Steam desuperheater

A steam desuperheater achieves partial or total desuperheating of superheated steam. This operation, which generally leads to detrimental irreversibilities, is performed only in specific circumstances, as at mechanical vapor compression outlet, or to regulate the production of superheated steam for cogeneration.

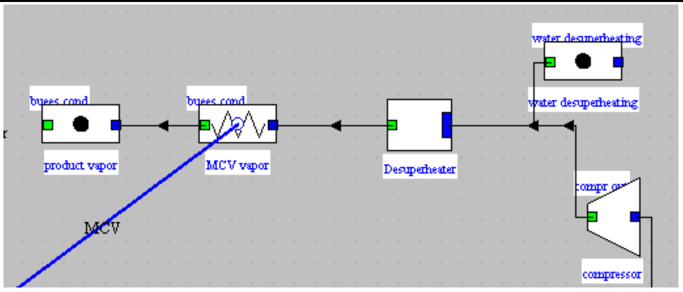


Figure 1: Desuperheater

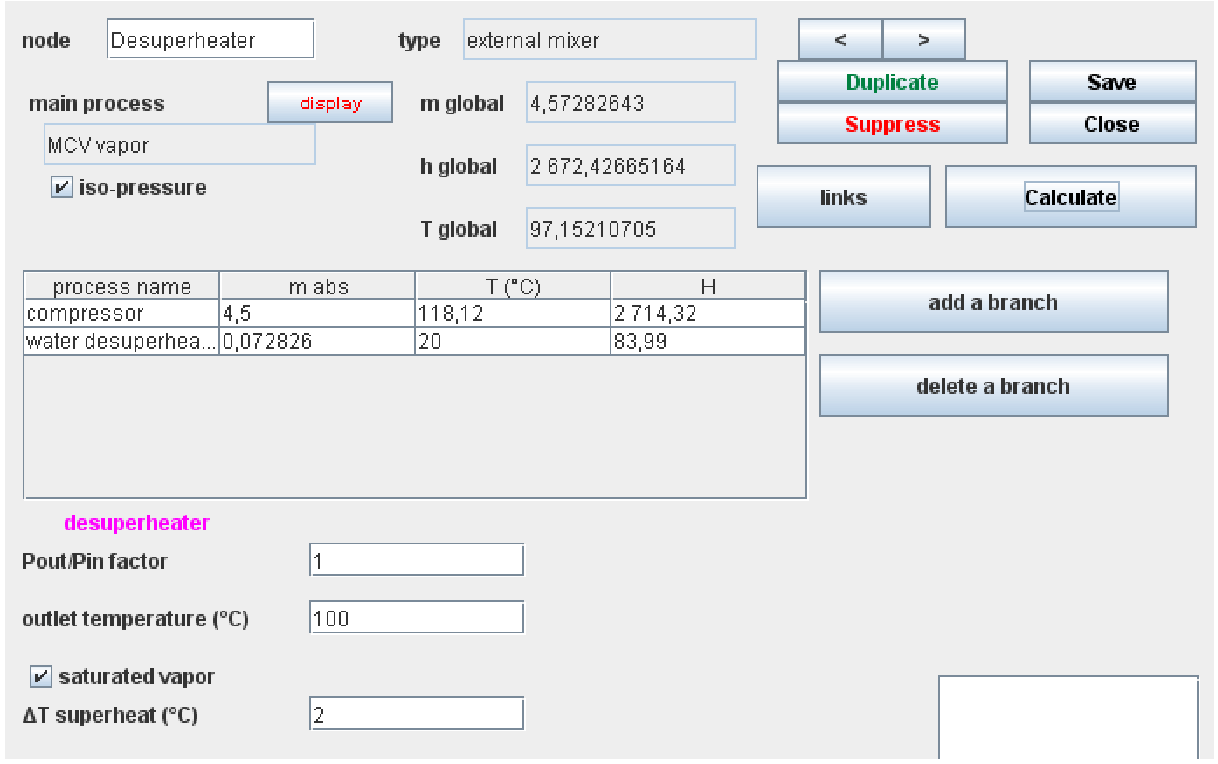
Technologically, the superheating is achieved by spraying a liquid stream in a vapor flow.

One of the difficulties in using such a component is that the injected liquid flow rate depends on the thermodynamic state of the steam, and it is regulated according to the desired state for desuperheated steam. It is not known a priori, so that a simple Thermoptim mixer is not well suited for modeling it.

The DeSuperHeater component we have built (Figure 1) is an external mixer which receives upstream two processes, one representing the steam and other the liquid. It is set by defining the desired output state for the mixture (either by a temperature difference relative to the saturated state, or by its temperature), taking into account any pressure drop. The component determines the flow rate of liquid required, and updates recursively processes which are connected upstream.

Graphical interface of the desuperheater

A graphical interface possible for the desuperheater is given in Figure 2.



node Desuperheater

type external mixer

main process MCV vapor

☑ **iso-pressure**

m global 4,57282643

h global 2 672,42665164

T global 97,15210705

process name	m abs	T (°C)	H
compressor	4,5	118,12	2 714,32
water desuperhea...	0,072826	20	83,99

desuperheater

Pout/Pin factor 1

outlet temperature (°C) 100

☑ **saturated vapor**

ΔT superheat (°C) 2

Figure 2: Desuperheater GUI

The setting used here is: no steam pressure drop (outlet pressure equal to the inlet pressure) and outlet temperature equal to the saturation plus 2 °C of superheating. The necessary desuperheating water flow is determined from the mass and enthalpy balances, and updated in the corresponding process and those upstream.

Thermodynamic model

The model parameters are, firstly, a pressure drop factor relative to the steam pressure, and secondly the temperature to be reached at the end of desuperheating, defined either directly by its value, or relative to the saturation .

The model input data are as follows (provided by other system components):
- The thermodynamic state of steam, including its pressure and enthalpy h_v
- The thermodynamic state of desuperheating water, and in particular its enthalpy h_l
- Flow of steam m_v

The outputs are:
- The thermodynamic state of the desuperheated steam and in particular enthalpy h_{vd}
- Flow of desuperheating water m_l
- Flow of desuperheated steam m_{vd}

The outlet pressure and temperature allowing one to determine the enthalpy of desuperheated steam h_{vd}, the desuperheater equations correspond to a linear system of equations with two unknowns, m_l and m_{vd}:

The conservation of mass provides: $m_{vd} = m_v + m_l$
The conservation of enthalpy gives $m_v h_v + m_l h_l = m_{vd} h_{vd}$

This system is solved without any difficulty and leads to results in Figure 3.

Computer implementation

GUI

The component GUI is given in Figure 2. It allows one to build the lower third of the, the rest being defined as a standard Thermoptim.

The parameters correspond to the four lines added, and calculation results automatically appear in the rest of the screen. The input data are supplied by processes upstream of the system in which the component is inserted: steam flow rate and upstream point states.

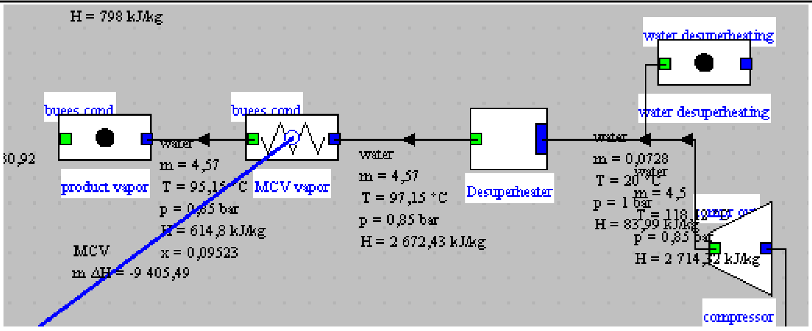


Figure 3: External mixer representing the desuperheater, with its connections

Physical model

With the previous notations, the model equation is as follows, and the calculation of the output rate is trivial:

$$m_l (h_{vd} - h_l) = m_v (h_v - h_{vd})$$

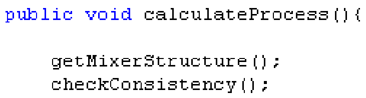
Specifically, the sequence of calculations is as follows:
1) checking the consistency of the node;
2) updating the component before calculation by loading values of the upstream processes and points;
3) reading the parameters on the screen of the external component;
4) calculation of the state of the downstream point and the flow of liquid;
5) update the external node.

Presentation of the code

Now consider the problems encountered in practice in each of these steps.

1) checking the consistency of the node

The component receiving several upstream processes, it is first necessary to ensure that their structure is consistent with what is expected. It is the role of methods getMixerStructure() and checkConsistency()

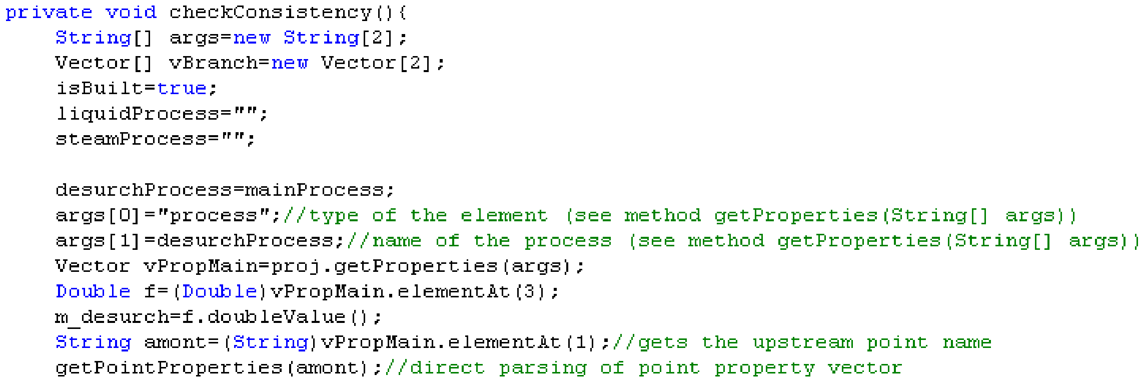


```
public void calculateProcess(){

    getMixerStructure();
    checkConsistency();
```

2) update the component by loading upstream processes and points values

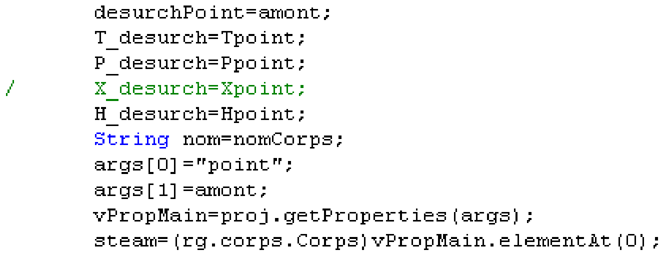
One difficulty here is that an external component has no direct access to simulator variables: these variables are obtained by special generic methods which build Vectors of different structure according to the desired object.

The procedure is not complicated but needs to be respected:



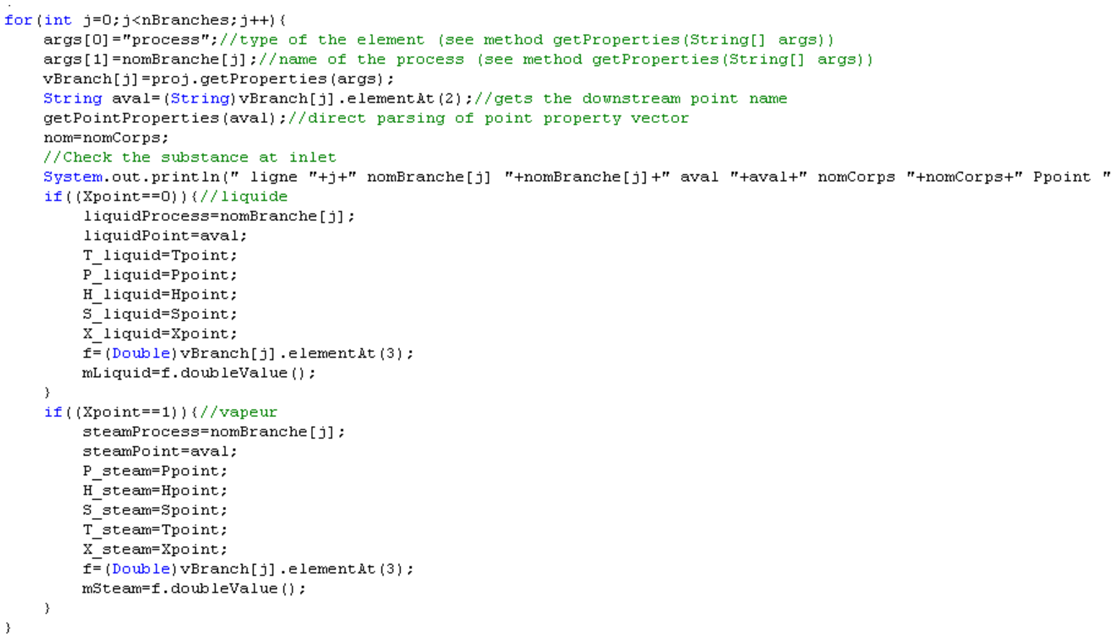
```
private void checkConsistency(){
    String[] args=new String[2];
    Vector[] vBranch=new Vector[2];
    isBuilt=true;
    liquidProcess="";
    steamProcess="";

    desurchProcess=mainProcess;
    args[0]="process";//type of the element (see method getProperties(String[] args))
    args[1]=desurchProcess;//name of the process (see method getProperties(String[] args))
    Vector vPropMain=proj.getProperties(args);
    Double f=(Double)vPropMain.elementAt(3);
    m_desurch=f.doubleValue();
    String amont=(String)vPropMain.elementAt(1);//gets the upstream point name
    getPointProperties(amont);//direct parsing of point property vector
```

This method loads the state from the "upstream" point into variables that can be used to initialize the values which we would then need for calculations, here the substance "steam" represents the steam



```
        desurchPoint=amont;
        T_desurch=Tpoint;
        P_desurch=Ppoint;
/       X_desurch=Xpoint;
        H_desurch=Hpoint;
        String nom=nomCorps;
        args[0]="point";
        args[1]=amont;
        vPropMain=proj.getProperties(args);
        steam=(rg.corps.Corps)vPropMain.elementAt(0);
```

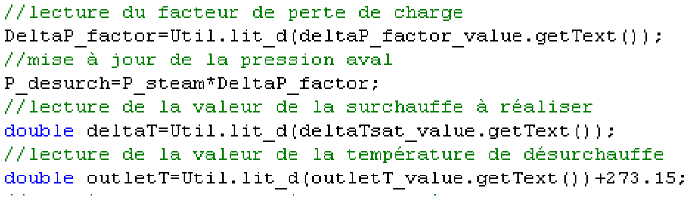
or the state of upstream and downstream points:



```
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);
    String aval=(String)vBranch[j].elementAt(2);//gets the downstream point name
    getPointProperties(aval);//direct parsing of point property vector
    nom=nomCorps;
    //Check the substance at inlet
    System.out.println(" ligne "+j+" nomBranche[j] "+nomBranche[j]+" aval "+aval+" nomCorps "+nomCorps+" Ppoint "
    if((Xpoint==0)){//liquide
        liquidProcess=nomBranche[j];
        liquidPoint=aval;
        T_liquid=Tpoint;
        P_liquid=Ppoint;
        H_liquid=Hpoint;
        S_liquid=Spoint;
        X_liquid=Xpoint;
        f=(Double)vBranch[j].elementAt(3);
        mLiquid=f.doubleValue();
    }
    if((Xpoint==1)){//vapeur
        steamProcess=nomBranche[j];
        steamPoint=aval;
        P_steam=Ppoint;
        H_steam=Hpoint;
        S_steam=Spoint;
        T_steam=Tpoint;
        X_steam=Xpoint;
        f=(Double)vBranch[j].elementAt(3);
        mSteam=f.doubleValue();
    }
}
```

3) reading the parameters on the screen of the external component

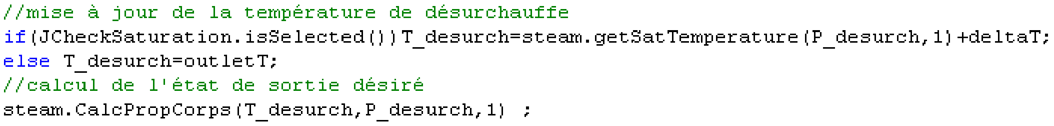
the ExtThopt package provides a number of simple but robust methods to convert in doubles the Strings displayed in the JTextField fields used in the GUI, and vice versa for displaying doubles in these fields. They are implemented as static methods of the class extThopt.Util:



```
//lecture du facteur de perte de charge
DeltaP_factor=Util.lit_d(deltaP_factor_value.getText());
//mise à jour de la pression aval
P_desurch=P_steam*DeltaP_factor;
//lecture de la valeur de la surchauffe à réaliser
double deltaT=Util.lit_d(deltaTsat_value.getText());
//lecture de la valeur de la température de désurchauffe
double outletT=Util.lit_d(outletT_value.getText())+273.15;
```

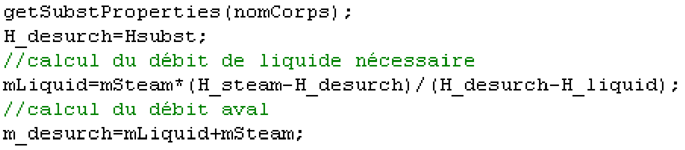
4) calculation of the state of the downstream point and the flow of liquid

Depending on whether one chooses to set the value of superheating or the temperature of superheating, we calculate the output state:



```
//mise à jour de la température de désurchauffe
if(JCheckSaturation.isSelected())T_desurch=steam.getSatTemperature(P_desurch,1)+deltaT;
else T_desurch=outletT;
//calcul de l'état de sortie désiré
steam.CalcPropCorps(T_desurch,P_desurch,1) ;
```

We then calculate the flow of fluid required, and the resulting total flow:

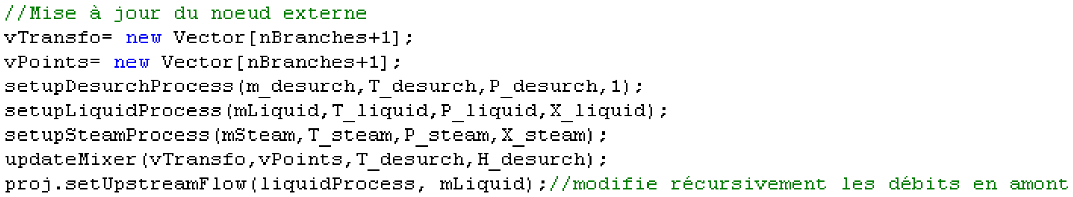


```
getSubstProperties(nomCorps);
H_desurch=Hsubst;
//calcul du débit de liquide nécessaire
mLiquid=mSteam*(H_steam-H_desurch)/(H_desurch-H_liquid);
//calcul du débit aval
m_desurch=mLiquid+mSteam;
```

5) update the external node

The Thermoptim method updateMixer () allows one to update the node from the values loaded into an array of Vectors corresponding to each branch connected to it.

Once updated, the processes connected upstream of the process corresponding to the liquid, if any, see their rate updated.



```
//Mise à jour du noeud externe
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupDesurchProcess(m_desurch,T_desurch,P_desurch,1);
setupLiquidProcess(mLiquid,T_liquid,P_liquid,X_liquid);
setupSteamProcess(mSteam,T_steam,P_steam,X_steam);
updateMixer(vTransfo,vPoints,T_desurch,H_desurch);
proj.setUpstreamFlow(liquidProcess, mLiquid);//modifie récursivement les débits en amont
```

Saving and loading of the model parameters

You can save in project files Thermoptim normal (and then re-read them) the settings of external components by using two specific methods:

public String saveCompParameters()

public void readCompParameters(String ligne_data)

The only constraint is that the entire setting of the external components must fit on one line, with a format compatible with that used in the kernel of the software: the various backup fields are separated by tabs.

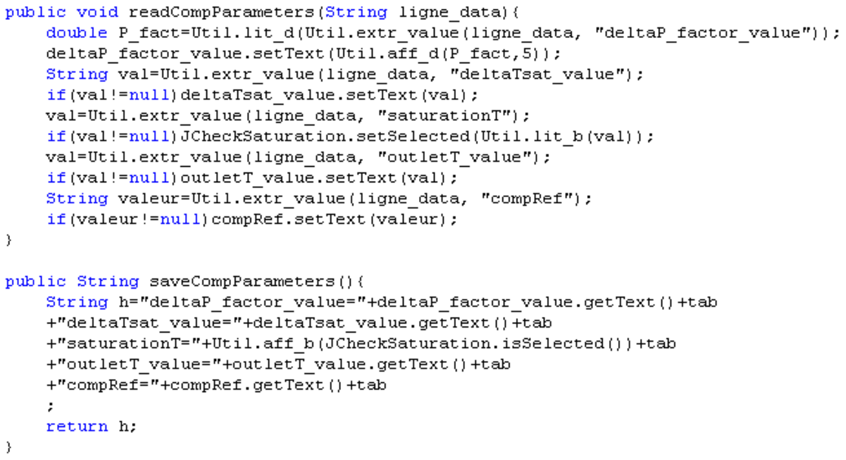
ExtThopt.Util provides a generic method to attribute to the value of a parameter a code allowing one to identify it:

public static String extr_value(String ligne_data, String search)

The backup is performed in the form "parameter = value", and research is in the form:

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

If the parameters of the component is too complex to be saved in this way, nothing prevents a user from using it to save the name of a specific setup file, and then re-read this file as desired.



```
public void readCompParameters(String ligne_data){
    double P_fact=Util.lit_d(Util.extr_value(ligne_data, "deltaP_factor_value"));
    deltaP_factor_value.setText(Util.aff_d(P_fact,5));
    String val=Util.extr_value(ligne_data, "deltaTsat_value");
    if(val!=null)deltaTsat_value.setText(val);
    val=Util.extr_value(ligne_data, "saturationT");
    if(val!=null)JCheckSaturation.setSelected(Util.lit_b(val));
    val=Util.extr_value(ligne_data, "outletT_value");
    if(val!=null)outletT_value.setText(val);
    String valeur=Util.extr_value(ligne_data, "compRef");
    if(valeur!=null)compRef.setText(valeur);
}

public String saveCompParameters(){
    String h="deltaP_factor_value="+deltaP_factor_value.getText()+tab
    +"deltaTsat_value="+deltaTsat_value.getText()+tab
    +"saturationT="+Util.aff_b(JCheckSaturation.isSelected())+tab
    +"outletT_value="+outletT_value.getText()+tab
    +"compRef="+compRef.getText()+tab
    ;
    return h;
}
```