y.doubleValue();
y=(Double)vProp.elementAt(3);
Ppoint=y.doubleValue();
y=(Double)vProp.elementAt(4);
Xpoint=y.doubleValue();
y=(Double)vProp.elementAt(5);
Vpoint=y.doubleValue();
y=(Double)vProp.elementAt(6);
Upoint=y.doubleValue();
y=(Double)vProp.elementAt(7);
Hpoint=y.doubleValue();
y=(Double)vProp.elementAt(9);
DTsatpoint=y.doubleValue();
String dum=(String)vProp.elementAt(8);
isTsatSet=Util.lit_b(Util.extr_value(dum));
dum=(String)vProp.elementAt(10);
isPsatSet=Util.lit_b(Util.extr_value(dum));
}
```

1. **read the parameters on the external component screen**

The package extThopt provides a number of simple but robust methods for converting the Strings displayed in the JTextField fields used on the graphic interface to doubles, and vice versa for displaying the doubles in these fields. They are implemented as static methods of the extThopt.Util class (see Appendix 3):

```
P=Util.lit_d(P_value.getText());
A=Util.lit_d(A_value.getText());
tau=Util.lit_d(tau_value.getText());
K=Util.lit_d(K_value.getText());
Tex=Util.lit_d(Tex_value.getText())+273.15;
```

1. **calculate the power used and the state of the downstream point**

We begin by estimating the Cp of the heat-conducting fluid by making a limited development of the enthalpy function, which means we use the method CalcPropCorps() of the package Corps and the method getSubstProperties() of ExtProcess, which automatically loads the state of a point in easily manipulable values, called Tsubst, Psubst, etc. :

```
public void getSubstProperties(String nom){
String[] args=new String[2];
args[0]="subst";//type of the element (see method getProperties(String[] args))
args[1]=nom;//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
Double y=(Double)vProp.elementAt(0);
Tsubst=y.doubleValue();//temperature
y=(Double)vProp.elementAt(1);
Psubst=y.doubleValue();//pressure
y=(Double)vProp.elementAt(2);
Xsubst=y.doubleValue();//value
y=(Double)vProp.elementAt(3);
Vsubst=y.doubleValue();//mass volume
y=(Double)vProp.elementAt(4);
Usubst=y.doubleValue();//internal mass energy
y=(Double)vProp.elementAt(5);
Hsubst=y.doubleValue();//mass enthalpy
y=(Double)vProp.elementAt(6);
Ssubst=y.doubleValue();mass entropy
y=(Double)vProp.elementAt(7);
Msubst=y.doubleValue();//molar mass
Integer i=(Integer)vProp.elementAt(8);
typeSubst=i.intValue();//type of substance (1 for water, 2 for a vapor, 3 for a pure gas, 4 for a compound gas, 5 for an external substance)
y=(Double)vProp.elementAt(13);
ChemExerSubst=y.doubleValue();
}
```

Then we can calculate the Cp as follows:

```
double H=Hpoint;//enthalpy of the upstream point
lecorps.CalcPropCorps(Tpoint+1, Ppoint, Xpoint);// recalculates the upstream substance by a Thermoptim function
getSubstProperties(nomCorps);//retrieves the recalculation values (method of the ExtSubstance class)
double Cp=(Hsubst-H);//estimated value of Cp
```

We then calculate an estimated Taval value to be able to determine the absorbed thermal power Qex:

```
double DT0=tau*P/K-Tamont+Tex;
double T=Tamont+(DT0)*(1-Math.exp(-K*A/flow/Cp));
double DT=T-Tpoint;
double Qex=Cp*DT*flow;
```

We determine the value of the mass enthalpy of the downstream point, then we invert this equation to determine the exact value of Taval (method of the Corps class)

double hAval=Qex/flow+Hpoint;

Tpoint=lecorps.getT_from_hP(hAval,Ppoint);

getSubstProperties(nomCorps);//retrieves the recalculation values (method of the ExtProcess class)

Xpoint=Xsubst;//updates the value of the vapor quality in case the state is diphasic

1. **calculate the thermal loads of the thermocouplers**

In this simple example, there is no problem, since the component does not use a thermocoupler. Brief instructions are given later in the manual, as well as in the section on external nodes.

1. **update the external component screen**

The Thermoptim method setupPointAval() updates the downstream point from the values loaded in a Vector constructed here by the method getProperties() of ExtProcess:

tfe.setupPointAval(getProperties());

The solar collector yield value is then determined and displayed.

eff_value.setText(Util.aff_d(Qex/P/A, 4));

For this example, the updates before and after recalculation are very simple. In other cases, it may be necessary to access other data from the simulator. We will explain how to do this below.

3.2.3 Moist gas calculations

To perform the calculations for moist gases from external classes, the method getPointProperties() of ExtProcess retrieves the moist properties values of a point by the following variables:

Wpoint for the absolute humidity w, Epsipoint for relative humidity ε, Qprimepoint for specific enthalpy q', Tprimepoint for adiabatic temperature t' (in °C), Trpoint for dew point temperature tr (in °C), VPrimepoint for specific volume vs, Condpoint for condensates, and M_secpoint for the molar mass of the dry gas.

The method updatePoint given in the appendix forces the following moist calculations:

"setW and calculate all", sets w and calculates all of the moist properties

"setW and calculate q'", sets w and calculates all of the moist properties except t'

"setEpsi", which sets ε

"setEpsi and calculate", sets ε and calculates all of the moist properties

"calcWsat", calculates wsat and all of the moist properties except t'

"modHum", modifies the composition of the gas

3.2.4 Calculation of exergy balances

The external components can also be represented in a productive structure. Their exergy balance screen can be configured (using String [] getExergyType ()) by the constructor of the external class, for the time being to display, like that of "exchange" processes, three options and a value input field.

The choices made by the user are then transmitted to the external class for evaluation of getExergyBalance (String [] args), defined below; they are saved in the structure file. If additional settings are required, it is always possible to set them in the component's physical diagram screen.

Many exergy balance calculations of components pose no particular problem. It is however necessary to specify carefully how their different elements should be taken into account in the overall balance, as the system boundary is not same as that of a given component.

To calculate its exergy balance, each simulator component returns a double [] method getExergyBalance (String [] args) which includes the seven values to be taken into account in the overall balance (five ports plus exergetic efficiency and irreversibility).

For Thermoptim core elements, the rules for weighting the values of provided to the cycle can be fixed once and for all, but it is not the same for external components, for which the designer must always specify which values should be included in the overall exergy balance sheet.

The models of components that can be implemented are so diverse that it is not possible to predict all cases. This is why the external components refer, in addition to the previous seven values, to two coefficients ranging between 0 and 1 (tauPlusFactor and deltaXhPlusFactor) which allow one to weight the fraction of the useful work and positive heat exergy supplied to the cycle.

The ExtProcess class includes a default implementation of String [] getExergyType() and double [] getExergyBalance (String [] args), to be sub-classed by external components. The exergy fluid properties must be calculated from the values of enthalpy, entropy and T0 provided by getSubstProperties ().

The exchange of getExergyBalance between Thermoptim and the external components is done by public double [] getCompExergyBalance (String [] args).

For the calculation of exergy of incoming and outgoing flows, a reference value is taken for T0 and 1 bar.

In PointCorpsDemo, its implementation without argument is as follows:

```
// calculates the reference exergy, chemical exergy not to be accounted
public double getExergyReference (){//modRG oct04//modRG exerg
lecorps.CalcPropCorps(Util.T0Exer,1,1);
return lecorps.H-Util.T0Exer*lecorps.S;
}
```

In ExtProcess it is, with two arguments:

/**

- returns exergy reference value and initializes T0Exer
- /

```
public double getExergyReference(Corps corps,String nomCorps){
String[] args=new String[2];
args[0]="project";//type of the element (see method getProperties(String[] args))
args[1]="";//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
Double f=(Double)vProp.elementAt(2);
T0Exer=f.doubleValue();
corps.CalcPropCorps(T0Exer,1.0,0);
getSubstProperties(nomCorps);
return Hsubst-T0Exer*Ssubst;
}
```

Default implementations for the external components are given below:

The ExtProcess class includes a default implementation of getExergyType () and getExergyBalance (String [] args), to be sub-classed by external components. The fluid exergy properties are given by getSubstProperties ().

```
double tauPlus,deltaXhPlus,xqPlus,tauProduct,deltaXhProduct,etaExer,deltaXhi;
void setExergyExtensor(boolean isExtensor){
if(isExtensor)exergyType="extensor";
else exergyType="reductor";
}
public String[] getExergyType(){//modRG exerg
String[] type=new String[9];
type[0]=exergyType;//extensor or reductor
type[1]="false";//affiche JCheckExtSource
type[2]="External source";//label JCheckExtSource
type[3]="false";//affiche JCheckIntExchange
type[4]="Internal exchange";//label JCheckIntExchange
type[5]="false";//affiche JCheckValuableExergy
type[6]="Valuable exergy";//label JCheckValuableExergy
type[7]="false";//affiche champ d'entrée de valeur
type[8]="Source T (°C)";//label sourceT_value
return type;
}
public double[] getExergyBalance(String[]args){//modRG exerg
double[] exergyBalance=new double[9];
exergyBalance[0]=tauPlus;
exergyBalance[1]=deltaXhPlus;
exergyBalance[2]=xqPlus;
exergyBalance[3]=tauProduct;
exergyBalance[4]=deltaXhProduct;
exergyBalance[5]=etaExer;
exergyBalance[6]=deltaXhi;
exergyBalance[7]=tauPlusFactor;
exergyBalance[8]=deltaXhPlusFactor;
return exergyBalance;
}
```

3.2.4.1 Class SolarConcentratorCC

A solar panel converts the solar flux received in heat transmitted to the fluid flowing through it. This is therefore an exergy extensor, and in the SolarConcentratorCC class constructor, the type of UPD is set by setExergyExtensor (true);

The exergy calculations can be performed as follows:

```
double Xh0=getExergyReference(refrig);
tauPlus=0;
deltaXhPlus=0;
tauProduct=0;
xqPlus=P/1000*Sc*(1-T0Exer/5800);//le soleil est une source à 5800 K
deltaXhProduct=(Haval-Hamont-T0Exer*(Saval-Samont))*flow;
deltaXhi=xqPlus-deltaXhProduct;
etaExer=deltaXhProduct/xqPlus;
```

3.2.4.2 Class FluidEjector

An ejector is modeled as an external mixer. Depending on whether one is interested in motor or entrained flow, an exergy reductor or an exergy extensor. However, it may also simply be considered as a simple mixer, which avoids associating a junction or a branch.

To ensure that the incoming exergy is not considered as an external input to the cycle, the following two lines are inserted in the constructor:

```
deltaXhPlusFactor = 0;
tauPlusFactor = 0;
```

The exergy calculations can be performed as follows:

```
double Xhmi=Hmi-T0Exer*Smi-Xh0-getExergyReference(refrig,nomCorps);
double Xhmsi=Hsi-T0Exer*Ssi-Xh0-getExergyReference(refrig,nomCorps);
double Xhsr=Hd_is-T0Exer*Smix-Xh0-getExergyReference(refrig,nomCorps);
deltaXhProduct=msr*Xhsr;
deltaXhPlus=msi*Xhmsi+mi*Xhmi;
etaExer=deltaXhProduct/deltaXhPlus;
deltaXhi=deltaXhPlus-deltaXhProduct;
```

3.2.4.3 Class SOFCH2outlet

A fuel cell converts hydrogen into electricity. It behaves like a quadrupole receiving two input fluids, and out of which come the other two. The quadrupole is formed by combining an input mixer and an output divider, the two being connected by a process-point which plays a passive role. The calculations are performed by the output divider.

The exergy calculations can be performed as follows:

```
double DH0=-241830;//kJ/kmol H2 vapeur
double DH0_vap=-285830;//kJ/kmol H2 liquide
double DG0=-237160;//kJ/kmol H2
double elecPower=tau*DG0*epsi*molFlowH2;
double Qlib=-tau*DH0*molFlowH2+elecPower;
tauProduct=-elecPower;
etaExer=elecPower/DH0/molFlowH2;
deltaXhi=(1-etaExer)*tauProduct;
```

3.2.5 Managing Energy Types

It may first be necessary to manage the assignment of energy types in a more complex manner than in the processes of Thermoptim’s basic set, which are basically mono-functional. For the processes, the energies used are either purchased, useful, or other. In an external process, things may be different, with for example a thermal load in purchased energy and a power in useful energy.

The method updateProcess() of ComposantExt assigns the desired values to the different types of energy. It is easily used with the method setEnergyTypes of ExtThopt:

```
tfe.updateProcess(setEnergyTypes(tfe.getCompName(),useful,purchased,other));
public Vector setEnergyTypes(String process, double useful,double purchased, double other){
Vector vEner=new Vector();
vEner.addElement(process);//process name
vEner.addElement(new Double(useful));//useful energy
vEner.addElement(new Double(purchased));//purchased energy
vEner.addElement(new Double(other));//other energy
return vEner;
}
```

3.2.5 Access to other elements of the simulator

Access to upstream and downstream processes

The names of the upstream and downstream processes are accessible by the elements 9 and 10 of the Properties Vector of the external process.

String[] args=new String[2];

args[0]="process";//type of the element (see method getProperties(String[] args))

args[1]=tfe.getCompName();//name of the process (see method getProperties(String[] args))

Vector vProp=proj.getProperties(args);

String transfoAmont=(String)vProp.elementAt(9);//name of the upstream process (or "null" if none)

String transfoAval=(String)vProp.elementAt(10);//name of the downstream process (or "null" if none)

Once the name has been obtained, we access its properties by sending it to args[1] in proj.getProperties(args). In this way we can recursively run through the upstream and downstream processes directly connected to a process.

This is the mechanism used to update the processes upstream and downstream of the external nodes in the method public void updateStraightlyConnectedProcess(String startProcess, String name, boolean downstream, boolean inletPoint, boolean outletPoint, boolean updateT, double T, boolean updateP, double P, boolean updateX, double x) of ExtNode.

The example provided in the note “CycleLiBrH2O.doc” that is partially reprinted as an illustration of external nodes shows how to use these mechanisms.

Let us note in passing that vProp=proj.getProperties(args) also gives access to certain global properties of the project if args[0]="project" (see Appendix 2), and specifically the flow unit. This is a way to verify that the flow and power units implicitly or explicitly used in the external class are compatible with those of the project, and to send a message otherwise.

Thermocouplers

Given that thermocouplers are a type of heat exchanger, it is valuable to define them by values such as effectiveness ε, UA, NTU or LMTD, that can be calculated using similar equations.

The external component must send to each of its thermocouplers the equivalent values for flow rates, inlet and outlet temperature and thermal energy transferred, which they must take into account in their calculations. Specific methods were placed in the interface for this purpose.

However, the analogy with exchanges has certain limits: for example, temperature crossovers unacceptable in an exchanger may occur in a thermocoupler, leading to absurd values.

So it is best to transmit values that are unlikely to lead to this type of situation. One possible solution is to assume that the thermocoupler is isothermal for calculations of characteristics that are similar to exchanger characteristics, as in the model selected from the absorber and the desorber in the example presented in the note "TrigenMicroTAG.doc".

To be able to accept multiple couplings, all the heatConnectable define an array of acceptable thermocoupler types (in the example above: String[]thermoCouplerTypes={"absorber","desorber"}). The calls are then done using the type of thermocoupler as the identifier, as in:

public double getInletTemperature(String thermoCouplerType);

When the array has dimension 1 and the heatConnectable is a process, the management can be simplified (see below). Otherwise, the heatConnectable has to be able to distinguish which thermocoupler is calling it, in order to know what to send it back by the interface methods. It must therefore register with each thermocoupler when the thermocoupler is created. This assumes that a consistency check is performed during the initial construction, and this check must be repeated for each subsequent reconstruction.

For any external component there are also additional difficulties, given that its access to internal values is limited (the cases mentioned here are relative to process, but the problems also arise for nodes):

- the dimension of the array of acceptable thermocoupler types and those that are associated to it is not necessarily known, which means it must be initialized specifically in the method setCompFrame(Object obj) of ComposantExt, which is the pivot class where a large part of the exchanges between the internal part of the code and the external components are performed.
- the thermocoupler updates are done by the non-obfuscated method of TransfoExterne updateThermoCouplers(Vector vTC)

- the exchanges of information between the heatConnectable and its thermocouplers pass through Vectors, and it must be able to distinguish the different values depending on the role of each thermocoupler.

The method updateThermoCoupler() of ExtProcess performs the updates:

updateThermoCoupler(String type, double Tin, double Tout, double Q, double flow)

It sends the update elements to the thermocoupler via a Vector constructed by the methods of the ExtProcess class:

```
protected Vector getThermoCouplerVector(String type, double Tin, double Tout, double Q, double flow){
Vector vTC=new Vector();
vTC.addElement(type);
Double d=new Double(Tin);
vTC.addElement(d);
d=new Double(Tout);
vTC.addElement(d);
d=new Double(Q);
vTC.addElement(d);
d=new Double(flow);
vTC.addElement(d);
return vTC;
}
```

Though it is not recommended, we can avoid using the method updateThermoCoupler() for a simple external process, with just one thermocoupler and for which the inlet and outlet temperatures, thermal load and flow rate values can be directly obtained from the values of the process. However, if there are more than one thermocouplers connected to the same process, each one must be updated after each recalculation.

Note that a thermocoupler can be initialized only by the external component that it depends on. When a project is loaded, the state of the thermocouplers is updated only when the external components are calculated. To avoid problems during the first automatic recalculation of a project, we recommend performing a calculation of the external components equipped with thermocouplers using the method init() (see section 3.2.1.1).

Nodes

To identify the nodes to which a process can be connected, it is necessary to get the project nodes using the method proj.getNodeList(), retrieve the structure of each one by calling proj.getProperties() with the right arguments, and see if the name of the process appears among the names of the branches or the main process of the node. It’s a bit laborious, but even from within Thermoptim, this is how it has to be done.

Accessing the diagram editor

It is also possible to access the content of the diagram editor from the following Project methods, which are somewhat difficult to use, especially the second one:

getEditorComponentList(): provides the list of components present in the diagram editor, in the form "name"=componentName+tab+"type"=componentType. This list can be extracted easily using Util.extr_value("name") and Util.extr_value("type").

getConnectionCode(String[] args): sends a code showing whether two components of the editor are connected or not: 1 if the component 1 is upstream of component 2, -1 if it is downstream, and 0 if they are not connected. The structure of the arguments is as follows:

```
String[] args=new String[4];
args[0]="Expansion";//type of component 1
args[1]="turbine";//name of component 1
args[2]="Exchange";//type of component 2
```

args[3]="régen gaz";//name of component 2

The type codes are those used in the saved project files.

3.2.6 Saving and loading the parameters of the model

It is possible to save the parameters of the external components in the usual Thermoptim project files, and subsequently read them.

public String saveCompParameters()

public void readCompParameters(String ligne_data)

The only constraint is that all of the parameters of the external component must fit on one line, in a format compatible with that used in the software basic set: the different save fields are separated by tabs.

ExtThopt.Util provides a generic method for associating an identifying code to the value of a parameter:

public static String extr_value(String ligne_data, String search)

The save is done in the form “parameter-value” and the search is done in the same way:

valeur=Util.lit_d(Util.extr_value(ligne_data, "parameter"));

If the parameters of the component are too complex to be saved in this way, the user can use this mechanism to save the name of a specific parameter file and then read that file as necessary.

3.3 External nodes

We recommend that you refer to the API of extThopt.MixerExterne, extThopt.DividerExterne, extThopt.ExtMixer, extThopt.ExtDivider and extThopt.ExtNode to learn about the syntax and functions of all of the methods available. This API is part of the external class development environment available to you (in the folder api_extThopt).

3.3.1 Construction

The construction of an external node is identical to that of an external process.

3.3.2 Updating and calculating a node

Here we refer to the example of the class Absorber, a model of which is presented in the specific note called "CycleLiBrH2O.doc" presenting the internal model of an absorption cycle.

The sequences of operations are as follows:

1. consistency check and updating of the node before calculation
2. reading the parameters on the external component screen
3. updating the processes connected to the external node
4. updating and calculating the associated thermocouplers
5. updating the screen of the external component
6. saving and loading the parameters of the model

Let us now look at the practical problems encountered during each of these steps. A few portions of the code are provided below, but we recommend reading the rest of this note while referring to the entire class Absorber.java.

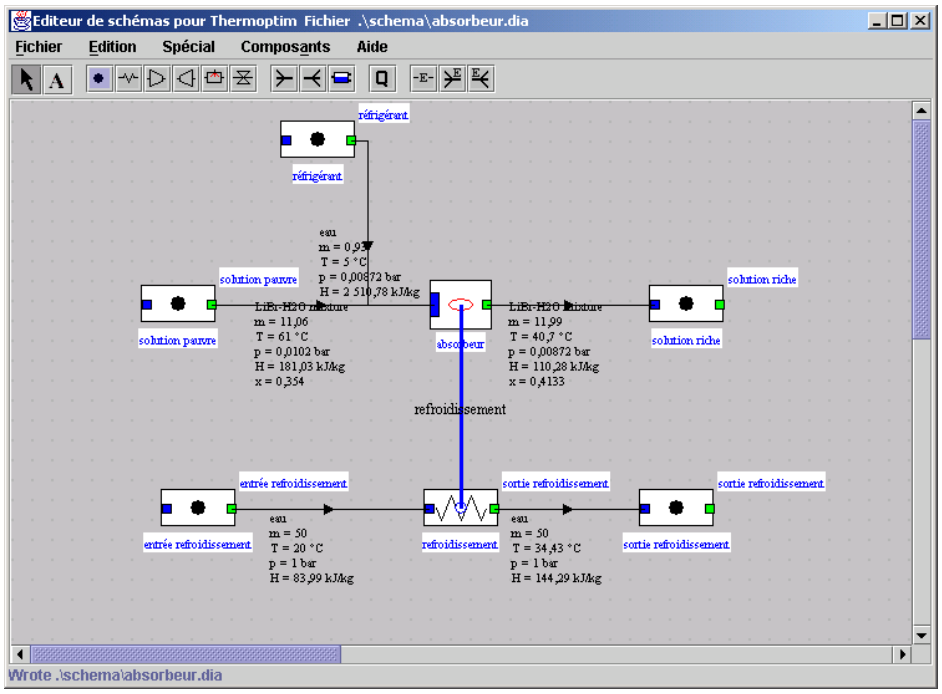
1) consistency check and updating of the node before calculation

There are several difficulties involved:

- first, it is highly important to make sure that the node is correctly constructed, i.e. that the branches and main process are indeed what the model designer expects. Since there is no consistency check for this in the diagram editor, a user can very easily make a mistake in describing the diagram.
- next, an external component has no direct access to the variables of the simulator: these values are obtained by very general methods, which construct Vectors with different structures depending on the desired object.

Consequently, the first step consists of **checking the consistency of the node construction.** To do this, the designer has a method of the ExtNode class, getNodeStructure(), which decodes the structure of the node, by loading in a String called mainProcess the name of the main process, and in a String Array called nomBranches[] the names of the processes of the different branches, the number of which corresponds to the dimension of the array.

The designer must program a method (called checkConsistency() in our example) in which he runs through the various processes of the node, performing all the tests he wants, and at the same time updating the inlet parameters of his model.



For example, for the absorber, the method is as follows (to facilitate subsequent processing, we decided to load the names of the processes into easily identifiable Strings called richSolutionProcess, refrigerantProcess and poorSolutionProcess, and the names of the points into richSolutionPoint, refrigerantPoint and poorSolutionPoint):

private void checkConsistency(){

String[] args=new String[2];

Vector[] vBranch=new Vector[2];

isBuilt=true;

poorSolutionProcess="";

refrigerantProcess="";

if(nBranches!=2){//**first test to check the number of branches**

String msg = "Error on the number of branches which is "+nBranches+" instead of 2";

JOptionPane.showMessageDialog(me, msg);

isBuilt=false;

}

else{

for(int j=0;j<nBranches;j++){

args[0]="process";//type of the element (see method getProperties(String[] args))

args[1]=nomBranche[j];//name of the process (see method getProperties(String[] args))

vBranch[j]=proj.getProperties(args);//**loads the corresponding process**

String aval=(String)vBranch[j].elementAt(2);//gets the downstream point name

```
getPointProperties(aval);//direct parsing of point property vector
String nom=nomCorps;
//Check the substance at inlet //checks the name of the substance
System.out.println(" ligne "+j+" nomCorps "+nomCorps);
if(nom.equals("LiBr-H2O mixture")){//if it is a mixture, the poor solution was detected
poorSolutionProcess=nomBranche[j];
poorSolutionPoint=aval;
//initialise the inlet variables
Tsp=Tpoint;
Psp=Ppoint;
Hsp=Hpoint;
Xsp=Xpoint;
Double f=(Double)vBranch[j].elementAt(3);
msp=f.doubleValue();
}
if(nom.equals("eau")){// if it is water, refrigerant was detected
refrigerantProcess=nomBranche[j];
refrigerantPoint=aval;
//initialise the inlet variables
Prefr=Ppoint;
Hrefr=Hpoint;
Trefr=Tpoint;
Double f=(Double)vBranch[j].elementAt(3);
mr=f.doubleValue();
}
}
if((refrigerantProcess.equals(""))||(poorSolutionProcess.equals(""))){//otherwise there is an error
String msg = "Error on at least one of the branch substances";
JOptionPane.showMessageDialog(me, msg);
isBuilt=false;
}
}
richSolutionProcess=mainProcess;
args[0]="process";//type of the element (see method getProperties(String[] args))
args[1]=richSolutionProcess;//name of the process (see method getProperties(String[] args))
Vector vPropMain=proj.getProperties(args);
Double f=(Double)vPropMain.elementAt(3);
msr=f.doubleValue();
String amont=(String)vPropMain.elementAt(1);//gets the upstream point name
getPointProperties(amont);//direct parsing of point property vector
```

richSolutionPoint=amont;

Tsr=Tpoint;

Psr=Ppoint;

Hsr=Hpoint;

Xsr=Xpoint;

String nom=nomCorps;

//Check the substance at inlet

if(!(nom.equals("LiBr-H2O mixture"))){//**also check that the main process has the right substance**

String msg = "Error on main process substance,which is "+nomCorps+" instead of LiBr-H2O

mixture";

JOptionPane.showMessageDialog(me, msg);

isBuilt=false;

}

}

1. **reading the parameters on the external component screen**

The package extThopt provides a number of simple but robust methods for converting to doubles the Strings displayed in the JtestField fields used on the graphic interface, and vice versa for displaying the doubles in these fields. They are implemented as static methods of the extThopt.Util class:

Tabs=Util.lit_d(Tabs_value.getText())+273.15;

1. **updating the processes connected to the external node**

The processes associated with the node are updated in Thermoptim by the methods updateMixer(Vector vTC) and updateDivider(Vector vTC), which, in order to account for all possible cases, update the node and the process flow rates, as well as the upstream points of the outlet processes and the downstream points of the inlet processes.

The designer of the external node must therefore correctly configure the Vector which passes as an argument. To do so, he can use two generic methods of ExtMixer and ExtDivider classes:

- updateMixer(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal)
- updateDivider(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal)

to which he must provide Vector arrays for the processes and points of each branch and the main vein, as well as the global temperature and enthalpy values.

To set up the Vectors, ExtNode has a generic method:

void setupVector(String process, String point, int i, double m, double T, double P, double X)

i is the index of the two Vectors, process and point are the names of the process and the point, m is the flow rate of the process, T is the temperature, P is the pressure and X is the vapor quality of the process point closest to the node.

In our example, the solution was to use three intermediate methods to set up the Vectors:

- setupRichSolution(double m, double T, double P, double X)
- setupPoorSolution(double m, double T, double P, double X)
- setupRefrigerant(double m, double T, double P, double X)

Then size the Vector arrays defined in ExtMixer and run updateMixer():

vTransfo= new Vector[nBranches+1];

vPoints= new Vector[nBranches+1];

setupRichSolution(msr,Tabs,Psr,Xsr);

setupPoorSolution(msp,Tsp,Psp,Xsp);

setupRefrigerant(mr,Trefr,Prefr,1);

updateMixer(vTransfo,vPoints,Tsr,Hsr);

For routine cases, once the flow rates and the upstream or downstream points of the processes linked to the node have been updated, Thermoptim propagates the recalculation of the various elements using the automatic recalculation engine. However, this may not suffice and the node designer may have to propagate the updates himself. In fact, this is true in our example for the propagation of the refrigerant concentration value of the mixture LiBr-H2O. This value is stored in what is normally the vapor quality, which is never propagated automatically.

In this case, the designer must look at all the points downstream and upstream of the one whose composition was modified, to make sure that they have been correctly updated as well. He can use the method updateStraightlyConnectedProcess() of the ExtNode class, which recursively runs through the processes connected upstream and downstream of the node:

updateStraightlyConnectedProcess(richSolutionProcess, richSolutionProcess,

false,//boolean downstream,

true,//boolean inletPoint,

true,//boolean outletPoint,

false,//boolean updateT,

0,//double T,

false,//boolean updateP,

0,//double P,

true,//boolean updateX,

Xsr);

1. **updating and calculating the associated thermocouplers**

Refer to section 3.2.3 on external processes for a general explanation on updating thermocouplers.

The objective is for the component to send to each of its thermocouplers the equivalent values for flow rates, inlet and outlet temperature and thermal energy transferred, which they must take into account in their calculations.

The method updateThermoCoupler() of the ExtMixer or ExtDivider classes performs the updates:

updateThermoCoupler("absorber", Tabs, Tabs, Qabs, msr);

For example, here we assumed that the absorber is at temperature Tabs, that it is receiving thermal load Qabs and that it is supplied with a flow rate msr. If we had not taken the temperature of the absorber as a reference for the exchange calculations, keeping the temperatures of the steam entering and exiting the external process, we would have ended up with an unacceptable temperature crossover resulting in an error diagnosis by Thermoptim.

Unlike external processes, the update should always be done with the method updateThermoCoupler(), even for external nodes using only one thermocoupler.

1. **updating the screen of the external component**

Refer to the section on external processes.

3.3.3 Managing energy types

As in the case of external processes, it may be necessary to manage the assignment of energy types of an external node, whereas this problem does not arise for the nodes of the Thermoptim basic set.

To avoid making the existing system more cumbersome, the solution is to artificially assign the desired values to the different types of energy of the main vein of the external node (see section 3.2.3).

The method updateProcess() of the ComposantExt class can do this, provided that the main vein is a process point. Otherwise, the energy types set will be reinitialized each time a recalculation is performed.

For example the Desorber class could set its thermocoupler load as purchased energy by:

```
de.updateProcess(setEnergyTypes(richSolutionProcess,0,Qgen,0));
```

3.3.4 Saving and loading the parameters of the model

Refer to the section on external processes.

3.4 Access to instances of external nodes and processes

The preceding sections and Appendix 1 explain how to access the various unprotected instances of Thermoptim. The procedures, which are relatively cumbersome, are limited due to the internal consistency constraints of the software.

However, an external component developer may wish to have full access to the classes he creates, in order to synchronize the calculations among them. He can do this with the getExternalClassInstances() method of the Project class, which builds a Vector provided in Appendix 2, containing the various instantiated external nodes and processes, with an identifying description.

As an example, the external node of the main vein “myProcess” and of the type “myType” can be obtained by the following code:

```
MaClasse monInstance ;
Vector vExt=proj.getExternalClassInstances();
for(int i=0;i<vExt.size();i++){
Object[] obj=new Object[6];
obj=(Object[])vExt.elementAt(i);
String type=(String)obj[0];
if(type.equals("node")){
String main=(String)obj[4];
String nodeType=(String)obj[3];
if((main.equals(myProcess))&&((nodeType.equals("myType ")))){
myInstance=(MyClasse)obj[1];
myInstanceName=(String)obj[2];
}
}
}
```

Once the instance is found, its members can be accessed under Java rules, according to their access modifiers (public, protected, etc.)

3.5 External driving of Thermoptim

External driving of Thermoptim has three main applications:

- to facilitate the development of external classes by testing them as they are being defined;
- to allow one to emulate Thermoptim from other applications, such as working in client-server mode;
- to give access to all of the non-protected libraries for various purposes (perform external calculations between recalculations, guide a user in a training module, etc.).

In this volume, we focus primarily on the first two applications, Volume 4 is especially dedicated to the technological design and the study of off-design behavior, for which the drivers reveal particularly interesting because they can coordinate recalculations between different Thermoptim components, including solving coupling equations that appear for this kind of study.

To make external driving possible, the main class of the simulator, Projet, can be subclassed, and a number of non-protected methods can be overloaded to define specific calculations in addition to those of the basic set. The details of these methods are provided in the Appendix. In the examples below, the class ProjetThopt inherits from Projet, while the class Pilot is used to manage the driver.

To avoid confusion, these two classes have been defined in a separate package called “pilot”.

3.5.1 Facilitating the development of external classes

The principle consists of instantiating Thermoptim from the external class Pilot, then launching it using the appropriate methods of the class ProjetThopt. Windows users should refer to section 6 “External Class Development Environment” which explains how to use the freeware Eclipse to set up a small user-friendly working environment.

In the example below, Pilot instantiates Thermoptim using the following code; only the first two methods are necessary, and the other are given to illustrate certain possibilities:

runThopt();

loadProject();

listProcesses();

printProperties();

recalcProperties();

The mandatory methods are:

public void runThopt(){

proj=new ProjetThopt();//instantiate Thermoptim

proj.show();//display the simulator screen

proj.openEditor();//display the diagram editor

}

public void loadProject(){

String[] args=new String[4];

args[0]="complet.prj";//project file name

args[1]="complet.dia";//diagram file name

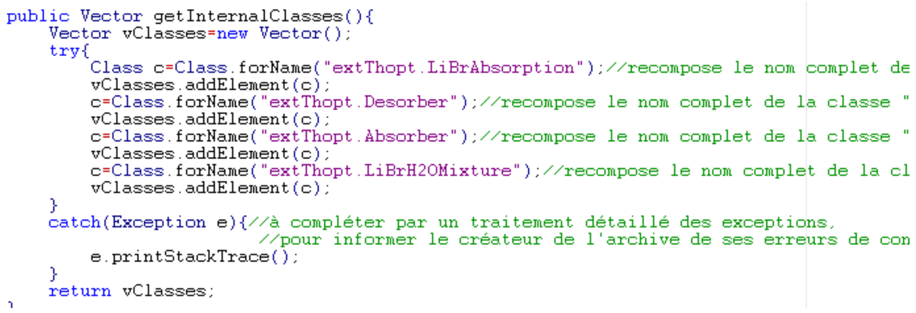
args[2]=".";//working directory

args[3]="noTest";//no save test if a project is already open

proj.loadSelection(args);//load the example defined in args[]

}

In order for the external classes under development to be taken into account in Thermoptim, they must be declared in the method getInternalClasses() of the ProjetThopt class. The figure below shows how to proceed.



```
public Vector getInternalClasses(){
    Vector vClasses=new Vector();
    try{
        Class c=Class.forName("extThopt.LiBrAbsorption");//recompose le nom complet de
        vClasses.addElement(c);
        c=Class.forName("extThopt.Desorber");//recompose le nom complet de la classe "
        vClasses.addElement(c);
        c=Class.forName("extThopt.Absorber");//recompose le nom complet de la classe "
        vClasses.addElement(c);
        c=Class.forName("extThopt.LiBrH2OMixture");//recompose le nom complet de la cl
        vClasses.addElement(c);
    }
    catch(Exception e){//à compléter par un traitement détaillé des exceptions,
                        //pour informer le créateur de l'archive de ses erreurs de con
        e.printStackTrace();
    }
    return vClasses;
}
```

The package extThopt includes a Util class that provides a number of utility methods for formatting numbers, reading them onscreen, saving and reading values, finding roots using the bisection method, etc. (see Appendix 3).

Once the classes are defined, compile them, then launch the driver by typing F5. The example defined in the method loadProject() of the Pilot class will be loaded when you click on the button "Run Thermoptim and load example" on the driver screen.

3.5.2 Providing access to the unprotected libraries

There are two ways to drive Thermoptim: either completely from an external application that instantiates and sets parameters for the software, or partially, for a given project.

In the first case, the principle consists of instantiating Thermoptim from the external class Pilot, then accessing its libraries using the appropriate methods of the class ProjetThopt as explained in the previous section.

For example, you can instantiate a substance and calculate its thermodynamic properties, or display a thermodynamic diagram and plot a pre-recorded cycle.

The driver acts as an independent application that can launch Thermoptim, which it considers as a library of external classes, and build projects by instantiating Thermoptim classes. Since it knows these instances, it can learn their state at any time. For the time being, it is impossible to access the driver from inside Thermoptim.

When the driver is launched, a window appears with buttons for launching Thermoptim and loading an example. The corresponding class (Pilot) has a few basic methods for:

launching Thermoptim: runThopt()

loading an example: loadProject()

accessing the properties of the simulator elements: proj.getProperties(args)

simplifying the parsing of the properties vectors: getSubstProperties(String nom), getPointProperties(String nom), extr_value(String s), lit_b(String s)

In the second case, a specific driver class is assigned to a given project, and this class is instantiated when the project is loaded. This class can coordinate the recalculations of the project according to specific rules. The driver class is an extension of extThopt.ExtPilot, which derives from rg.thopt.PilotFrame. To be able to link it to a project, it must first be made into an external class so that it can be loaded in Thermoptim when it is launched. Once the project is open, select the driver class linked to it via the line “Pilot frame” in the Special menu. If the project is saved, the name of the driver class is saved so that it can be instantiated when the project is loaded.

Driving Thermoptim

calcThopt(): launches a recalculation in Thermoptim

setupThopt(): instantiates the diagram editor and disables certain functions of Thermoptim

openEditor(): instantiates the diagram editor

setControls(): enables to take control between two recalculations, to modify certain parameters of Thermoptim, for example

loadSelection(String[] args): loads the examples

Accessing the simulator

String[] getHxList (): list of the exchangers of the project

String[] getNodeList (): list of the nodes of the project

String[] getPointList (): list of the points of the project

String[] getProcessList(): list of the processes of the project

String[] getSubstanceList (): list of the substances of the project

Vector getProperties(String[] args): properties Vector. Its structure, which depends on the type of element in question, is provided in the Appendix

void notifyElementCalculated(String[] element) : this method is executed in Thermoptim each time an element (point, process, node, exchanger) is calculated. Argument element is constructed as follows:

```
if(pt instanceof Transfo) element[0]="process";
else if(pt instanceof PointCorps) element[0]="point";
else if(pt instanceof Node) element[0]="node";
else if(pt instanceof HeatEx) element[0]="heatEx";
element[1]=pt.getName();
element[2]=pt.getType();
```

Thus, element can be directly used as an argument in getProperties(): getProperties(element) gives the state of the element at stake.

Another access point is the Class Corps, which has a number of public methods accessible from the driver, including:

CalcPropCorps (double T, double p, double x) : calculates the complete state of a substance

getSubstProperties() : properties Vector

a number of methods for inverting the state functions.

Accessing the diagram editor

getEditorComponentList() : provides the list of components present in the diagram editor, in the form name="componentName"+tab+type="componentType". This list can be extracted easily using extr_value("name") and extr_value("type").

getConnectionCode(String[] args) : sends a code showing whether two components of the editor are connected or not: 1 if the component 1 is upstream of component 2, -1 if it is downstream, and 0 if they are not connected. The structure of the arguments is as follows:

String[] args=new String[4];

args[0]="Expansion";//type of component 1

args[1]="turbine";//name of component 1

args[2]="Exchange";//type of component 2

args[3]="régen gaz";//name of component 2

3.5.3 Operation in client-server mode

For various reasons, it may be advantageous to use Thermoptim through a network, for educational purposes, or to associate a detailed thermodynamic model to a controller (regulation), such a model of heat pump coupled to a heating regulator.

The solution was to develop a small application based on sti.ThoptEmulator class, instantiated by sti.ThoptLauncher, which can launch a server managed by sti.ThoptServer class that manages exchanges with the outside world.

The server is launched by EmulatorServerExec.jar, which uses EmulatorServer.zip which contains the basic libraries. Both files must be placed in the Thermoptim installation directory.

Data exchange is via BufferedReader and PrintWriter, **as multiple line** String**s**, which are decoded by the client and server classes. The client is a Java class called Emulator.java, which must of course be adapted according to the needs.

The proposed solution is to write an external driver, which must implement the Emulable interface, which defines two methods **for sending and receiving** requests **(String [] getValues() and** s**etParameters (String [] params))**. The String [] params of setParameters**() and getValues**() comprise **in the** first line the number of lines **and in** the following **the** value records. ThoptServer simply passes from client to server and vice versa both String [], without modifying them.

It is also possible that the exchange of requests is done using the ModBus protocol, which, although more complex, has the advantage of being able to communicate with applications not using Java encodings that are often used in the world of control system.

3.6 Integrating external classes in the library extUser2.zip

Once the external classes are developed, they must be integrated into the library extUser2.zip so that they can be automatically recognized in the purely executable version of Thermoptim.

As indicated by its extension, file extUser2.zip is a compressed archive which must be built in a specific format. You can do it by using either a utility named "external class manager" or a standard compressor such as Winzip, as indicated in the next sections.

3.6.1 Using the external class manager

A Thermoptim functionality allows you to perform this operation: the **external class manager** is available from the menu Special in the simulator. It requires installing some files containing the libraries it needs.

Files which must be present are:

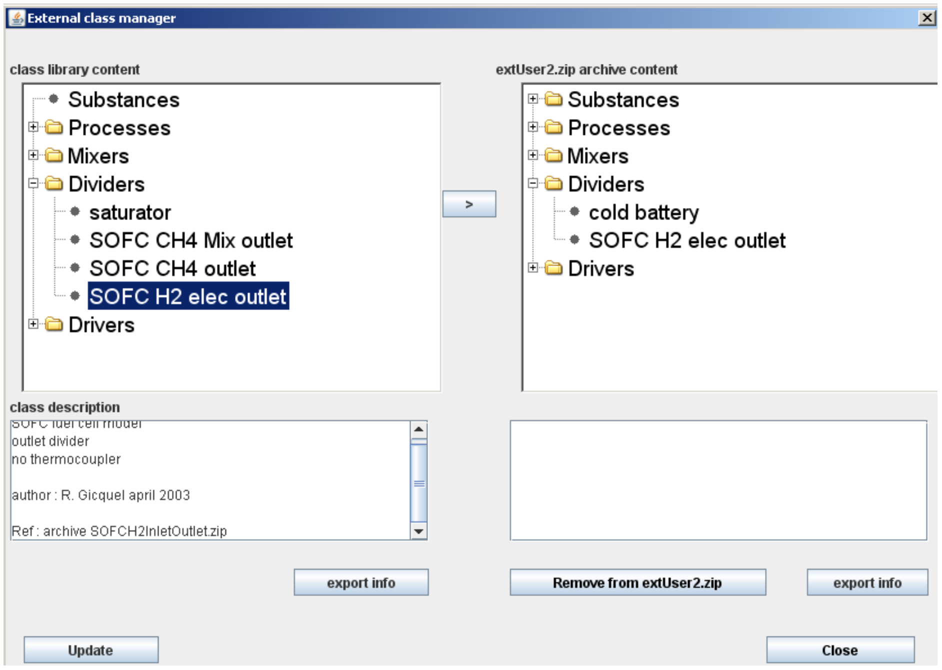
- The Extlib.ini file, which contains a single line giving the name of the path to the directory (by default "externalClassLibrary2") where the library (with a subdirectory extThopt);
- The externalClassLibrary2.zip archive where are temporarily stored by Thermoptim the library classes. Otherwise, the content extUser2.zip can only be displayed.

Place the new classes not included in the archive in the directory \externalClassLibrary2\extThopt.

If external libraries are loaded by the classes, also copy them to where they should be relative to the directory \externalClassLibrary2.

Caution: It is recommended to start by making a copy of extUser2.zip to be able to restore it in case of error. However, as the manager begins by duplicating the file extUser2.zip as extUser2_copy.zip, it is possible in case of problems start from this file by renaming it.

Launch the manager from the Special menu of the simulator. If conflicts exist between the code names assigned to classes, messages are displayed and written to the file output.txt. The screen below is displayed.



SOFC fuel cell model
outlet divider
no thermocoupler

author : R. Gicquel april 2003

Ref : archive SOFCH2InletOutlet.zip

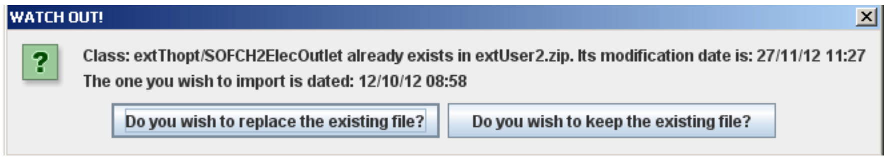
In the upper left hand part, it displays all of the classes available in the class library directory whose path is specified in the file ExtLib.ini (by default the directory "externalClassLibrary2"). These classes are arranged by type (substances, external processes, external mixtures, external dividers, drivers). On the right hand side, the content of the extUser2.zip archive is displayed, arranged in the same manner. Thus it is easy to compare the two sets of classes.

If you select one class in either of the lists, a description appears in the window below. You can also select multiple classes, in which case nothing is displayed.

You can transfer a single or multiple selection from the class library (left) to the archive extUser2.zip by clicking on the small central arrow

.

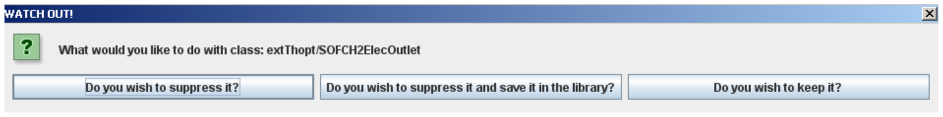
If a class selected already exists in the archive, a message warns you, recalling the dates of creation of both classes, and asks if you want to replace the existing one, or keep it.


WATCH OUT!

Class: extThopt/SOFCH2ElecOutlet already exists in extUser2.zip. Its modification date is: 27/11/12 11:27
The one you wish to import is dated: 12/10/12 08:58

Do you wish to replace the existing file?
Do you wish to keep the existing file?

You can remove one or more classes from the archive extUser2.zip by clicking on the button "Remove from extUser2.zip". A message asks you to confirm the deletion for each class selected, with the option to save the class in the class library. If you want to save a class that exists already in the library, a message is displayed showing the creation dates of the two classes, and asking you if you want to replace or keep the existing one.


WATCH OUT!

What would you like to do with class: extThopt/SOFCH2ElecOutlet

Do you wish to suppress it?
Do you wish to suppress it and save it in the library?
Do you wish to keep it?

With the two buttons called “export info”, you can create text files called "zipList.txt" or "libraryList.txt" which contain the name, type and description of each class.

When the left and right windows are loaded, a test is performed to check that the classes are not of the same type, which could cause errors in Thermoptim. If they are the same type, you must differentiate them and recompile the modified classes.

Finally, the button “update” updates the two windows once all transfers and removals have been completed, this operation not being automatically done.

If you have modified the file extUser2.zip, it is strongly recommended that you close and then reopen Thermoptim because libraries have been modified and Thermoptim may need to be reset.

Attention to related external classes (including superclasses): it is imperative to add or remove them all, otherwise Thermoptim may not be able to start correctly (the problem would of course be the same using a standard compactor as shown in the Volume 3 of the Thermoptim reference manual). However, if you use classes other than those recognized Thermoptim, you should include them in extUser2.zip manually, because this process cannot be automated.

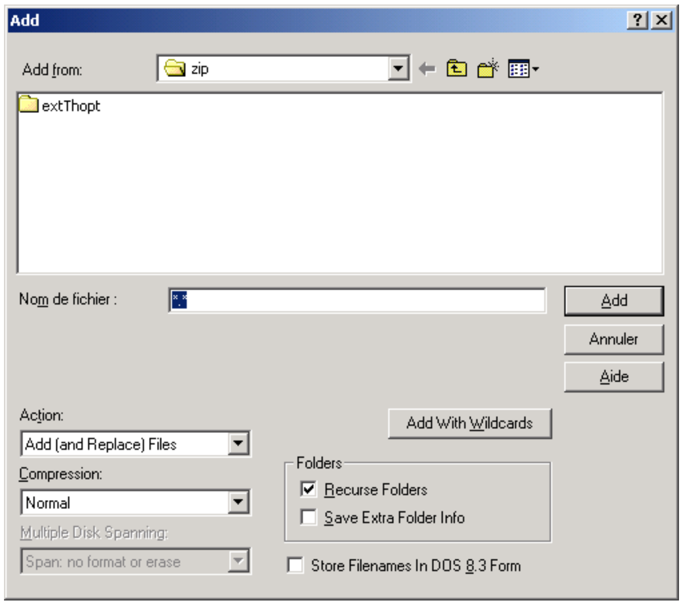
If Thermoptim does not start after you have changed extUser2.zip, open the file error.txt, which generally contains information on problems which have occurred.

Lastly, note that the external class viewer analyzes both extThopt2.zip and extUser2.zip files, while the manager only analyzes extUser2.zip.

You can transfer by hand extThopt2.zip external classes in the library for transferring them in extUser2.zip, then delete them from extThopt2.zip, but this is not recommended.

3.6.2 Using a standard compressor

It is also possible to use a standard compressor such as WinZip by adding the new class in the archive. However, problems have been reported with certain versions of WinZip.



Add from: zip

extThopt

Nom de fichier : *.*

Action: Add (and Replace) Files

Compression: Normal

Multiple Disk Spanning: Span: no format or erase

[✓] Recurse Folders

[ ] Save Extra Folder Info

[ ] Store Filenames In DOS 8.3 Form

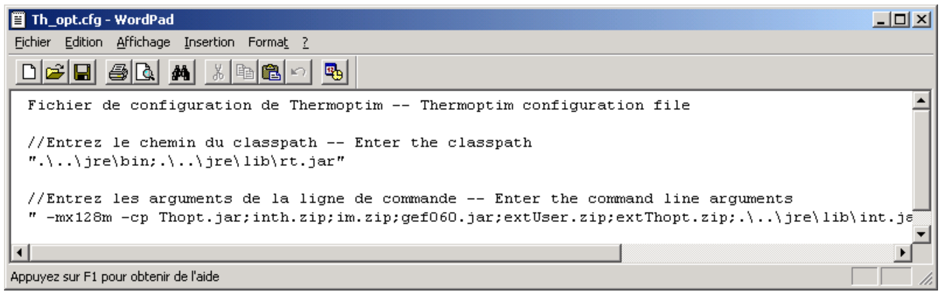
With WinZip, you can proceed as follows:

- create a new directory and call it “zip”, then create a sub-directory in that directory called “extThopt”;
- copy all the classes that you want to load in extUser.zip or extUser2.zip into this directory (the classes are located in the class creation directory of your development environment (/extThopt/);
- Don’t forget to quit Thermoptim, otherwise you will not be able to write to the archive extUser.zip or extUser2.zip;
- Open extUser.zip or extUser2.zip, click on “Add” and go to the directory “zip”. Select the option “Recurse folders” then click on “Add With Wildcards”, which will add the entire contents of the sub-directory extThopt (the file path to the classes in the archive must be correct, otherwise Thermoptim will not be able to load them).

If you encounter difficulties, send an e-mail to contact@s4e2.com explaining your problem, and attach the class you want to add to extUser2.zip. The new archive will be returned to you as soon as possible.

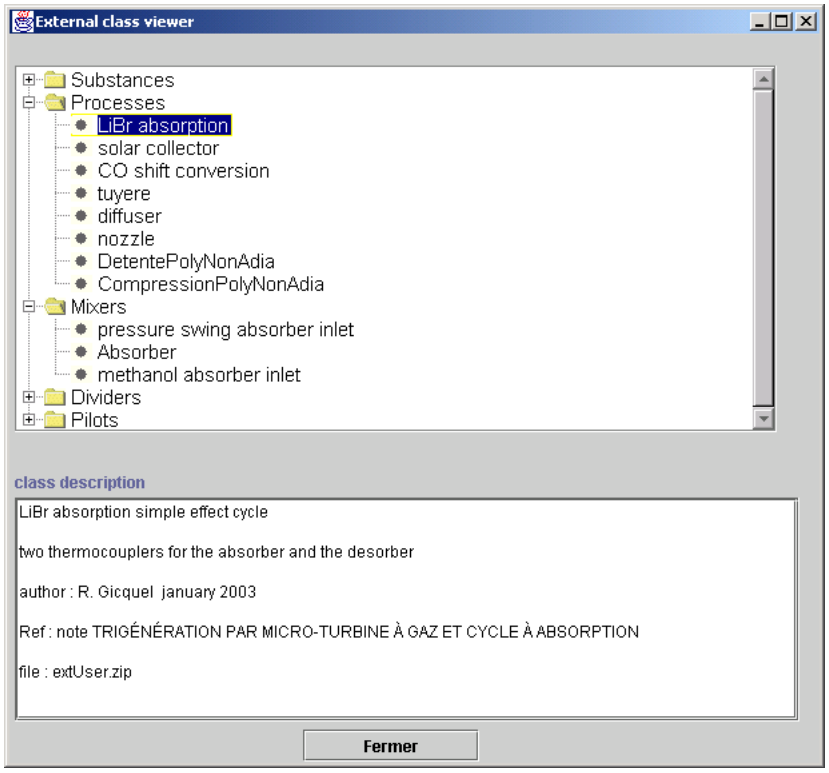
3.6.3 Modifying the classpath for adding java libraries

If you want to add other Java libraries that your classes call, you have to indicate their file path in the classpath. In Windows, open the Thermoptim configuration file “Thopt.cfg” in a text editor, and add their file paths after “-cp”, separated by semicolons. Use a similar procedure for Unix and Macintosh.



```
Fichier de configuration de Thermoptim -- Thermoptim configuration file

//Entrez le chemin du classpath -- Enter the classpath
".\..\jre\bin;.\..\jre\lib\rt.jar"

//Entrez les arguments de la ligne de commande -- Enter the command line arguments
" -mx128m -cp Thopt.jar;inth.zip;im.zip;gef060.jar;extUser.zip;extThopt.zip;.\..\jre\lib\int.ja
```

3.6.4 Viewing external classes



LiBr absorption simple effect cycle

two thermocouplers for the absorber and the desorber

author : R. Gicquel january 2003

Ref : note TRIGÉNÉRATION PAR MICRO-TURBINE À GAZ ET CYCLE À ABSORPTION

file : extUser.zip

To help you use and manage the external classes, the line External Class Viewer from the Special menu of the simulator displays all of the external classes available. They are sorted by type (substances, processes, mixers, dividers, drivers) with a short description of the class selected and where it comes from (extThopt.zip and extUser2.zip archives as well as classes under development).

This screen can be consulted while you are developing your model. The more careful you are in writing the description of your classes, the more helpful this screen will be to users. Remember that it is very important to document your classes thoroughly, and you can include the documentation reference in your description. The documentation is composed of two parts, one for users and one for class designers. A poorly documented class will be difficult to use or modify.

4 External class development environment

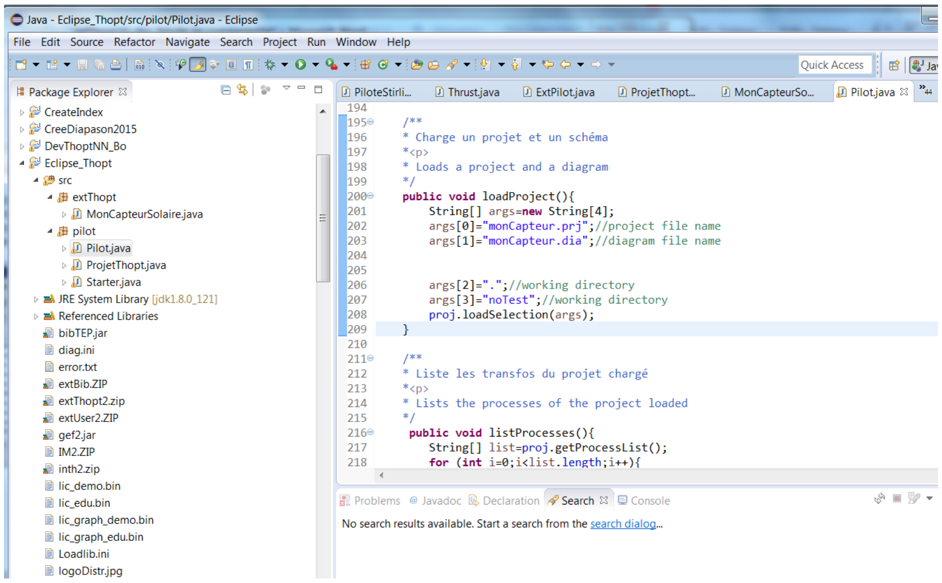
To develop external classes, if you do not already have a Java development environment and if you work in Windows, you can use a freeware application called Eclipse. Developed as open source, Eclipse can be downloaded from http://www.Eclipse.org.

With Eclipse, you can write your classes in Java, and compile and test their integration in Thermoptim. To make things easier, a start-up workspace is provided. It is called Eclipse_Thopt.

4.1 Presentation of the workspace

The workspace consists of the following:

- in the left part of the screen there is a list of Java files, including a small class called Starter, which launches the driver, and the classes under development (external files).
- in the right part you have the various tabs containing the code of the classes.
- the bottom window is for messages.



```
    /**
    * Charge un projet et un schéma
    *<p>
    * Loads a project and a diagram
    */
    public void loadProject(){
        String[] args=new String[4];
        args[0]="monCapteur.prj";//project file name
        args[1]="monCapteur.dia";//diagram file name


        args[2]=".";//working directory
        args[3]="noTest";//working directory
        proj.loadSelection(args);
    }

    /**
    * Liste les transfos du projet chargé
    *<p>
    * Lists the processes of the project loaded
    */
     public void listProcesses(){
        String[] list=proj.getProcessList();
        for (int i=0;i<list.length;i++){
```

No search results available. Start a search from the search dialog...

4.2 Installing Eclipse

To install the development environment, log on to the Eclipse website, and follow the instructions.

Explanations are given in the **Thermoptim-UNIT portal**. However our explanations can only be succinct and related to the particularities of Thermoptim.

Given the potential of Eclipse, it is a widely used environment, and you can easily find sites explaining in detail how to install it for various operating systems.

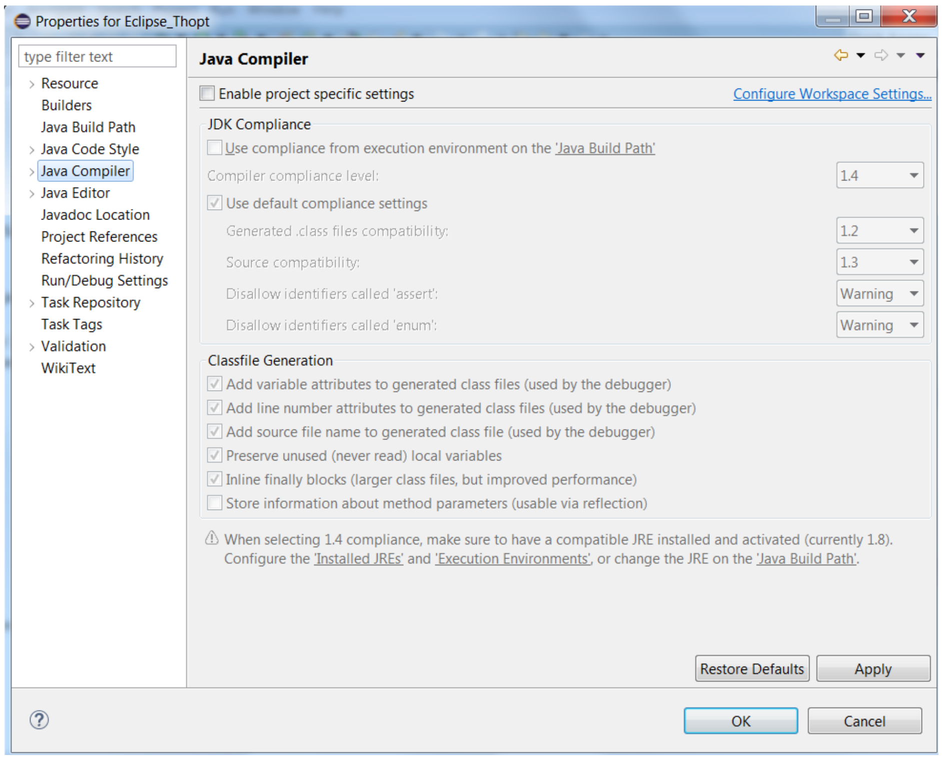
In what follows, we assume that the reader knows some basics of the Java language. If this is not the case, we suggest that he refer to an introductory book before attempting to use the development environment.

Start by installing the JDK and its documentation, then Eclipse. When you run Eclipse for the first time, the software suggests that you associate the extensions of certain files with it. Click "OK" unless you have a good reason not to make that choice. Then choose the directory paths containing the JDK and its documentation.

Unpack the Thopt25_EclipseDemo.zip file and place the Eclipse_Thopt folder in the Eclipse installation directory. This is a Workspace with all the files you need to work with Thermoptim. Optionally add the ThoptEdu2.jar or Thopt2.jar archive of Thermoptim and inth2.zip, as well as your license files (all these files are in the Thermoptim installation directory).

For the project to run properly, it must be fully configured. Open the Projects / Properties menu and then Java Compiler. You get a screen that looks like the one below on the left. Check which JDKs appear and which one is checked.

Then you need to specify the libraries required by your application, in this case all of the ones Thermoptim needs to work, plus Thermoptim itself. To do this, open the Projects / Properties menu, then Java Build Path, and then Libraries. If the libraries of the figure do not appear, click on "Add JARs", and select in the directory "Eclipse_Thopt" the files that appear in the screen below on the right: ThoptEdu2.jar or Thopt2.jar, extUser2. zip, im2.zip, inth2.zip, extThopt2.zip, gef.jar. Once the archives are selected, click on "OK". You can now start developing your external classes as described in section 3.



Properties for Eclipse_Thopt

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Java Compiler

☐ Enable project specific settings — Configure Workspace Settings...

JDK Compliance

☐ Use compliance from execution environment on the 'Java Build Path'

Compiler compliance level:	1.4

☑ Use default compliance settings

Generated .class files compatibility:	1.2
Source compatibility:	1.3
Disallow identifiers called 'assert':	Warning
Disallow identifiers called 'enum':	Warning

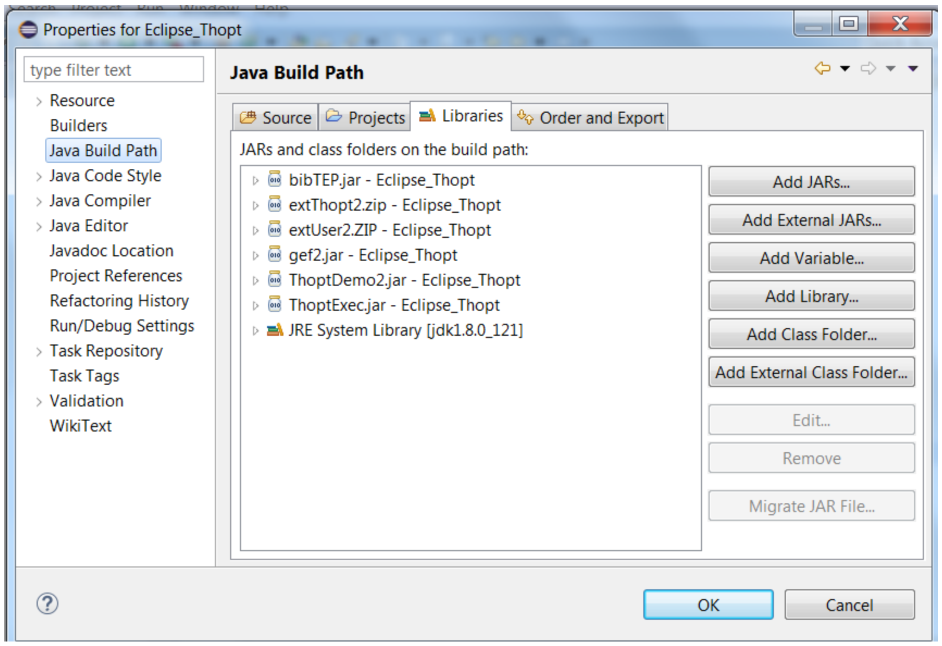
Classfile Generation

☑ Add variable attributes to generated class files (used by the debugger)
☑ Add line number attributes to generated class files (used by the debugger)
☑ Add source file name to generated class file (used by the debugger)
☑ Preserve unused (never read) local variables
☑ Inline finally blocks (larger class files, but improved performance)
☐ Store information about method parameters (usable via reflection)

⚠ When selecting 1.4 compliance, make sure to have a compatible JRE installed and activated (currently 1.8).
Configure the 'Installed JREs' and 'Execution Environments', or change the JRE on the 'Java Build Path'.

Restore Defaults | Apply

OK | Cancel



Properties for Eclipse_Thopt

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- bibTEP.jar - Eclipse_Thopt
- extThopt2.zip - Eclipse_Thopt
- extUser2.ZIP - Eclipse_Thopt
- gef2.jar - Eclipse_Thopt
- ThoptDemo2.jar - Eclipse_Thopt
- ThoptExec.jar - Eclipse_Thopt
- JRE System Library [jdk1.8.0_121]

Add JARs... | Add External JARs... | Add Variable... | Add Library... | Add Class Folder... | Add External Class Folder... | Edit... | Remove | Migrate JAR File...

OK | Cancel

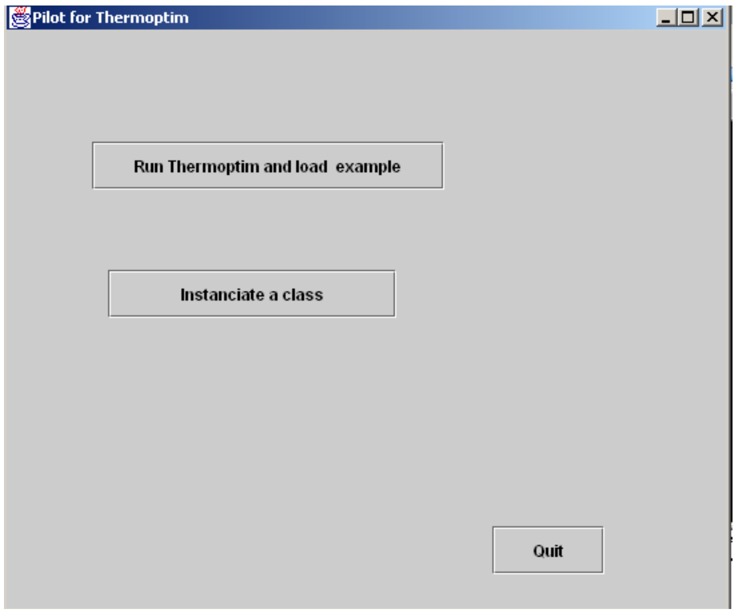
The source code of the base classes of the package extThopt is given in the form of an archive. These classes are already contained in the archive extThopt.zip and should not be loaded in the development environment. You can modify them, but they may no longer be compatible with the “official” versions of the Thermoptim external classes.

Appendix 1 describes how they are used and gives a number of examples. Appendix 2 gives the code for the Thermoptim methods that return Vectors whose details it is useful to know for programming external classes. Appendix 3 contains a brief description of the methods in the Util class.

4.3 Emulating Thermoptim from Eclipse

To be able to emulate Thermoptim in order to test a class under development, we created a special workspace in which three classes that we present here play a key role:

- Starter is the class launched when Eclipse starts up. It initializes the output files ("output.txt" and "error.txt") and instantiates the Pilot class of the pilot package;
- Pilot is the class that launches Thermoptim, instantiating the class ProjetThopt of the same package and running the method runThopt(). Next, the method loadProject() displayed on the screen of the figure presenting the workspace loads a project and a diagram for testing the external classes under development. In the Eclipse workspace figure, we have entered different examples, since only the last one loaded in the array args[] is taken into account (this enables us to change examples easily, by moving the two lines involved);
- ProjetThopt, which inherits from the class Projet of Thermoptim, has a method called getInternalClasses() in which the external classes to be loaded in Thermoptim are defined.



Once the methods loadProject() and getInternalClasses() have been modified, simply recompile the project and run it so that the load window of Thermoptim and its project appear.

By clicking on “Run Thermoptim and load example”, you load the example you want. You can use all the features of the software, including the ability to modify and save your example. If a project or diagram file is missing, a message will alert you.

In this way, you can develop your class and directly test it in Thermoptim, which will save a lot of time. Once the class is developed simply archive it in extUser2.zip so that it can be distributed and used outside the development environment.

Practically speaking, the procedure is as follows:

- Following the instructions given in chapter 3, start by creating your external class, either by modifying one of the ones provided (the java files are in the directory “src” of the Workspace) or from the Eclipse assistant (menu Project/New Class), and compile it (menu Build/Compile file)
- Declare your class in the method getInternalClasses() of the class ProjetThopt, following the instructions provided, so that it is loaded in Thermoptim when the software is loaded, and recompile ProjetThopt
- To test your class, duplicate a project and diagram example, enter the new names on the last lines of the array args[] in the method loadProject() of Pilot, and recompile Pilot.
- Launch execute (menu Build/Execute Project, or F5) and click on “Run Thermoptim and load example”

The screen of the Pilot class has another button allowing you to instantiate another class if you want to. Simply write the corresponding code in the method bOtherClass_actionPerformed().

Appendix 1 : Thermoptim methods which can be called by external classes

In this appendix, we present Thermoptim methods which can be called by external classes, indicating what is their purpose and giving a number of implementation examples. Their API is itself defined in the JavaDoc that is in the directory api_Thermoptim of the development environment.

Package rg.corps

Two class names are not obfuscated: rg.corps.Corps, mother class of all substances and rg.corps.CorpsExt, mother class of all external substances.

The following methods can be called by external classes:

In Corps

To calculate a substance state

public void CalcPropCorps (double T, double p, double x)

To instantiate a substance whose name is known

public static Object createSubstance(String nomCorps)

example :

Object obj=Corps.createSubstance("eau");//instantiation of the substance, wrapped in an Object

Corps leau=(Corps)obj;//transtyping of the Object

double T=leau.getT_from_hP(3400,50);// Reverse calculation inT of the enthalpy the pressure being known

System.out.println("T eau : "+T);

To get the Cp of a substance knowing its Cv

public double getCp(double Cv)

Reverse calculation of state functions

public double getP_from_hT(double hv,double T)

public double getP_from_sT(double sv,double T)

public double getP_from_sT(double sv,double T, double pmin, double pmax)

public double getP_from_vT(double v,double T)

public double getT_from_hP(double hv,double P)

public double getT_from_sP(double sv,double P)

public double getT_from_sv(double s,double v)

public double getT_from_uv(double u,double v)

public double[] getQuality(), used with all previous methods, gives the the vapor-liquid equilibrium composition. In particular, for pure substances, getQuality() [0] is the steam quality.

Methods for calculating the saturation pressure or temperature

public double getSatPressure(double T, double x)

public double getSatTemperature(double P, double x)

To get a substance state

public Vector getSubstProperties()

method getSubstProperties() of extProcess indirectly uses this method to load the values of the double Tsubst,Psubst,Xsubst,Vsubst,Usubst,Hsubst,Ssubst,Msubst,typeSubst;

example :

lecorps.CalcPropCorps(Tpoint+1, Ppoint, Xpoint);//calculates the substance state

getSubstProperties(nomCorps);// retrieves the calculated values and loads them in particular in Hsubst

double Cp=(Hsubst-H);

typeSubst is 1 for water, 2for steam, 3 for a pure gas, 4 for a compound gas, 5 for a single external substance, and 6 for an external substance of the Mixture type.

Initialization of an external substance

public void initCorpsExt(double M, double PC, double TC, double VC,

double Tmini, double Tmaxi, double Pmini, double Pmaxi, int typeCorps)

Defines a comment for an external substance (chemical formula, composition ...)

public void setComment(String comment)

Gives a name to an external substance

public void setNom(String name)

Loads values of calculations made by the external substance

public void setState(double P, double T, double xx,

double U, double H, double S, double V, double Cv,

double Xh)

To get a gas composition

public Vector getGasComposition()

example :

getPointProperties(cO2Point);// retrieves the properties of point cO2Point

Vector nouvComp=lecorps.getGasComposition();//retrieves the gas composition

updateGasComp(nouvComp, cO2Process);//updates the gas composition

To update a gas composition

public void updateGasComp(Vector vComp)

The Vector structure is as follows:

```
public void updateGasComp(Vector vComp){
Integer i=(Integer)vComp.elementAt(0);
int nComp=i.intValue();
String[] Comp= new String[nComp];
double[]fractmol= new double[nComp],fractmass= new double[nComp];
Double[]fracMol= new Double[nComp],fracMass= new Double[nComp];
Comp=(String[])vComp.elementAt(1);
fracMol=(Double[])vComp.elementAt(2);
fracMass=(Double[])vComp.elementAt(3);
for(int j=0;j<nComp;j++){
Double f=(Double)fracMol[j];
fractmol[j]=f.doubleValue();
f=(Double)fracMass[j];
fractmass[j]=f.doubleValue();
}
updateSubstComp(Comp,fractmol,fractmass);
```

}

In CorpsExt

public double getLiquidConductivity(double T)

public double getLiquidViscosity(double T)

public double getVaporConductivity(double T)

public double getVaporViscosity(double T)

Package rg.thopt

The following class names are not obfuscated:

- rg.thopt.Projet, simulator class,
- rg.thopt.TransfoExterne, rg.thopt.DividerExterne, rg.thopt.MixerExterne et rg.thopt.ComposantExt, used to define the external components
- rg.thopt.Compression and rg.thopt.ComprExt, used to define the external compressors.

The following methods can be called

In Projet

Launches Thermoptim recalculation from the driver

public void calcThopt()

returns a code that allows the driver to know if two components of the editor are connected or not:

public int getConnectionCode(String[] args)

1 if component 1 is the upstream component 2, -1 otherwise, and 0 if not connected.

The argument structure is the following :

String[] args=new String[4];

args[0]="Expansion";//type of component 1

args[1]="turbine";//name of component 1

args[2]="Exchange";//type of component 2

args[3]="régen gaz";// name of component 2

Lists the components in the diagram editor

public String[] getEditorComponentList()

edited as name="nomComposant"+tab+type="typeComposant". This list can be broken easily with extr_value("name") and extr_value("type")

Lists of elements

public String[] getHxList ()

public String[] getNodeList ()

public String[] getPointList ()

public String[] getProcessList()

public String[] getSubstanceList ()

Vector of properties whose structure depends on the type of element considered

public Vector getProperties(String[] args)

type=args[0]; "subst" / "point" / "process" / "heatEx"

nomType=args[1];

This method is fundamental: it allows access to all data related to a primitive type once one the type and name known. It is widely used by generic methods of extThopt package.

The corresponding Thermoptim internal code is given in annex 2. It lets you know the structure of the Vector returned (depending on the type of primitive called).

Loading of examples by the Thermoptim driver

public void loadSelection(String[] args)

method executed in Thermoptim each time an item (point, process, node, exchanger) is calculated

public void notifyElementCalculated(String[] element)

element is constructed as follows:

if(pt instanceof Transfo) element[0]="process";

else if(pt instanceof PointCorps) element[0]="point";

else if(pt instanceof Node) element[0]="node";

else if(pt instanceof HeatEx) element[0]="heatEx";

element[1]=pt.getName();

element[2]=pt.getType();

Thus, it can directly be used as an argument in getProperties(): getProperties(element) provides the state of the element.

allows the driver to take control between two recalculations, to modify certain Thermoptim parameters

public void setControls()

instantiates the diagram editor and used to disable certain functions by the Thermoptim driver

public void setupThopt()

instantiates the diagram editor

public void openEditor()

loading external classes in the Thermoptim driver

public Vector getInternalClasses()

Operations on the charts from the Thermoptim driver

public void setupChart (Vector v)

example of opening chart, choice of substance and drawing of two cycles

if(proj!=null)proj.calcThopt();//if necessarye instantiation of Thermoptim

Vector v=new Vector();

v.addElement("openDiag");//opening of a chart

v.addElement("1");// chart type (1 for vapors, 2 for ideal gas, 3 for psychrometric)

proj.setupChart(v);

v=new Vector();

v.addElement("selectSubstance");//selection of a substance

v.addElement("R134a");//name of the substance

proj.setupChart(v);

v=new Vector();

v.addElement("readCycle");//loading a cycle

v.addElement("frigoR134aFin.txt");//file name

proj.setupChart(v);

v=new Vector();

v.addElement("readCycle");// loading a cycle

v.addElement("frigoR134a.txt");// file name

proj.setupChart(v);

updates a point or a process

public void updatePoint(Vector v)

example : method updateStraightlyConnectedProcess() of ExtNode includes:

Vector vPoint=new Vector();

vPoint.addElement(outletPointName);

vPoint.addElement(Util.aff_b(updateT));

vPoint.addElement(Util.aff_d(T));

vPoint.addElement(Util.aff_b(updateP));

vPoint.addElement(Util.aff_d(P));

vPoint.addElement(Util.aff_b(updateX));

vPoint.addElement(Util.aff_d(x));

proj.updatePoint(vPoint);

In PilotFrame

To get a handle on the Projet calling

public void getProjet ()

In ComposantExt

To calculate the component

public void calcProcess()

To get the component’s name

public String getCompName()

To get the component’s type

public String getCompType()

To read the component settings saved in the project file

public void readCompParameters(String ligne_data)

To save the component settings in the project file

public String saveCompParameters()

To initialize the component screen and its internal logical configuration

public void setCompFrame(Object obj)

public void setDivFrame (Object obj)

public void setMixFrame (Object obj)

to update the process (flow, inlet point, outlet point, energy types)

public void setupFlow(double flow)

public void setupPointAmont(Vector vProp)

public void setupPointAval (Vector vProp)

public void updateProcess(Vector vEner)

To construct the Vector vProp, ExtProcess has a generic method, which wraps the point state:

public Vector getProperties(){

```
Vector vProp=new Vector();
vProp.addElement(lecorps);//Corps
vProp.addElement(nomCorps);//Corps
vProp.addElement(new Double(Tpoint));
vProp.addElement(new Double(Ppoint));
vProp.addElement(new Double(Xpoint));
vProp.addElement(new Double(Vpoint));
vProp.addElement(new Double(Upoint));
vProp.addElement(new Double(Hpoint));
return vProp;
}
```

example :

```
tfe.setupPointAval(getProperties());
```

method updateProcess() of ComposantExt can assign values to different types of energy. It is easily implemented with the method setEnergyTypes() ExtThopt (see Section 3.2.3).

example :

```
tfe.updateProcess(setEnergyTypes(useful,purchased,other));
```

In TransfoExterne

To update a thermocoupler

```
public void updateThermoCouplers(Vector vTC)
```

To construct the Vector vTC, ExtProcess has a generic method, which wraps the values:

```
protected void updateThermoCoupler(String type, double Tin, double Tout, double Q, double flow){
Vector vTC=new Vector();
vTC.addElement(type);//type du thermocoupleur considéré
Double d=new Double(Tin);
vTC.addElement(d);
d=new Double(Tout);
vTC.addElement(d);
d=new Double(Q);
vTC.addElement(d);
d=new Double(flow);
vTC.addElement(d);
tfe.te.updateThermoCouplers(vTC);
}
```

Provides the values of the thermocoupler of the type called

```
public Vector getThermoCouplerData(String thermoCouplerType)
```

The Vector returned contains only for the moment the thermocoupler name, but this should evolve.

In DividerExterne

Updates the divider

```
public void updateDivider (Vector vTC)
```

To construct the Vector vTC, ExtProcess has a generic method, which wraps the values:

```
protected Vector getUpdateVector (Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal){
Vector vTC=new Vector();
for(int j=0;j<vTransfo.length;j++){
vTC.addElement(vTransfo[j]);
}
for(int j=0;j<vPoints.length;j++){
vTC.addElement(vPoints[j]);
}
Vector vGlobal=new Vector();
vGlobal.addElement(new Double(TGlobal));
vGlobal.addElement(new Double(hGlobal));
vTC.addElement(vGlobal);
de.de.updateDivider(vTC);
}
```

In ExtDivider the call is made as follows :

```
protected void updateDivider(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal){
Vector vTC=getUpdateVector(vTransfo,vPoints, TGlobal, hGlobal);
de.de.updateDivider(vTC);
}
```

These methods make use of tables of Vector vTransfo and vPoint representing the main vein and branches, which may be useful to automate the construction, as in the Desorber class, where generic methods called by a name with a physical sense are used to build the Vector:

```
private void setupRichSolution(double m, double T, double P, double X){
vTransfo[0]=new Vector();
vPoints[0]=new Vector();
vTransfo[0].addElement(richSolutionProcess);
vTransfo[0].addElement(new Double(m));
vPoints[0].addElement(richSolutionPoint);
vPoints[0].addElement(new Double(T));
vPoints[0].addElement(new Double(P));
vPoints[0].addElement(new Double(X));
}
```

example : update of external divider Desorber after calculation :

```
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupRichSolution(msr,Tsr,Psr,Xsr);
setupPoorSolution(msp,Tgen,P,Xsp);
setupRefrigerant(mr,Trefr,Prefr,1);
updateDivider(vTransfo,vPoints,Tsr,Hsr);
```

Gives the divider values

public Vector getDividerData()

Currently not used, because duplicating getProperties() Project executed for a node. ExtDivider it provides method public void getDividerStructure() which loads the structure of the node.

To update a thermocoupler (see TransfoExterne)

public void updateThermoCouplers(Vector vTC)

Provides the values of the thermocoupler of the type called (see TransfoExterne)

public Vector getThermoCouplerData(String thermoCouplerType)

In MixerExterne

Updates the mixer (see DividerExterne)

public void updateMixer (Vector vTC)

Gives the mixer values (see DividerExterne)

public Vector getMixerData ()

Pour mettre à jour un thermocoupleur (voir TransfoExterne)

public void updateThermoCouplers(Vector vTC)

Provides the values of the thermocoupler of the type called (see TransfoExterne)

public Vector getThermoCouplerData(String thermoCouplerType)

In ComprExt

public double getComprFlow()

public double getComprIsentropicEfficiency()

public String getComprName()

public double getComprRatio()

public double getComprRelaxValue()

public double getComprRotationSpeed()

public double getComprSweptVolume()

public String getComprType()

public double getComprVolumetricEfficiency()

public double getComprVolumetricFlow()

public double getPumpFanVolumetricFlow()

public Vector getSupplyPointProperties()

public void makeComprDesign()

public void readComprParameters(String line)

public String saveComprParameters()

public void setComprFrame(Object ob)

Annex 2 : Thermoptim methods code

We give here the code of certain Project and Corps methods whose structure must be faithfully reproduced if they are to be used in subclasses.

Method getProperties() of Projet

/**

- Fournit les valeurs thermodynamiques des divers types primitifs

- <p>
- Gives thermodynamic values for the various element
- @param args String[]
- @param type=args[0]; "project" / "subst" / "point" / "process" / "heatEx"
- @param nomType=args[1];
- */

```
public Vector getProperties(String[] args){
String type, nomType;
type=args[0];
nomType=args[1];
Vector vProp=new Vector();
if(type.equals("project")){
vProp.addElement(Util.unit_m);//unité des débits [0]
vProp.addElement(Util.unitT);//unité des températures[1]
vProp.addElement(new Double(Util.T0Exer)); // température de l'environnement[2]
vProp.addElement(new Double(getEnergyBalance()[0]));//énergie payante [3]
vProp.addElement(new Double(getEnergyBalance()[1])); //énergie utile [4]
vProp.addElement(new Double(getEnergyBalance()[2])); //efficacité [5]
}
else if(type.equals("subst")){
Corps pt=getCorps(nomType);
if(pt!=null){
vProp=pt.getSubstProperties();
}
}
else if(type.equals("point")){
PointCorps pt=getPoint(nomType);
if(pt!=null){
vProp.addElement(pt.lecorps);//Substance [0]
vProp.addElement(pt.lecorps.getNom());//Substance name [1]
vProp.addElement(new Double(pt.getT()));//Temperature [2]
vProp.addElement(new Double(pt.getP()));//Pressure [3]
vProp.addElement(new Double(pt.getXx()));//Quality [4]
vProp.addElement(new Double(pt.getV()));//Volume [5]
vProp.addElement(new Double(pt.getU()));//Internal energy [6]
vProp.addElement(new Double(pt.getH()));//Enthalpy [7]
vProp.addElement(new Double(pt.getS()));//Entropy [8]
String setTsat="set_Tsat="+Util.aff_b(pt.JCheckSetTsat.isSelected());
vProp.addElement(setTsat);//setTsat [9]
vProp.addElement(new Double(pt.dTsat_value.getValue()));//Dtsat [10]
```

```
String setpsat="set_psat="+Util.aff_b(pt.JCheckSetPsat.isSelected());
vProp.addElement(setpsat);//setpsat [11]
//wet gas values
vProp.addElement(new Double(pt.w_value.getValue()));//specific humidity [12]
vProp.addElement(new Double(pt.epsi_value.getValue()));//relative humidity [13]
vProp.addElement(new Double(pt.qprime_value.getValue()));//specific enthalpy [14]
vProp.addElement(new Double(pt.tprime_value.getValue()));//adiabatic temperature [15]
vProp.addElement(new Double(pt.tr_value.getValue()));//dew point temperature [16]
vProp.addElement(new Double(pt.v_spec_value.getValue()));//specific volume [17]
vProp.addElement(new Double(pt.cond_value.getValue()));//condensates [18]
vProp.addElement(new Double(pt.lecorps.M_sec));//Dry gas molar mass [19]
vProp.addElement(new Double(pt.xhprime_value.getValue()));//specific exergy [20]
vProp.addElement("dummy");//réserved if necessary for wet gases [21]
//pressure setting
if(pt.JCheckPvalue.isSelected()){
PointCorps pt2=(PointCorps)pt;
vProp.addElement("pressureSet="+pt2.thePres.getName());//set pressure [22]
vProp.addElement(new Double(pt2.pressCorrFact_value.getValue()));//correction factor [23]
}
else {
vProp.addElement("pressureSet=");// [22]
vProp.addElement(new Double(1.));// [23]
}
}
}
else if(type.equals("process")){
Transfo tf=getTransfo(nomType);
vProp.addElement(tf.getType());//[0]
if(tf!=null){
vProp.addElement(tf.getPointAmont().getName());// [1]
vProp.addElement(tf.getPointAval().getName());// [2]
vProp.addElement(new Double(tf.getFlow()));//flow rate [3]
vProp.addElement(new Double(tf.DeltaH));//Enthalpy [4]
vProp.addElement(tf.ener_type_value.getText());//Energy type [5]
String direct="calcDirect="+Util.aff_b(tf.IcalcDirect);
vProp.addElement(direct);//true if direct calculation [6]
String ouvert="openSyst="+Util.aff_b(tf.JCheckOuvert.isSelected());
vProp.addElement(ouvert);//true for open system [7]
String setflow="setFlow="+Util.aff_b(tf.JCheckSetFlow.isSelected());
vProp.addElement(setflow);//true for set flow [8]
```

```
String inletProcess="null";
if(tf.isInletStrConnected())inletProcess=tf.getTransfoAmontName();//inlet process name (if exits)
vProp.addElement(inletProcess);// [9]
String outletProcess="null";
if(tf.isOutletStrConnected())outletProcess=tf.getTransfoAvalName();//outlet process name (if exits)
vProp.addElement(outletProcess);// [10]
if(tf instanceof ComprExpan){
ComprExpan cb=(ComprExpan)tf;
vProp.addElement(new Double(cb.rendIs));// rendement isentropique [11]
vProp.addElement(new Double(cb.Q_value.getValue()));// chaleur apportée (si non adiabatique) [12]
}
if(tf instanceof Combustion){
Combustion cb=(Combustion)tf;
String fuel="null";
if(cb.getFuel()!=null)fuel=cb.getFuel().getName();
vProp.addElement(fuel);// [11]
vProp.addElement(new Double(cb.lambda_value.getValue()));//lambda [12]
vProp.addElement(new Double(cb.Tfluegas));//combustion temperature [13]
vProp.addElement(new Double(cb.tfig_value.getValue()));//quenching temperature [14]
vProp.addElement(new Double(cb.t_diss_value.getValue()));//dissociation rate [15]
String calcLambda="calcLambda="+Util.aff_b(cb.JCheckCalcLambda.isSelected());
vProp.addElement(calcLambda);//true if calculate lambda set [16]
String calcT="calcT="+Util.aff_b(cb.JCheckCalcT.isSelected());
vProp.addElement(calcT);//true if calculate T set [17]
String setFuelFlow="setFuelFlow="+Util.aff_b(cb.JCheckFuelFlow.isSelected());
vProp.addElement(setFuelFlow);//true if calculate with set fuel flow [18]
String dissoc="setDissoc="+Util.aff_b(cb.Check_dissoc.isSelected());
vProp.addElement(dissoc);//true if dissociation set [19]
String premix="setPremix="+Util.aff_b(cb.JCheckPremix.isSelected());
vProp.addElement(premix);//true if premix set [20]
String setV="setV="+Util.aff_b(cb.JChecksetV.isSelected());
vProp.addElement(setV);//true if set volume [21]
String setP="setP="+Util.aff_b(cb.JChecksetP.isSelected());
vProp.addElement(setP);//true if set pressure [22]
String setT="setT="+Util.aff_b(cb.JChecksetT.isSelected());
vProp.addElement(setT);//true if set temperature [23]
vProp.addElement(new Double(cb.getUsefulEnergy()));//useful energy [24]
}
}
}
```

```
else if(type.equals("node")){
Node the_node=getNode(nomType);
if(the_node!=null){
vProp.addElement(the_node.mainProcess.getName());//main process name
vProp.addElement(the_node.getClassType());//node type
vProp.addElement(the_node.getEffNode());//node efficiency
vProp.addElement(Util.aff_b(the_node.isoBaric));//is node isobaric?
vProp.addElement(new Integer(the_node.vBranch.size()));//number of branches
for(int j=0;j<the_node.vBranch.size();j++){
Object[] branch=(Object[])the_node.vBranch.elementAt(j);
Transfo the_process=(Transfo)branch[0];
vProp.addElement(the_process.getName());//branch process name
}
}
}
else if(type.equals("heatEx")){
HeatExDemo hX=getHX(nomType);
if(hX!=null){
String coldFluid="",hotFluid="";
if(hX instanceof ThermoCoupler){
ThermoCoupler tc=(ThermoCoupler)hX;
coldFluid=tc.getHcName();
hotFluid=tc.getExchange().getName();
}
else {
coldFluid=hX.getColdFluid().getName();
hotFluid=hX.getHotFluid().getName();
}
vProp.addElement(hotFluid);//hot fluid process name
vProp.addElement(coldFluid);//cold fluid process name
vProp.addElement(hX.getHxType());//heat exchanger type (counterflow, parallel flow...)
vProp.addElement(Util.aff_b(hX.JCheckTceImp.isSelected()));//set Tce
vProp.addElement(Util.aff_b(hX.JCheckTcsImp.isSelected()));//set Tcs
vProp.addElement(Util.aff_b(hX.JCheckMcImp.isSelected()));//set mc
vProp.addElement(Util.aff_b(hX.JCheckTfeImp.isSelected()));//set Tfe
vProp.addElement(Util.aff_b(hX.JCheckTfsImp.isSelected()));//set Tfs
vProp.addElement(Util.aff_b(hX.JCheckMfImp.isSelected()));//set mf
vProp.addElement(Util.aff_b(hX.JCheckMinPinch.isSelected()));//set minimum pinch
vProp.addElement(Util.aff_b(hX.JCheckSetEff.isSelected()));//set efficiency
vProp.addElement(Util.aff_b(hX.IcalcDirect));//set direct calculation (not used I think)
```

vProp.addElement(Util.aff_b(hX.JCheckDesign.isSelected()));//set design mode

vProp.addElement(new Double(hX.R));//R value

vProp.addElement(new Double(hX.NUT));//NUT value

vProp.addElement(new Double(hX.UA));//UA value

vProp.addElement(new Double(hX.DTML));//DTML value

vProp.addElement(new Double(hX.epsi_value.getValue()));//efficiency value

vProp.addElement(new Double(hX.DTmin_value.getValue()));//pinch value

}

}

return vProp;

}

Method updatePoint() of Projet

/**

- met à jour un point, avec recalcul éventuel
- valable pour calculs humides
- <p>
- updates a point, possibly with recalculation
- valid for moist gas calculations
- /

public void updatePoint(Vector properties){//profondément modifié en 2018 pour version 2.8

String nomPoint=(String)properties.elementAt(0);

// System.out.println("properties.size() "+properties.size());//properties.size()

PointCorps point=getPoint(nomPoint);

if(point!=null){

String test=(String)properties.elementAt(1);

boolean updateT=Util.lit_b(test);

String value=(String)properties.elementAt(2);

double T=Util.lit_d(value);

test=(String)properties.elementAt(3);

boolean updateP=Util.lit_b(test);

value=(String)properties.elementAt(4);

double P=Util.lit_d(value);

test=(String)properties.elementAt(5);

boolean updateX=Util.lit_b(test);

value=(String)properties.elementAt(6);

double x=Util.lit_d(value);

//calculs a effectuer dans le cas general

if(updateT)point.setT(T);

if(updateP){

if(point.open_syst)point.setP(P);

```
else {
point.setV(P);
point.calcPoint();
}
}
if(updateX)point.setX(x);
point.CalculeUnPoint();
//pour melanges humides
if(properties.size()>7){//modRG 2018
test=(String)properties.elementAt(7);
boolean melHum=Util.lit_b(test);
if(melHum){
if(point.lecorps.typeCorps==4){//modRG 01/06
String task=(String)properties.elementAt(8);
// System.out.println("task "+task);
// Attention : le fluide chaud ne se refroidit
value=(String)properties.elementAt(9);
if(task.equals("setW and calculate all")){//sets w and calculates moist properties
double w=Util.lit_d(value);
point.setW(w);
point.calcHum();
}
if(task.equals("setW and calculate q'")){//sets w and calculates moist properties except t'
double w=Util.lit_d(value);
point.setW(w);
point.calcQprime(false);//modRG 08/06
// System.out.println("epsi "+point.epsi_value.getValue()+" q' "+point.qprime_value.getValue());
}
if(task.equals("setEpsi")){//sets epsilon
double epsi=Util.lit_d(value);
point.setEpsi(epsi);
point.impHumRel();
}
if(task.equals("setEpsi and calculate")){//sets epsilon and calculates moist properties
double epsi=Util.lit_d(value);
point.setEpsi(epsi);
point.impHumRel();
point.calcQprime(false);//modRG 08/06
}
if(task.equals("calcWsat")){//calculates saturation properties and moist properties except t'
T=Util.lit_d(value);
double wsat=point.wsat(T);
point.setW(wsat);
point.calcQprime(false);//modRG 08/06
}
if(task.equals("modHum")){//modifies the gas composition, setting the gas humidity to WPoint
point.modGasHum(false);
}
if(task.equals("setGasHum")){//sets the gas humidity to w//modRG 01/06
double w=Util.lit_d(value);
point.setGasHum(w);
}
}
else{//modRG 01/06 (modifier la ressource)
String mess="Watch out! point "+point.getName()+"'s substance is not a compound gas.\nYou cannot use it for wet gas
calculations.";
JOptionPane.showMessageDialog(this,mess);
}
}
}
if((properties.size()>10)&&(properties.size()<=12)){//mise a jour du facteur de correction d'une pression imposee
test=(String)properties.elementAt(10);
boolean updateCorrFact=Util.lit_b(test);
if(updateCorrFact){//
value=(String)properties.elementAt(11);
double corr=Util.lit_d(value);
point.pressCorrFact_value.setValue(corr);
}
}
if(properties.size()>12){//mise a jour du sous-refroidissement
test=(String)properties.elementAt(12);
boolean updateCorrFact=Util.lit_b(test);
// System.out.println("updateCorrFact.P "+test);//properties.size()
if(updateCorrFact){//
value=(String)properties.elementAt(13);
double corr=Util.lit_d(value);
point.dTsat_value.setValue(corr);
// System.out.println("dTsat_value.P "+corr);
}
}
}
```

}

Method getSubstProperties() of Corps

/**

- donne les fonctions d'état du substance
- <p>
- gives the substance state functions
- @return Vector
- <p>
- do not override
- /

```
public Vector getSubstProperties(){
Vector vProp=new Vector();
vProp.addElement(new Double(T));//Temperature
vProp.addElement(new Double(P));//Pressure
vProp.addElement(new Double(xx));//Quality
vProp.addElement(new Double(V));//Volume
vProp.addElement(new Double(U));//Internal energy
vProp.addElement(new Double(H));//Enthalpy
vProp.addElement(new Double(S));//Entropy
vProp.addElement(new Double(M));//Molar mass
vProp.addElement(new Integer(typeCorps));//Substance type
vProp.addElement(new Double(M_sec));//Dry gas molar mass [9]
vProp.addElement(new Double(TC));//Critical temperature [10]
vProp.addElement(new Double(PC));//Critical pressure [11]
vProp.addElement(new Double(VC));//Critical volume [12]
vProp.addElement(new Double(getChemicalExergy()));//Chemical exergy [13]
return vProp;
}
```

Method getExternalClassInstances() of Projet

/**

- Provides the external class instances
-
- @return a Vector containing the instances and their type
- /

```
public Vector getExternalClassInstances(){
Vector vInstances=new Vector();
Primtype pt;
int nbProcess=vProcess.size();
for(int i=0;i<nbProcess;i++){
```

```
pt=(Primtype)vProcess.elementAt(i);
if(pt instanceof TransfoExterne){
Object[] obj=new Object[6];
TransfoExterne te=(TransfoExterne)pt;
obj[0]="process";
obj[1]=te.cType.externalInstance;//instance de la classe externe
obj[2]=te.getName();
obj[3]=te.cType.externalInstance.getType();//type de la classe externe
obj[4]=te.getPointAmont().getName();
obj[5]=te.getPointAval().getName();
vInstances.addElement(obj);
}
}
int nbNode=vNode.size();
for(int i=0;i<nbNode;i++){
pt=(Primtype)vNode.elementAt(i);
if((pt instanceof MixerExterne)||(pt instanceof DividerExterne)){
Node nd=(Node)pt;
Object[] obj=new Object[6];
obj[0]="node";
obj[1]=nd.cType.externalInstance;//instance de la classe externe
obj[2]=nd.getName();
obj[3]=nd.cType.externalInstance.getType();//type de la classe externe
obj[4]=nd.getMainProcess().getName();
obj[5]=new Integer(nd.getBranches().size());
vInstances.addElement(obj);
}
}
return vInstances;
}
```

Method updateProcess() of Projet

/**

- met à jour une transfo
- <p>
- updates a process
- /

```
public void updateProcess(Vector properties){
String nomTransfo=(String)properties.elementAt(0);//getName()
Transfo trsf=getTransfo(nomTransfo);
```

```
if(trsf!=null){
String typeTransfo=(String)properties.elementAt(1);//getClassType()
String test=(String)properties.elementAt(2);
boolean updateFlow=Util.lit_b(test);
Double y=(Double)properties.elementAt(3);//getFlow()
double flowRate=y.doubleValue();
if(updateFlow)trsf.setFlow(flowRate);
test=(String)properties.elementAt(4);
boolean reCalculate=Util.lit_b(test);
test=(String)properties.elementAt(5);
trsf.setSetFlow(Util.lit_b(test));
if(typeTransfo.equals("Compression")){
Compression ce=(Compression)trsf;
test=(String)properties.elementAt(6);
boolean updateEta=Util.lit_b(test);
y=(Double)properties.elementAt(7);//getIsentropicEfficiency()
if(updateEta)ce.setIsentropicEfficiency(y.doubleValue());
if(properties.size()>8){
test=(String)properties.elementAt(8);
boolean updatePressRatio=Util.lit_b(test);
y=(Double)properties.elementAt(9);//getIsentropicEfficiency()
if(updatePressRatio)ce.setPressureRatio(y.doubleValue());
}
}
if(typeTransfo.equals("Expansion")){
Expansion ce=(Expansion)trsf;
test=(String)properties.elementAt(6);
boolean updateEta=Util.lit_b(test);
y=(Double)properties.elementAt(7);//getIsentropicEfficiency()
if(updateEta)ce.setIsentropicEfficiency(y.doubleValue());
if(properties.size()>8){
test=(String)properties.elementAt(8);
boolean updatePressRatio=Util.lit_b(test);
y=(Double)properties.elementAt(9);//getIsentropicEfficiency()
if(updatePressRatio)ce.setPressureRatio(y.doubleValue());
}
}
if(typeTransfo.equals("Combustion")){
Combustion ce=(Combustion)trsf;
test=(String)properties.elementAt(6);
boolean updateTout=Util.lit_b(test);
y=(Double)properties.elementAt(7);//getOutletTemperature()
if(updateTout)ce.setOutletTemperature(y.doubleValue());
if(properties.size()>8){
test=(String)properties.elementAt(8);
boolean updateLambda=Util.lit_b(test);
y=(Double)properties.elementAt(9);//
if(updateLambda)ce.setLambda(y.doubleValue());
}
if(properties.size()>10){
test=(String)properties.elementAt(10);
boolean updateEta=Util.lit_b(test);
y=(Double)properties.elementAt(11);//
if(updateEta)ce.setThermalEfficiency(y.doubleValue());
}
}
if(reCalculate)trsf.Calculate();
}
}
```

Method updateNode() of Projet

/**

- met à jour un noeud
- <p>
- updates a node
- /

```
public void updateNode(Vector properties){//modRG 2018
String name=(String)properties.elementAt(0);//getName()
Node nd=getNode(name);
String test=(String)properties.elementAt(1);
boolean reCalculate=Util.lit_b(test);
if(nd instanceof Separator){//on passe la valeur true or false en 3ème argument, et la valeur de l'efficacité de séchage en 4ème
Separator sp=(Separator)nd;
test=(String)properties.elementAt(2);
boolean updateEpsi=Util.lit_b(test);
Double y=(Double)properties.elementAt(3);
double epsi=y.doubleValue();
if(updateEpsi)sp.setEfficiency(epsi);
}
if(nd instanceof Divider){//modifié en 2018 //modRG 2018
```

//on passe la valeur true or false pour updateFactor en 3ème argument,

//le nom de la transfo dont on souhaite changer le facteur de débit en 4ème

//et la valeur du facteur de débit en 5ème

Divider div=(Divider)nd;

if(properties.size()==5){

test=(String)properties.elementAt(2);

boolean updateFactor=Util.lit_b(test);

if(updateFactor){

reCalculate=false;//le diviseur ne doit alors être recalculé que quand tous les facteurs de débit ont été mis à jour

String branchName=(String)properties.elementAt(3);//le recalcul est effectué par un appel sans mise à jour des facteurs de débit

String flowFactor=(String)properties.elementAt(4);

div.setupFlowFactor(branchName, flowFactor);

}

}

}

if(reCalculate)nd.Calculate();

}

Method updateHX() of Projet

/**

- met à jour un échangeur de chaleur
- <p>
- updates a heat exchanger
- /

public void updateHX(Vector properties){

String name=(String)properties.elementAt(0);//getName()

HeatExDemo hx=getHX(name);

String test=(String)properties.elementAt(1);

boolean reCalculate=Util.lit_b(test);

test=(String)properties.elementAt(2);

boolean updateUA=Util.lit_b(test);

Double y=(Double)properties.elementAt(3);

double UA=y.doubleValue();

if(updateUA)hx.setUA(UA);

test=(String)properties.elementAt(4);

boolean updateEpsi=Util.lit_b(test);

y=(Double)properties.elementAt(5);

if(updateEpsi)hx.setEpsi(y.doubleValue());

test=(String)properties.elementAt(6);

boolean updateDTmin=Util.lit_b(test);

y=(Double)properties.elementAt(7);

if(updateDTmin)hx.setDTmin(y.doubleValue());

test=(String)properties.elementAt(8);

boolean updateCalcMode=Util.lit_b(test);

test=(String)properties.elementAt(9);

boolean calcMode=Util.lit_b(test);

if(updateCalcMode)hx.setCalcMode(calcMode);

if(reCalculate)hx.Calculate();

}

Method updateSetPressure() of Projet

/**

- met à jour une pression imposée et recalcule la pression des points
- les facteurs de correction de ces derniers sont modifiables par updatePoint()
- <p>
- updates a set pressure and recalculates the point pressures
- their correction factors can be updated by updatePoint()
- /

public void updateSetPressure(Vector properties){//modRG 2008

String nomPres=(String)properties.elementAt(0);//getName()

PresSetValue thePres=getPresSet(nomPres);

if(thePres!=null){//le false signifie que la pression est changée, mais sans notification, pour éviter un recalcul complet

Double y=(Double)properties.elementAt(1);

double value=y.doubleValue();

thePres.setValue(value);

thePres.modifyPressures(false);

}

}

Method getSubstProperties() of Corps

/**

- donne les fonctions d'état du corps
- <p>
- gives the substance state functions
- @return Vector
- <p>
- do not override
- /

public Vector getSubstProperties(){

Vector vProp=new Vector();

```
vProp.addElement(new Double(T));//Temperature [0]
vProp.addElement(new Double(P));//Pressure [1]
vProp.addElement(new Double(xx));//Quality [2]
vProp.addElement(new Double(V));//Volume [3]
vProp.addElement(new Double(U));//Internal energy [4]
vProp.addElement(new Double(H));//Enthalpy [5]
vProp.addElement(new Double(S));//Entropy [6]
vProp.addElement(new Double(M));//Molar mass [7]
vProp.addElement(new Integer(typeCorps));//Substance type [8]
vProp.addElement(new Double(M_sec));//Dry gas molar mass [9]
vProp.addElement(new Double(TC));//Critical temperature [10]
vProp.addElement(new Double(PC));//Critical pressure [11]
vProp.addElement(new Double(VC));//Critical volume [12]
vProp.addElement(new Double(getChemicalExergy()));//Chemical exergy [13]
return vProp;
}
```

Method getExternalClassInstances() of Projet

/**

- Provides the external class instances
-
- @return a Vector containing the instances and their type
- /

```
public Vector getExternalClassInstances(){
Vector vInstances=new Vector();
Primtype pt;
int nbProcess=vProcess.size();
for(int i=0;i<nbProcess;i++){
pt=(Primtype)vProcess.elementAt(i);
if(pt instanceof TransfoExterne){
Object[] obj=new Object[6];
TransfoExterne te=(TransfoExterne)pt;
obj[0]="process";
obj[1]=te.cType.externalInstance;//instance de la classe externe
obj[2]=te.getName();
obj[3]=te.cType.externalInstance.getType();//type de la classe externe
obj[4]=te.getPointAmont().getName();
obj[5]=te.getPointAval().getName();
vInstances.addElement(obj);
}
}
int nbNode=vNode.size();
for(int i=0;i<nbNode;i++){
pt=(Primtype)vNode.elementAt(i);
if((pt instanceof MixerExterne)||(pt instanceof DividerExterne)){
Node nd=(Node)pt;
Object[] obj=new Object[6];
obj[0]="node";
obj[1]=nd.cType.externalInstance;//instance de la classe externe
obj[2]=nd.getName();
obj[3]=nd.cType.externalInstance.getType();//type de la classe externe
obj[4]=nd.getMainProcess().getName();
obj[5]=new Integer(nd.getBranches().size());
vInstances.addElement(obj);
}
}
return vInstances;
}
```

Method setupChart() of Projet

```
public void setupChart(Vector properties){
String task=(String)properties.elementAt(0);
String value=(String)properties.elementAt(1);
if(task.equals("openDiag")){
int i=Util.lit_i(value);
setupChart(false);
cm.setupChart(i);
}
else if(task.equals("selectSubstance")&& cm!=null){
Graph graph=cm.getChart();
graph.getDiagIni().setSelectedSubstance(value);
graph.select_vap();
}
else if(task.equals("readCycle")&& cm!=null){
Graph graph=cm.getChart();
graph.litCycle(value,"");
graph.setConnectedCycle();
}
else if(task.equals("unSelectCycle")&& cm!=null){
Graph graph=cm.getChart();
graph.setupCycleManager();
CycleManager cM=graph.getCycleManager();
cM.unSelect(value);
graph.repaint();
}
else if(task.equals("removeCycle")&& cm!=null){
Graph graph=cm.getChart();
graph.setupCycleManager();
CycleManager cM=graph.getCycleManager();
cM.removeCycle(value);
graph.repaint();
}
else if(task.equals("setChartType")&& cm!=null){
Graph graph=cm.getChart();
graph.setChartType(value);
}
}
```

Annex 3 : utility methods of class Util

Class ExtThopt.Util provides a number of utility methods to facilitate the programming of external classes:

Display methods

- public static String aff_i(int i) : affiche un entier
- public static String aff_b(boolean b) : affiche un booléen
- public static String aff_d(double d) : affiche un double
- public static String aff_d(double d, int dec) : affiche un double avec dec décimales

Conversion methods

- public static int lit_i(String s) : lit un entier
- public static boolean lit_b(String s) : lit un booléen
- public static double lit_d(String s) : lit un double

Methods extracting values from a String

- public static String extr_value(String s)
- public static String extr_value(String ligne_data, String search)

The first method extracts "3.4" from "value = 3.4", whatever the text to the left of the equal sign. It can be used with a StringTokeniser separating couples "value = xxx".

The second combines with StringTokeniser separator tab and the previous method, searching for the couple "search = xxx".

Both methods return null in case of failure.

Function inversion methods

- public static double dicho_T (Inversable inv, double value,double param, String function, double valMin, double valMax,double epsilon)

This generic method allows the inversion of a function by dichotomy. The class must implement the Inversable interface, which requires to define method f_dicho:

public double f_dicho(double x, double param, String function)

value is the objective value for the method returned by f_dicho, the variable x varies between valMin and valMax, epsilon is the accuracy criterion, and function is a String allowing to identify a particular method in f_dicho:

if (function.equals("P_LiBr"))return getP(x,T);

After 50 iterations, dicho_T returns 0 if no convergence.

Methods for handling a pure gas in a gas composition Vector

- public static double molarComp(Vector vComp,String pureGas) : gives the molar fraction of pureGas
- public static boolean contains(Vector vComp,String pureGas) : gives true if pureGas exists
- public static void updateMolarComp(Vector vComp,String pureGas, double newFractMol) : updates the molar fraction of pureGas

Annex 4 : TEP ThermoSoft - Java / Delphi Interface – Application to Thermoptim

(par F. Rivollet)

Ce document explique le principe de passerelle entre TEP ThermoSoft et Thermoptim. Ces deux programmes sont écrits respectivement en Pascal sous environnement Delphi et en Java. L'objectif est le calcul des propriétés thermodynamiques nécessaires sous Thermoptim à partir des modèles développés pour TEP ThermoSoft.

Structure de dialogue entre les deux programmes

Le schéma ci-dessous reprend le principe de dialogue entre les deux programmes et les fichiers nécessaire à un calcul.

Fig. 1 : Relations entre les fichiers

La principale difficulté est la compatibilité des fonctions appelées du Java vers le Pascal. Pour ce faire, une librairie spécifique (*TEPThermoSoftJava.dll*) permet d'assurer le passage des variables entre les deux entités.

La définition et l'exécution des calculs au sein de TEP ThermoSoft nécessitent :

- La routine principale (*TEPThermoSoftCalc.dll*)
- un fichier système (*fichier .mel*) qui rassemble l'ensemble des propriétés de calcul.
- les modèles thermodynamiques écrits sous forme de librairies (*.dll*) et contenus dans un répertoire spécifique.

Remarque : Les deux fichiers TEPThermoSoftCalc.dll et TEPThermoSoftJava.dll doivent se situer dans le même répertoire.

Exécuter un calcul en java

Définition des méthodes de dialogue avec TEP ThermoSoft

Les méthodes suivantes doivent être définies en Java comme faisant référence à la librairie externe « Delphi2Java.dll ».
Leur fonction et leur syntaxe sont expliquées dans les paragraphes 2.2 à 2.5 et un exemple est donné en 2.6.

public native int InitSession(String RepMODELES);

public native void FermerSession();

public native int ChargerSysteme (String FichierMEL);

public native double Lire(String Symbole);

public native void Ecrire(String Symbole, double valeur);

public native void Calculer(String code);

Chargement/Libération des modèles thermodynamiques en mémoire

Avant tout calcul, la méthode « InitSession » doit être appelée afin d'initialiser l'ensemble des paramètres et de spécifier le répertoire contenant les modèles thermodynamiques « .dll ».

InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);

Cette fonction retourne un entier qui détermine un code d’état :

1. Pas d’erreur

> -1 Erreur inconnue

Une fois que les fonctions de calcul ne sont plus nécessaires, il est souhaitable de libérer la mémoire en appelant « FermerSession »

FermerSession() ;

Définition d’un système

Un « système » représente soit un substance pur, soit un mélange de plusieurs composés. Toutes les informations relatives au calcul de ses propriétés (Modèles thermodynamiques, valeurs des paramètres de calcul, …), sont regroupées dans un fichier unique dont l’extension est « .mel ». Ce fichier peut être ouvert avec un simple éditeur de texte. Sa structure reprend celle d’un fichier « .ini » sous Windows. Cela signifie que des catégories sont définies entre crochets :

[GENERAL]

NOM = Melange

DATE = 25/04/2005

HEURE = 08:17:56

[CPS]

994 AMMONIA

1000 WATER

[ELV]

S CAlpha AlphaMC.dll

S CCPGP CPGP107.dll

// … //

P a(1)

P a(2)

P ALPHA(1)

// .. //

P CP107(2)[0] 33363

P CP107(2)[1] 26790

P P Pa

P Pc(1) 11280005.625 Pa

P Pc(2) 22055007.45 Pa

P PHI|l(1)

// .. //

Au sein de TEP ThermoSoft, tout calcul se réfère à un système donné. Pour cela, il faut, à partir de l’application Java, charger les paramètres de calcul en mémoire à l’aide de la fonction « *ChargerSysteme* » en spécifiant le fichier système voulu :

ChargerCalcul(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File.separator+sele

Cette méthode retourne un entier qui spécifie le numéro du mélange chargé en mémoire.

De plus, ce numéro, s’il est négatif, indique une erreur lors du chargement du fichier. Pour le moment les codes d’erreur sont les suivants :

- 1 Erreur inconnue
- 2 Fichier introuvable

- 3 Aucun composé défini dans le fichier.

Modifier / Lire des variables de TEP ThermoSoft

Une fois un fichier de mélange chargé en mémoire, deux méthodes ont été écrites pour permettre de modifier et de lire les valeurs des variables se référant au système. Toute variable intervenant dans les calculs peut être modifiée ou lue. Pour cela il suffit de respecter la typographie suivante :

SYMB(cps)[vecteur]|phase

SYMB symbole de la variable à modifier (*Ex. T : Température – Cf.*)

cps composé ou interaction entre composés (Ex. *(1) , (2), (1_2)*). La numérotation des composés commence à 1.

vecteur numéro du vecteur (ex. *[0], [1], …*). La numérotation des vecteurs commence à 0.

phase lors de données multiphasiques, la barre | suivi d'une lettre (l ou v) permet de définir la phase recherchée (ex. |l ou |v).

Par exemple, la lecture de la température peut se faire par la méthode :

double temperature = Lire("T");

De même la modification du premier paramètre de la fonction Alpha de Mathias-Copeman pour le composé 2 s'écrit :

Ecrire("MC(2)[0]", 0.152);

Lancer un calcul

Pour lancer un calcul, il suffit d'écrire la valeur des paramètres de calcul au moyen de la méthode « Ecrire » vue précédemment, puis de lancer le calcul par la méthode « Calculer » :

Calculer("Tx");

La méthode « Calculer » attend un paramètre qui définit le type de calcul souhaité (cf.) Dans l'exemple ci-dessus, il s'agit d'un calcul d'équilibre LV en mélange en spécifiant la température et la composition liquide. Ainsi, avant d'exécuter cette fonction il est nécessaire de bien définir les variables T et x comme par exemple pour un binaire :

Ecrire("T",280);

Ecrire("x(1)",0.1);

Ecrire("x(2)",0.9);

Exemple d'écriture d'un calcul complet

Voici un exemple de lignes de commande permettant un calcul à l'équilibre LV du système « NH3-H2O» défini dans le fichier système « NH3-H2O.mel » :

InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);

//

ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File.separator+" H2O.mel");

//

Ecrire("T",280);

Ecrire("x(1)",0.1);

Ecrire("x(2)",0.9);

//

Calculer("Tx");

//

P = Lire("P");

y[0] = Lire("y(1)");

y[1] = Lire("y(2)");

//

FermerSession() ;

Variables et méthodes de calcul disponibles

Variables classiques

Voici une liste des symboles les plus utilisés dans les méthodes « Lire » et « Ecrire ».

Symbole	Unité	Description
T	K	Température
P	Pa	Pression
x(i)		Composition liquide du composé « i »
y(i)		Composition gaz du composé « i »
z(i)		Composition globale du composé « i »
h, h|l, h|v	J/mol	Enthalpie
s, s|l, s|v	J/mol/K	
TauVap		Taux de vaporisation (= -1 lors d'un calcul ELV Px, Tx, Ty ou Py).
h0	J/mol	Enthalpie de référence.
s0	J/mol	Entropie de référence
Mw(i)	kg/mol	Masse molaire du composé « i »

Méthodes de calcul

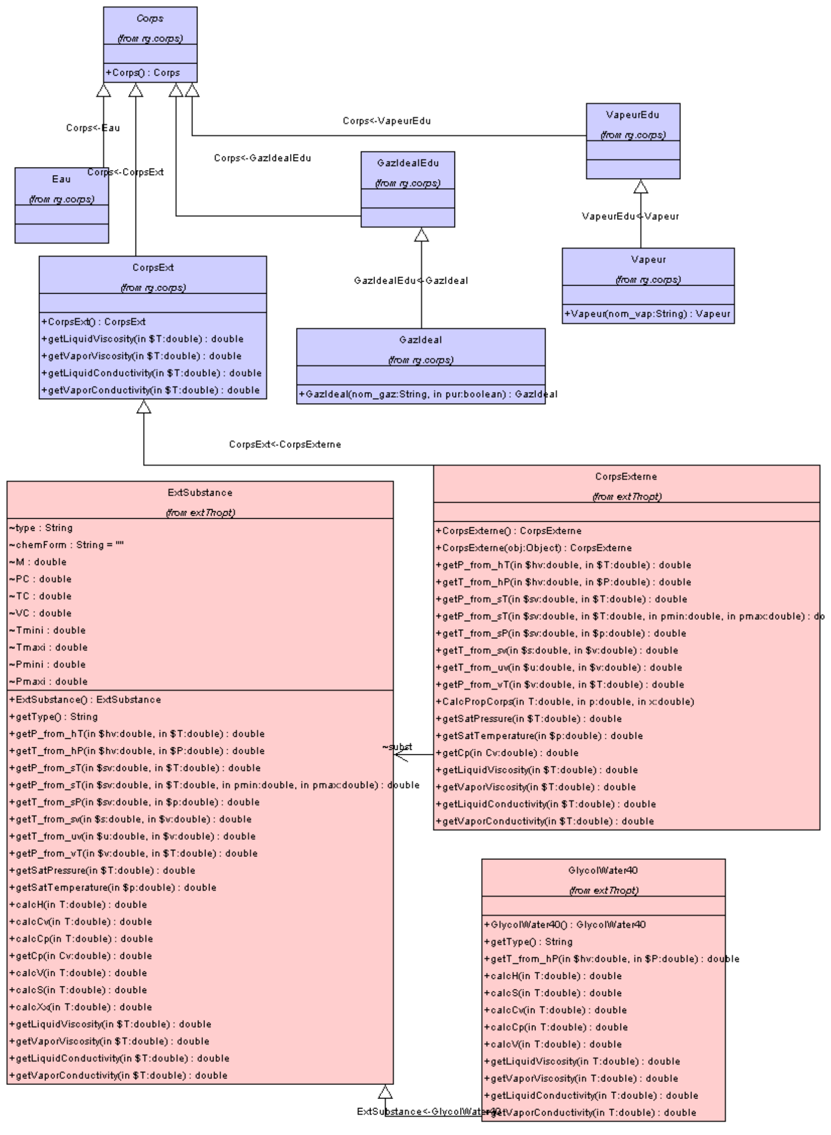
Voici une liste des symboles des méthodes disponibles (elle sera complétée au fur et à mesure des besoins). *MEL concerne les mélanges et CP, les corps purs*.

Symbole	Type de calcul	Description
Tx	MEL	Calcul à l'équilibre LV en spécifiant la température et la composition liquide.
Ty	MEL	Calcul à l'équilibre LV en spécifiant la température et la composition gaz.
Px	MEL	Calcul à l'équilibre LV en spécifiant la pression et la composition liquide.
Py	MEL	Calcul à l'équilibre LV en spécifiant la pression et la composition gaz.
PTz	MEL	Calcul en spécifiant la composition globale, la pression et la température (données à l'équilibre ou hors équilibre LV).

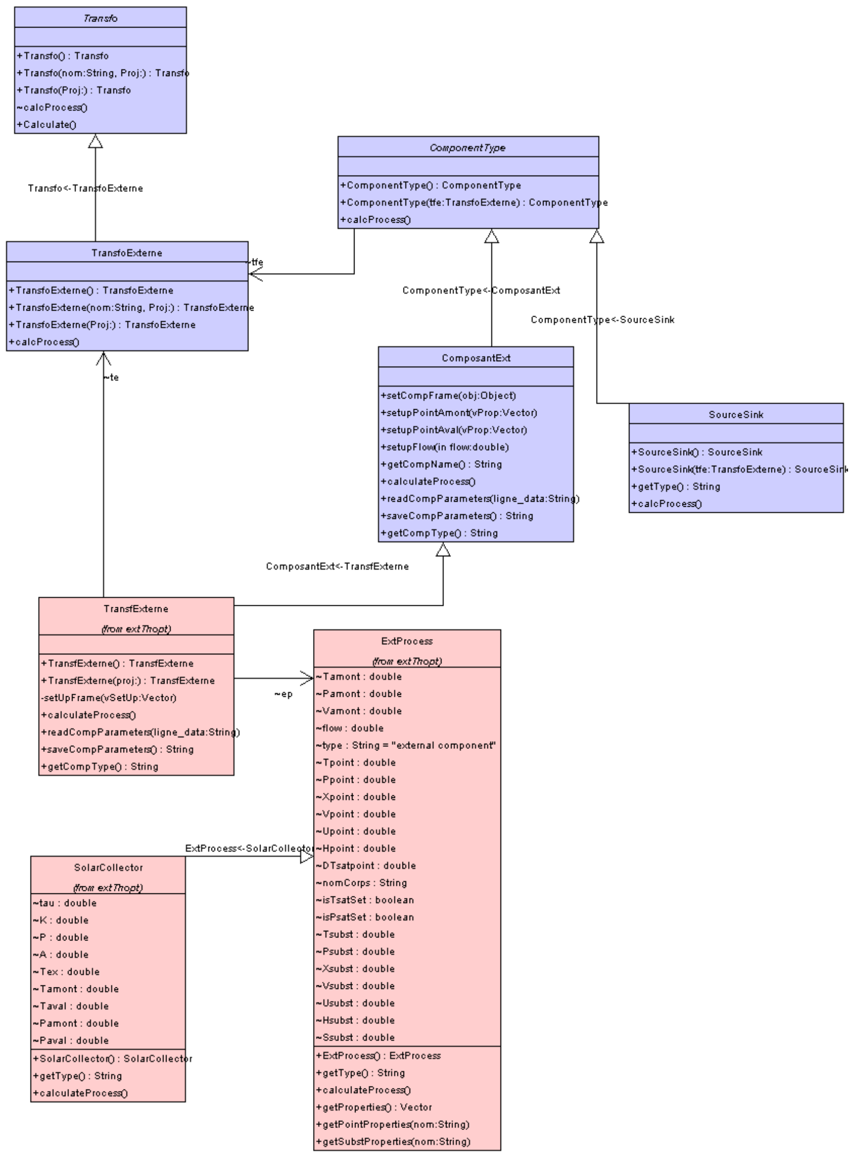
Pour le moment la méthode PTz semble pouvoir suffire car valable en substance purs et en mélange (suivant les valeurs de z). De plus elle est continue à et hors ELV. Ainsi une méthode numérique simple devrait permettre d'estimer les valeurs à h et s constant.

Annex 5 : UML Diagrams of external classes

Substance class diagram


Corps
(from rg.corps)
+Corps() : Corps
Corps<-Eau
Corps<-CorpsExt
Corps<-GazIdealEdu
Corps<-VapeurEdu
VapeurEdu
(from rg.corps)
GazIdealEdu
(from rg.corps)
Eau
(from rg.corps)
VapeurEdu<-Vapeur
Vapeur
(from rg.corps)
+Vapeur(nom_vap:String) : Vapeur
CorpsExt
(from rg.corps)
+CorpsExt() : CorpsExt
+getLiquidViscosity(in $T:double) : double
+getVaporViscosity(in $T:double) : double
+getLiquidConductivity(in $T:double) : double
+getVaporConductivity(in $T:double) : double
GazIdealEdu<-GazIdeal
GazIdeal
(from rg.corps)
+GazIdeal(nom_gaz:String, in pur:boolean) : GazIdeal
CorpsExt<-CorpsExterne
ExtSubstance
(from extThopt)
~type : String
~chemForm : String = ""
~M : double
~PC : double
~TC : double
~VC : double
~Tmini : double
~Tmaxi : double
~Pmini : double
~Pmaxi : double
+ExtSubstance() : ExtSubstance
+getType() : String
+getP_from_hT(in $hv:double, in $T:double) : double
+getT_from_hP(in $hv:double, in $P:double) : double
+getP_from_sT(in $sv:double, in $T:double) : double
+getP_from_sT(in $sv:double, in $T:double, in pmin:double, in pmax:double) : double
+getT_from_sP(in $sv:double, in $p:double) : double
+getT_from_sv(in $s:double, in $v:double) : double
+getT_from_uv(in $u:double, in $v:double) : double
+getP_from_vT(in $v:double, in $T:double) : double
+getSatPressure(in $T:double) : double
+getSatTemperature(in $p:double) : double
+calcH(in T:double) : double
+calcCv(in T:double) : double
+calcCp(in T:double) : double
+getCp(in Cv:double) : double
+calcV(in T:double) : double
+calcS(in T:double) : double
+calcXx(in T:double) : double
+getLiquidViscosity(in $T:double) : double
+getVaporViscosity(in $T:double) : double
+getLiquidConductivity(in $T:double) : double
+getVaporConductivity(in $T:double) : double
~subst
CorpsExterne
(from extThopt)
+CorpsExterne() : CorpsExterne
+CorpsExterne(obj:Object) : CorpsExterne
+getP_from_hT(in $hv:double, in $T:double) : double
+getT_from_hP(in $hv:double, in $P:double) : double
+getP_from_sT(in $sv:double, in $T:double) : double
+getP_from_sT(in $sv:double, in $T:double, in pmin:double, in pmax:double) : do
+getT_from_sP(in $sv:double, in $p:double) : double
+getT_from_sv(in $s:double, in $v:double) : double
+getT_from_uv(in $u:double, in $v:double) : double
+getP_from_vT(in $v:double, in $T:double) : double
+CalcPropCorps(in T:double, in p:double, in x:double)
+getSatPressure(in $T:double) : double
+getSatTemperature(in $p:double) : double
+getCp(in Cv:double) : double
+getLiquidViscosity(in $T:double) : double
+getVaporViscosity(in $T:double) : double
+getLiquidConductivity(in $T:double) : double
+getVaporConductivity(in $T:double) : double
GlycolWater40
(from extThopt)
+GlycolWater40() : GlycolWater40
+getType() : String
+getT_from_hP(in $hv:double, in $P:double) : double
+calcH(in T:double) : double
+calcS(in T:double) : double
+calcCv(in T:double) : double
+calcCp(in T:double) : double
+calcV(in T:double) : double
+getLiquidViscosity(in T:double) : double
+getVaporViscosity(in T:double) : double
+getLiquidConductivity(in T:double) : double
+getVaporConductivity(in T:double) : double
ExtSubstance<-GlycolWater40

External component class diagram


Transfo
+Transfo() : Transfo
+Transfo(nom:String, Proj:) : Transfo
+Transfo(Proj:) : Transfo
~calcProcess()
+Calculate()
Transfo<-TransfoExterne
ComponentType
+ComponentType() : ComponentType
+ComponentType(tfe:TransfoExterne) : ComponentType
+calcProcess()
TransfoExterne
~tfe
+TransfoExterne() : TransfoExterne
+TransfoExterne(nom:String, Proj:) : TransfoExterne
+TransfoExterne(Proj:) : TransfoExterne
+calcProcess()
ComponentType<-ComposantExt
ComponentType<-SourceSink
ComposantExt
+setCompFrame(obj:Object)
+setupPointAmont(vProp:Vector)
+setupPointAval(vProp:Vector)
+setupFlow(in flow:double)
+getCompName() : String
+calculateProcess()
+readCompParameters(ligne_data:String)
+saveCompParameters() : String
+getCompType() : String
~te
SourceSink
+SourceSink() : SourceSink
+SourceSink(tfe:TransfoExterne) : SourceSink
+getType() : String
+calcProcess()
ComposantExt<-TransfExterne
TransfExterne
(from extThopt)
+TransfExterne() : TransfExterne
+TransfExterne(proj:) : TransfExterne
-setUpFrame(vSetUp:Vector)
+calculateProcess()
+readCompParameters(ligne_data:String)
+saveCompParameters() : String
+getCompType() : String
~ep
ExtProcess
(from extThopt)
~Tamont : double
~Pamont : double
~Vamont : double
~flow : double
~type : String = "external component"
~Tpoint : double
~Ppoint : double
~Xpoint : double
~Vpoint : double
~Upoint : double
~Hpoint : double
~DTsatpoint : double
~nomCorps : String
~isTsatSet : boolean
~isPsatSet : boolean
~Tsubst : double
~Psubst : double
~Xsubst : double
~Vsubst : double
~Usubst : double
~Hsubst : double
~Ssubst : double
+ExtProcess() : ExtProcess
+getType() : String
+calculateProcess()
+getProperties() : Vector
+getPointProperties(nom:String)
+getSubstProperties(nom:String)
ExtProcess<-SolarCollector
SolarCollector
(from extThopt)
~tau : double
~K : double
~P : double
~A : double
~Tex : double
~Tamont : double
~Taval : double
~Pamont : double
~Paval : double
+SolarCollector() : SolarCollector
+getType() : String
+calculateProcess()