RESTORE

Renewable Energy based seasonal Storage Technology in Order to Raise Economic and environmental sustainability of DHC

D5.1 Modelling of individual component models of the overall RESTORE system and its techno-economic simulation (V1)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101036766.

PROJECT INFORMATION SHEET	
Project Acronym	RESTORE
Project Full Title	Renewable Energy based seasonal Storage Technology in Order to Raise Environmental sustainability of DHC
Grant Agreement	101036766
Call Identifier	H2020-LC-GD-2020-1
Topic	Innovative land-based and offshore renewable energy technologies and their integration into the energy system
Project Duration	48 months (October 2021 – September 2025)
Project Website	www.restore-dhc.eu
Disclaimer	The sole responsibility for the content of this document lies with the authors. It does not necessarily reflect the opinion of the funding authorities. The funding authorities are not responsible for any use that may be made of the information contained herein.

DELIVERABLE INFORMATION SHEET	
Number	Deliverable D5.1
Full Title	D5.1 Modelling of individual components of the overall RESTORE system and its techno-economic simulation (V1)
Related WP	WP5 (RESTORE EU-wide Replication in Virtual Representation on Real Use-Cases)
Related Task	Task 5.1 - Modelling of individual components of the overall RESTORE system. Task 5.2 - RESTORE techno-economic modelling of the integrated system
Lead Beneficiary	SIMTECH (SIM)
Author(s)	Stefan Bergmann (SIM) – s.bergmann@simtechnology.com Fatima Dargam (SIM) - f.dargam@simtechnology.com
Contributor(s)	Javier Baigorri (CEN), Xabier Randez (CEN) Andreas Werner (TUW), Erhard Perz (SIM)
Reviewer(s)	Francisco Cabello (CENER)
Dissemination level	Public
Due Date	March 2024 (M30)
Submission Date	March 31st, 2024
Status	Version 1.0

QUALITY CONTROL ASSESSMENT SHEET			
ISSUE	DATE	COMMENT	AUTHOR
V0.1	21/03/2024	Draft_Version 0.1	Stefan Bergmann (SIM)
V0.2	26/03/2024	Economic Model Section	Javier Baigorri (CENER) Francisco Cabello (CENER)
V0.3	27/03/2024	Contribution to Chapter 3	Andreas Werner (TU WIEN)
V0.4	28/03/2024	Contribution to Deliverable Finalization & Overall Review	Fatima Dargam (SIM) Erhard Perz (SIM)
V0.5	29/03/2024	Final Version Review	Francisco Cabello (CENER)
V1.0	29/03/2024	Submission to the EC	Francisco Cabello (CENER)

Summary

This document provides information about the modelling of individual components of the overall RESTORE system. Deliverable D5.1 is the result of the work carried out in task T5.1 1 led by SIMTECH, during the period of M7 to M30 of the project. The main purpose of this report is to describe the dedicated component models contained in the model library RESTORE_Lib built in task T5.1.

In addition, this deliverable presents the economic model considered in the project. This model is highly flexible, allowing to the user to modify a high number of parameters that are considered for the economic simulations. In addition, the model receives information from the technical simulation to show as results relevant economic variables such as the Net Present Value, the Internal Return Rate and the Levelized Cost of Storage for Electricity and Heat.

The information provided in this document builds upon collaboration between SIMTECH, TU-WIEN, CENER, and POL, as RESTORE partners involved in task T5.1. In the sense, besides D5.1's central scope on the component models development using SIMTECH's simulation tool [2] [3] to build the customized model library for RESTORE, it also considered the outcomes published so far from WP1, WP2, and WP3 (Deliverables: D1.1 - Report on Requirements and Specifications of the Overall Concept [4]; D1.4 on the specifications of RESTORE Use-Cases and Models [5]; D2.3 - Report on TCES Task D2.3 - Small-scale (1-2kW) TCES reactor tested and optimized [6]; D2.4 - Design report of the 30 kW/150 kWh TCES reactor [7]; D2.5 - Dedicated models for the reactor simulation [8]; D3.1 - Numerical model for HPORC systems optimization and application to different Test Cases [9]; as well as D5.10 - RESTORE Replication Strategy V1 [10]).

As part of RESTORE project's participation in the "Open Research Data Pilot", Deliverable D5.1, as a public dissemination-level document, will be made available in the RESTORE open-access research data repository within the Zenodo RESTORE Community (https://zenodo.org/communities/101036766/?page=1&size=20), for further reference and dissemination.

Table of Contents

1. Introduction

This document focuses on the modelling of individual components of the overall RESTORE system, also presenting the economic model for the RESTORE project [1].

D5.1 includes results and software-models related to the work carried out in Task T5.1, concerning the "Modelling of individual components of the overall RESTORE system". The main goal of task T5.1 was twofold: (1) Modelling, fine-tuning & testing all component-models of the overall RESTORE system, based on Partners' received information; as well as (2) Creation & Maintenance of the customized Process Model Library (RESTORE_Lib) for the project, to be used in the simulations of IPSEpro and IPSE GO [2], [3].

The task of modelling the individual components for the overall RESTORE system in T5.1 was directly responsible to feed input information for the development of the the WP5 tasks T5.2 (Techno-Economic Modelling of the Integrated Systems), T5.3 (Web-Platform Adaptation for RESTORE Dynamic and Techno-economic Modelling to represent the Use-Cases), and T5.4 (Implementation, Optimization, Management & Validation of RESTORE Use-Cases using the Simulation Web Platform), with main input from the work packages WP2 and WP3, mostly related to the "TCES dedicated models for the reactor simulation", and the "Dedicated models for the thermodynamic cycle and the dynamic behavior of RESTORE system".

To model individual components of the overall RESTORE system and to be able to use them in system simulations, the model library RESTORE_Lib has been developed in task T5.1 led by SIMTECH. D5.1 includes the results and software-models derived from task T5.1, with an overview of the library content.

In addition, this deliverable also presents the economic model considered in the project, which is a highly flexible model, allowing the user to modify a high number of parameters for performing the economic simulations. The economic model considered for RESTORE, developed in collaboration with CENER, receives information from the technical simulation to show as results relevant economic variables such as the Net Present Value, the Internal Return Rate and the Levelized Cost of Storage for Electricity and Heat.

Besides this Introduction (Chapter 1), this D5.1 document is structured in the following way:

- Chapter (2) provides an overview of the models contained in the RESTORE_Lib, which is the customized model library created for the RESTORE system.
- Chapter (3) presents details of the component models that were specifically developed for the RESTORE project to represent the overall RESTORE system.
- Chapter (4) explains the economic model that was implemented for RESTORE.
- Chapter (5) presents the references that the work done was based upon.

2. Component Models Contained in the RESTORE_Lib

This chapter presents the component models, which are contained in the model library RESTORE_Lib. These models have been developed using the IPSEpro Model Development Kit (MDK).

The tables presented here contain component models which are either RESTORE specific developments, models which are adapted from general IPSE models, or strictly general models shared with other IPSE model libraries.

2.1. Available units

Table 1: Available Units

Unit Name 1	Description	Type
G_OR_Htex	heat exchanger for transfer from gas on hot side to organic fluid on cold side	adapted
G_Pipe	pipe for gas streams	general
G_Sink	sink for a gas stream	general
G_Source	source for a gas stream	general
OR_Boiler	simple boiler model for ORC fluids	adapted
OR_Compressor	compressor for ORC fluids	adapted
OR_Condenser	condenser for ORC fluids, water cooled	adapted
OR_Condenser_a	condenser for ORC fluids, air cooled, dry	adapted
OR_Connector	connector for ORC streams to be used in closed loops	adapted
OR_Expander	expander for ORC fluids	adapted
OR_G_Htex	heat exchanger for transfer from ORC fluid on hot side to gas on cold side	adapted
OR_Heat_sink	heat sink for usage with OR streams	adapted
OR_Heat_source	heat source for usage with OR streams	adapted
OR_Htex	general purpose heat exchanger for ORC fluids	adapted
OR_Mixer	mixer for ORC streams	adapted
OR_Pipe	pipe for ORC fluids	adapted
OR_Pump	pump for ORC fluids	adapted
OR_Separator	vapour-liquid separator for ORC fluids	adapted
OR_Sink	sink for an ORC stream	adapted
OR_Source	source for an ORC stream	adapted

OR_Splitter	splitter for ORC streams	adapted
OR_T_Htex	heat exchanger for transfer from ORC fluid on hot side to thermofluid on cold side	adapted
OR_Turbine	turbine for ORC fluids	adapted
OR_Valve	valve for ORC fluid	adapted
OR_W_Htex	heat exchanger for transfer from ORC fluid on hot side to water on cold side	adapted
OR_Xprescription	prescription/calculation of vapor quality of an ORC fluid	adapted
OR_wall_htex	heat exchanger with wall transferring heat from organic fluid (OR) to another side	specific
TCM_Heat_sink	heat sink for TCM	specific
TCM_Heat_source	heat source for TCM	specific
TCM_Htex	heat exchanger for transfer from TCM fluid on hot side to TCM fluid on cold side	specific
TCM_Mixer	mixer for TCM streams	specific
TCM_Pump	pump for TCM fluids	specific
TCM_Reactor_Charging	Reactor for charging step of TCM. High temperature side transferring heat to the reactor is optional. Different heat delivering working fluids (OR_ or T_) can be connected.	specific
TCM_Reactor_Discharging	Reactor for discharging step of TCM. Low temperature side receiving heat from the reactor is optional. Different heat receiving working fluids (OR_or T_) can be connected.	specific
TCM_Separator	separator for TCM stream	specific
TCM_Sink	sink for a TCM stream	specific
TCM_Source	source for a TCM stream	specific
TCM_Splitter	splitter for TCM streams	specific
TCM_T_Htex	heat exchanger for transfer from TCM fluid on hot side to thermofluid on cold side	specific
TCM_Valve	valve for TCM stream	specific
TCM_W_Separator	separator for water from TCM stream	specific
T_Connector	connector for heat transfer fluids to be used in closed loops	general
T_Heat_sink	heat sink for heat transfer fluids	general
T_Heat_source	heat source for heat transfer fluids	general

T_Htex	general purpose heat exchanger for heat transfer fluids	general
T_Mixer	mixer for heat transfer fluid streams	general
T_OR_Htex	heat exchanger for transfer from thermofluid on hot side to ORC fluids on cold side	adapted
T_Pipe	pipe for heat transfer fluids	general
T_Pump	pump for heat transfer fluids	general
T_Sink	sink for a heat transfer fluid stream	general
T_Source	source for a heat transfer fluid	general
T_Splitter	splitter for heat transfer fluid streams	general
T_TCM_Htex	heat exchanger for transfer from thermofluid on hot side to TCM fluid on cold side	specific
T_W_Htex	heat exchanger for transfer from thermofluid on hot side to water on cold side	general
T_wall_htex	heat exchanger with wall transferring heat from thermooil (T) to another side	specific
W_Compressor	compressor for steam	general
W_Connector	connector for closed loops	general
W_Heat_sink	heat sink for water streams	general
W_Heat_source	heat source for water streams	general
W_Mixer	mixer for water streams	general
W_OR_Htex	heat exchanger for transfer from water on hot side to OR fluid on cold side	adapted
W_Pipe	pipe for water	general
W_Pump	pump for water	general
W_Sink	sink for a water stream	general
W_Source	source for a water stream	general
W_Splitter	splitter for water streams	general
W_T_Htex	heat exchanger for transfer from water on hot side to thermofluids on cold side	general
W_Valve	valve for water	general
W_Xprescription	prescription/calculation of steam quality	general
free_var	free variable	general
gear	gears	general
generator	generator	general
mech_loss	mechanical loss	general

motor	motor	general
optimization	optimization element	general
wall_OR_htex	heat exchanger with wall transferring heat from wall to organic fluid (OR)	specific
wall_T_htex	heat exchanger with wall transferring heat from wall to thermooil (T)	specific

2.2. Available connections

Table 2: Available Connections

Connection Name 1	Description	Type
G_Stream	stream using gas composition	general
OR_Stream	stream using working fluids for ORC, refrigeration and heat pump processes	adapted
TCM_Stream	thermochemical storage material (TCM) stream	specific
T_Stream	stream representing a heat transfer fluid (thermofluid)	general
W_Stream	stream for water	general
q_cond_trans	conductive heat transfer through a wall (solid boundary)	specific
Shaft	shaft	general

2.3. Available global objects

Table 3: Available Globals

Global Name 1	Description	Type
G_Composition	chemical composition of a gaseous working fluid	general
OR_Composition	working fluid (usually organic hydrocarbons and some other alternatives)	adapted
TCM	thermochemical material working pair	specific
T_Composition	heat transfer fluid	general

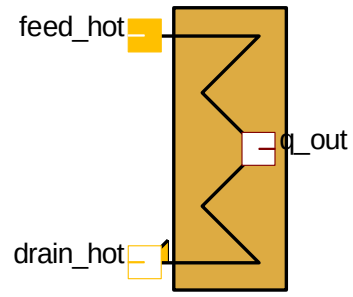
3. Component Model Details

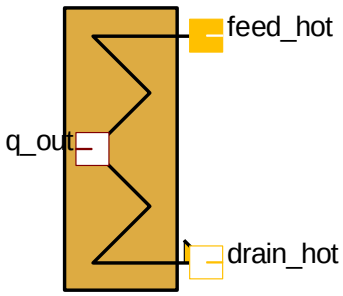
This chapter provides details of the RESTORE specific component models contained in RESTORE_Lib. It presents the model description, lists the model equations and variables representing individual models, and displays the icons representing the individual units.

3.1. OR_wall_htex

Purpose

Heat exchanger with wall transferring heat from organic fluid (OR) to another side.





Connections

OR_Stream: feed_hot
OR_Stream: drain_hot
q_cond_trans: q_out

3.1.1. OR_wall_htex

Purpose

design model

Model equations

```
# heat transfer from fluid inside the tubes through the wall

# mass transfer
f_mass:   feed_hot.mass = drain_hot.mass;

# pressure drop through the wall tubes
```

```
f_delta_p:          feed_hot.p - delta_p = drain_hot.p;

# hot fluid is cooled down
f_energy:feed_hot.mass*feed_hot.h - q_trans = drain_hot.mass*drain_hot.h;

# temperatue change
f_delta_t: feed_hot.t - delta_t = drain_hot.t;

# conductive heat transfer through the wall
f_q_q_trans:        q_trans = q_out.q_trans;

# transfer of prevailing temperatures
ifl Type == cocurrent then
  f_t_hot_in_co:   q_out.t_hot_in = feed_hot.t;
  f_t_hot_out_co: q_out.t_hot_out = drain_hot.t;
endifl

ifl Type == counter_current then
  f_t_hot_in_counter:      q_out.t_hot_in = drain_hot.t;
  f_t_hot_out_counter:     q_out.t_hot_out = feed_hot.t;
endifl

# additional calculations and data for operating points of the working fluid with phase change
(inside the tubes)
# heat is flowing out, working fluid is cooled and may condense

# pressure drop is assumed to be linear between feed and drain

# intermediate point is either inlet conditions (if already 2-phase) or where condensation starts
(saturated vapour)
f_h_II_aux:         h_II_aux = feed_hot.Composition.o_h_px(feed_hot.p, 1.0);

f_h_II:     if (feed_hot.h > h_II_aux) then
                        h_II = feed_hot.Composition.o_h_px(p_II, 1.0);
            else
                        h_II = feed_hot.h;
f_p_II:     if (feed_hot.h > h_II_aux) then
                        p_II = drain_hot.p + (h_II-drain_hot.h)/(feed_hot.h-
drain_hot.h)*(feed_hot.p - drain_hot.p);
            else
                        p_II = feed_hot.p;

f_t_II: if (feed_hot.h > h_II_aux) then
                        t_II = feed_hot.Composition.o_tsat_p(p_II);
            else
                        t_II = feed_hot.t;

f_s_II:     if (feed_hot.h > h_II_aux) then
                        s_II = feed_hot.Composition.o_s_px(p_II, 1.0);
            else
                        s_II = feed_hot.s;

# condensation end point is saturated liquid
```

```
f_h_sl:    h_sl = feed_hot.Composition.o_h_px(p_sl, 0.0);
f_p_sl:    p_sl = drain_hot.p + (h_sl-drain_hot.h)/(feed_hot.h-drain_hot.h)*(feed_hot.p -
drain_hot.p);
f_t_sl:    t_sl = feed_hot.Composition.o_tsat_p(p_sl);
f_s_sl:    s_sl = feed_hot.Composition.o_s_px(p_sl, 0.0);

# test  conditions
t_q_trans:          test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:                  test (delta_p >= 0.0) warning "delta_p negative";

# feed and drain terminal must reference the same composition
ifl ref(feed_hot.Composition) != ref(drain_hot.Composition) then
  t_Comp:           test(1!=1) error "Composition must be the same at feed and drain";
endifl
```

Parameters

delta_p	absolute pressure drop

Variables

delta_t	temperature difference between feed and drain mass flow
q_trans	transferred heat
p_II	intermediate pressure (sat vap resp. inlet)
t_II	intermediate temperature (sat vap resp. inlet)
h_II	intermediate enthalpy (sat vap resp. inlet)
s_II	intermediate entropy (sat vap resp. inlet)
h_II_aux	auxiliary vapour enthalpy
p_sl	saturated condensate pressure
t_sl	saturated condensate temperature
h_sl	saturated condensate enthalpy
s_sl	saturated condensate entropy

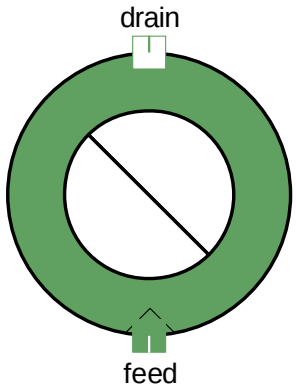
Switches

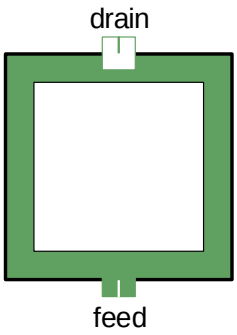
Type (cocurrent, counter_current)	defines the type of heat exchanger

3.2. TCM_Heat_sink

Purpose

Heat sink or general heat consumer in a heat network for thermochemical materials:





Connections

TCM_Stream: feed
TCM_Stream: drain

3.2.1. TCM_Heat_sink

Purpose

default model

Model equations

```
# mass balance

f_mass:                 feed.mass = drain.mass;
#f_massOil:             drain.mass_Oil = feed.mass_Oil;
f_massMaterial1: feed.mass_Material1 = drain.mass_Material1;
f_massMaterial2: feed.mass_Material2 = drain.mass_Material2;
f_massWater:            feed.mass_Water = drain.mass_Water;

# energy balance
# q_trans > 0

f_energy: feed.mass * feed.h - q_trans = drain.mass * drain.h;

f_delta_t: feed.t - delta_t = drain.t;
```

```
f_delta_p:          feed.p - delta_p = drain.p;

# test  conditions

t_q_trans:          test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:          test (delta_p >= 0.0) warning "delta_p negative";
t_delta_t: test (delta_t >= 0.0) warning "delta_t negative";

# no change of substances
t_conn_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
                        error "different material 1 at feed and drain!";
t_conn_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                        error "different material 2 at feed and drain!";
t_conn_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
                        error "different oil at feed and drain!";
```

Parameters

delta_p	absolute pressure drop

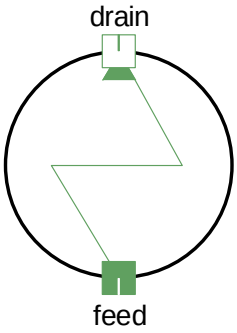
Variables

q_trans	transferred heat
delta_t	temperature difference between feed and drain mass flow

3.3. TCM_Heat_source

Purpose

General heat source in a heat network for thermochemical materials.



Connections

TCM_Stream: feed
TCM_Stream: drain

3.3.1. TCM_Heat_source

Purpose

default model

Model equations

```
# mass balance

f_mass:                 feed.mass = drain.mass;
#f_massOil:             drain.mass_Oil = feed.mass_Oil;
f_massMaterial1: feed.mass_Material1 = drain.mass_Material1;
f_massMaterial2: feed.mass_Material2 = drain.mass_Material2;
f_massWater:            feed.mass_Water = drain.mass_Water;

# energy balance

f_energy: q_trans = drain.mass * drain.h - feed.mass * feed.h;

f_delta_t: feed.t + delta_t = drain.t;
f_delta_p:      feed.p - delta_p = drain.p;

# test  conditions
t_q_trans:      test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:      test (delta_p >= 0.0) warning "delta_p negative";
t_delta_t: test (delta_t >= 0.0) warning "delta_t negative";

# no change of substances
t_conn_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
                        error "different material 1 at feed and drain!";
t_conn_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                        error "different material 2 at feed and drain!";
t_conn_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
```

error "different oil at feed and drain!";

Parameters

delta_p	absolute pressure drop

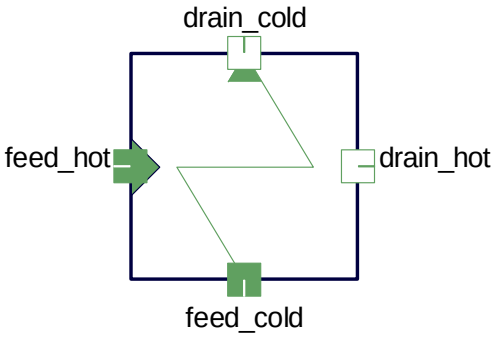
Variables

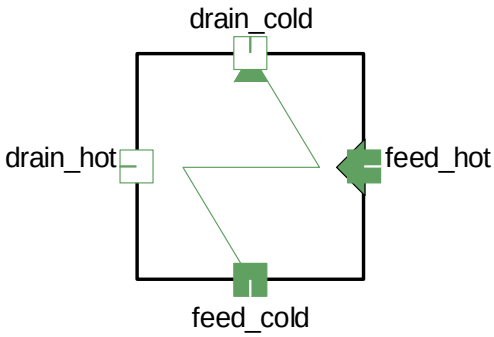
q_trans	transferred heat
delta_t	temperature difference between drain and feed

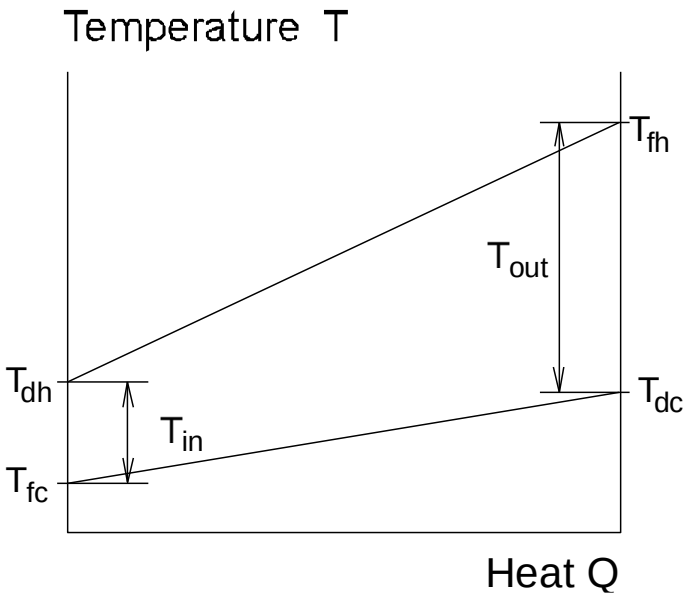
3.4. TCM_Htex

Purpose

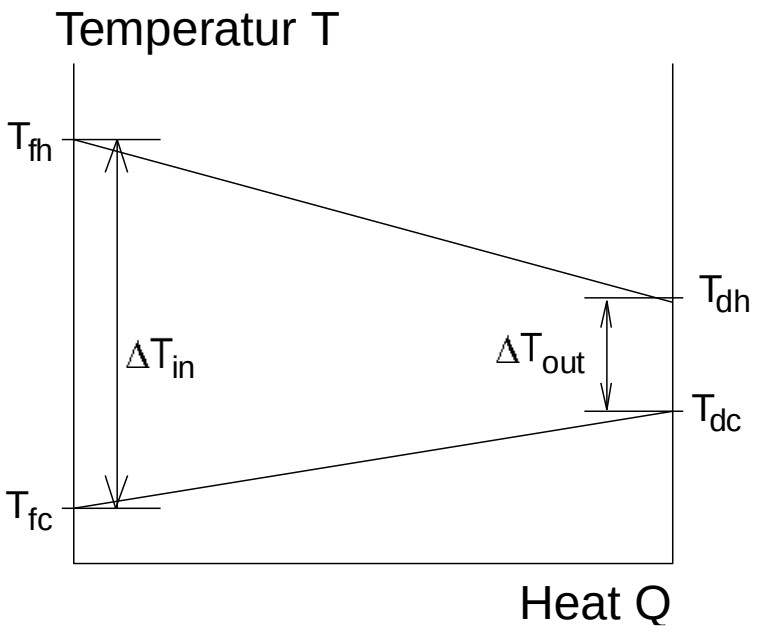
Heat exchanger for transfer from TCM fluid on the hot side to TCM fluid on the cold side. Cocurrent or counter current flow.







QT-diagram, counter current



QT-diagram, cocurrent

Connections

TCM_Stream: drain_hot
TCM_Stream: feed_hot
TCM_Stream: drain_cold
TCM_Stream: feed_cold

3.4.1. TCM_Htex

Purpose

design model

Model equations

```
# mass balances

# hot side total and individual mass balances (one is redundant)
f_mass_hot:                 feed_hot.mass = drain_hot.mass;
#f_mass_Oil_hot:        feed_hot.mass_Oil = drain_hot.mass_Oil;
f_mass_Material1_hot:       feed_hot.mass_Material1 = drain_hot.mass_Material1;
f_mass_Material2_hot:       feed_hot.mass_Material2 = drain_hot.mass_Material2;
f_mass_Water_hot:           feed_hot.mass_Water = drain_hot.mass_Water;

# cold side total and individual mass balances (one is redundant)
f_mass_cold:                feed_cold.mass = drain_cold.mass;
#f_mass_Oil_cold:           feed_cold.mass_Oil = drain_cold.mass_Oil;
f_mass_Material1_cold:  feed_cold.mass_Material1 = drain_cold.mass_Material1;
f_mass_Material2_cold:  feed_cold.mass_Material2 = drain_cold.mass_Material2;
f_mass_Water_cold:          feed_cold.mass_Water = drain_cold.mass_Water;

# pressure drops

f_delta_p_hot:          feed_hot.p - delta_p_hot = drain_hot.p;
f_delta_p_cold:         feed_cold.p - delta_p_cold = drain_cold.p;

# energy balance
```

```
f_e_hot:  feed_hot.mass * (feed_hot.h-drain_hot.h) * (1.0 - heat_loss/100) = q_trans;
f_e_cold: feed_cold.mass * (drain_cold.h - feed_cold.h) = q_trans;

# temperature differences
# They are differently defined for co- and counter current heat exchangers.

ifl Type == cocurrent then
  f_dt_in_co:       feed_hot.t - dt_in = feed_cold.t;
  f_dt_out_co:      drain_hot.t - dt_out = drain_cold.t;
endifl

ifl Type == counter_current then
  f_dt_in_counter:          drain_hot.t - dt_in = feed_cold.t;
  f_dt_out_counter:         feed_hot.t - dt_out = drain_cold.t;
endifl

# Mean temperature difference is the logarithmic temperature difference.
# For nearly equal inlet and outlet temperature differences, it is approximated
# by the arithmetic mean (which is the asymptotic approximation).
f_MTD: if abs(dt_in/dt_out) >=1.2 || abs(dt_out/dt_in) >=1.2 then
            MTD * ln(dt_in/dt_out) = (dt_in-dt_out);
   else
            MTD = (dt_in+dt_out)/2.0;

f_htc_area:         q_trans = htc_area * MTD;

# tests
t_dt_in:    test(dt_in > 0.0) error "dt_in <= 0.0";
t_dt_out: test(dt_out > 0.0)          error "dt_out <= 0.0";
t_q_trans:          test(q_trans > 0.0)         error "q_trans <= 0.0";

# no change of substances
t_drain_hot_material1: test (feed_hot.TCM.ID1 == drain_hot.TCM.ID1)
                                    error "Different material 1 at feed_hot and drain_hot!";
t_drain_hot_material2: test (feed_hot.TCM.ID2 == drain_hot.TCM.ID2)
                                    error "Different material 2 at feed_hot and drain_hot!";
t_drain_hot_oil: test (feed_hot.T_Composition.FluidID == drain_hot.T_Composition.FluidID)
                                error "Different oil at feed_hot and drain_hot!";
t_drain_cold_material1: test (feed_cold.TCM.ID1 == drain_cold.TCM.ID1)
                                    error "Different material 1 at feed_cold and drain_cold!";
t_drain_cold_material2: test (feed_cold.TCM.ID2 == drain_cold.TCM.ID2)
                                    error "Different material 2 at feed_cold and drain_cold!";
t_drain_cold_oil: test (feed_cold.T_Composition.FluidID == drain_cold.T_Composition.FluidID)
                                error "Different oil at feed_cold and drain_cold!";
```

Parameters

delta_p_hot	pressure drop of the high temperature side
delta_p_cold	pressure drop of the low temperature side
heat_loss	percentage of heat lost to the surroundings

Variables

dt_in	temperature difference at the inlet of the low temperature side
dt_out	temperature difference at the outlet of the low temperature side
htc_area	heat transfer coefficient x transfer area
MTD	mean temperature difference
q_trans	transferred heat

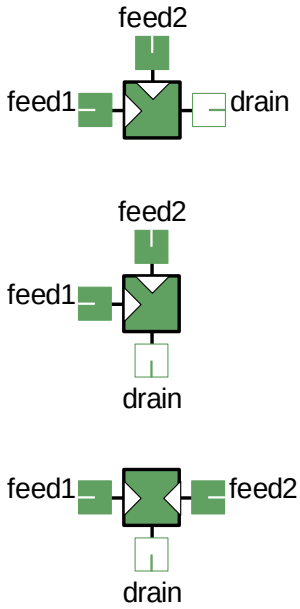
Switches

Type (cocurrent, counter_current)	defines the type of heat exchanger

3.5. TCM_Mixer

Purpose

Mixer for TCM streams.



Connections

TCM_Stream: feed1
TCM_Stream: feed2
TCM_Stream: drain

3.5.1. TCM_Mixer

Purpose

mixer model for two incoming streams

Model equations

```
# mass balances

# overall
f_mass:                 feed1.mass + feed2.mass = drain.mass;

# one of the individual mass balances is redundant and hence omitted
#f_mass_Oil:            feed1.mass_Oil + feed2.mass_Oil = drain.mass_Oil;
f_mass_Material1:       feed1.mass_Material1 + feed2.mass_Material1 =
drain.mass_Material1;
f_mass_Material2:       feed1.mass_Material2 + feed2.mass_Material2 =
drain.mass_Material2;
f_Mass_Water:           feed1.mass_Water + feed2.mass_Water = drain.mass_Water;

# pressure drops
f_p1: feed1.p - delta_p_1 = drain.p;
f_p2: feed2.p - delta_p_2 = drain.p;

# energy balance
```

```
f_energy:feed1.h * feed1.mass + feed2.h * feed2.mass = drain.h * drain.mass;

# test for positive pressure drops
t_delta_p_1: test (delta_p_1>=0.0)        warning "pressure drop delta_p_1 is negative";
t_delta_p_2: test (delta_p_2>=0.0)        warning "pressure drop delta_p_2 is negative";

# substances must be the same
t_feed1_material1: test (feed1.TCM.ID1 == drain.TCM.ID1)
                          error "different material 1 at feed1 and drain!";
t_feed1_material2: test (feed1.TCM.ID2 == drain.TCM.ID2)
                          error "different material 2 at feed1 and drain!";
t_feed1_oil: test (feed1.T_Composition.FluidID == drain.T_Composition.FluidID)
                  error "different oil at feed1 and drain!";
t_feed2_material1: test (feed2.TCM.ID1 == drain.TCM.ID1)
                          error "different material 1 at feed2 and drain!";
t_feed2_material2: test (feed2.TCM.ID2 == drain.TCM.ID2)
                          error "different material 2 at feed2 and drain!";
t_feed2_oil: test (feed2.T_Composition.FluidID == drain.T_Composition.FluidID)
                  error "different oil at feed2 and drain!";
```

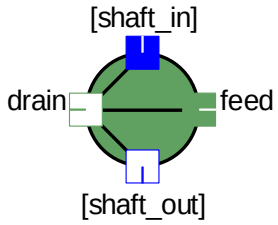
Variables

delta_p_1	absolute pressure drop of a stream attached to feed_1
delta_p_2	absolute pressure drop of stream attached to feed_2

3.6. TCM_Pump

Purpose

Pump for TCM fluids.



Connections

shaft: shaft_in
shaft: shaft_out
shaft: shaft_in
shaft: shaft_out
TCM_Stream: feed
TCM_Stream: drain

3.6.1. TCM_Pump

Purpose

design model

Model equations

```
# mass balances
# total and individual (one is redundant)
f_mass:   feed.mass = drain.mass;
f_mass_Material1:          feed.mass_Material1 = drain.mass_Material1;
f_mass_Material2:          feed.mass_Material2 = drain.mass_Material2;
f_mass_Water:    feed.mass_Water = drain.mass_Water;
# f_mass_Oil:      drain.mass_Oil = drain.mass_Oil;


# pump efficiency definition
f_eta_p:  (drain.p - feed.p)*feed.v*100 / eta_p = drain.h - feed.h;

# both sides connected
ifl ref(shaft_in) && ref(shaft_out) then
  f_powera:         (feed.h - drain.h)* feed.mass /eta_m + shaft_in.power - shaft_out.power = 0.0;
endifl

# left side shaft only
ifl ref(shaft_in) && !ref(shaft_out) then
  f_powerb:         (feed.h - drain.h)* feed.mass / eta_m + shaft_in.power = 0.0;
endifl

# right side shaft only
ifl !ref(shaft_in) && ref(shaft_out) then
  f_powerc:         (feed.h - drain.h) * feed.mass / eta_m - shaft_out.power = 0.0;
endifl
```

```
# test conditions
t_delta_p:      test((drain.p - feed.p) >= 0.0)    warning "outlet pressure lower than inlet
pressure";
t_eta_p_low:    test ( eta_p >= 0.0)               error "pump efficiency < 0.0";
t_eta_p_high:   test ( eta_p <= 1.0)               error "pump efficiency > 1.0";
t_eta_m_low:    test ( eta_m >= 0.0)               error "mechanical efficiency < 0.0";
t_eta_m_high:   test ( eta_m <= 1.0)               error "mechanical efficiency > 1.0";

# no water vapour at inlet
t_p_cav:  test (if(feed.w_Water < 1e-6) then feed.p_required else feed.p >= feed.p_required)
                    warning "water vapour at inlet, cavitation will occur!";

# no change of substances
t_conn_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
                            error "different material 1 at feed and drain!";
t_conn_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                            error "different material 2 at feed and drain!";
t_conn_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
                    error "different oil at feed and drain!";
```

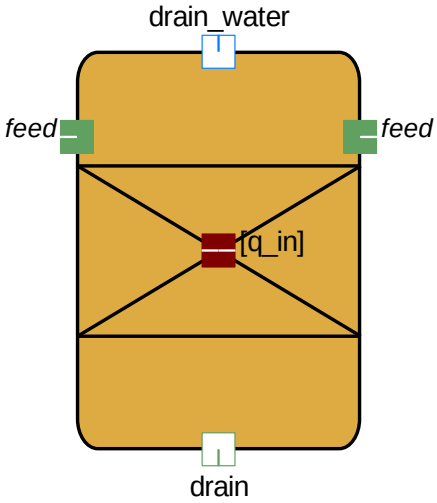
Parameters

eta_p	pump efficiency
eta_m	mechanical efficiency

3.7. TCM_Reactor_Charging

Purpose

Reactor for charging step of thermochemical material (TCM). High temperature side transferring heat to the reactor is optional. Different heat delivering working fluids (OR_ or heat transfer (T_) fluids) can be connected.



Connections

TCM_Stream: feed
TCM_Stream: drain
W_Stream: drain_water
q_cond_trans: q_in

3.7.1. TCM_Reactor_Charging

Purpose

Design model for isothermal reactor charging the TCM.

Model equations

```
# mass balances

# Due to the summary balance in the TCM_stream, one of the mass balances is redundant.
# The overall mass balance is used just for testing correctness, see test section
#f_mass:            feed.mass = drain.mass + drain_water.mass;

f_mass_oil:                 feed.mass_Oil = drain.mass_Oil;
f_mass_Material1:           drain.mass_Material1 = (1 - conv_TCM) * feed.mass_Material1;
f_mass_Material2:           drain.mass_Material2 = feed.mass_Material2 + conv_TCM *
feed.mass_Material1 / molarmass_Material1 * molarmass_Material2;

f_mass_Water:               drain_water.mass= ny_H2O_TCM * conv_TCM *
feed.mass_Material1/molarmass_Material1 * 18.0153;

# There must not be any water in feed, see test section.
# All water goes to drain_water, no water in exiting TCM
f_no_Water_drain:           drain.mass_Water = 0.0;
```

```
#---------------------
f_molarmass_Material1: molarmass_Material1 = feed.TCM.TCM_M(feed.TCM.ID1);
f_molarmass_Material2: molarmass_Material2 = feed.TCM.TCM_M(feed.TCM.ID2);

# stochiometric coefficient for the extracted water:
f_ny_H2O:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  ny_H2O_TCM = 1.0;
else
#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  ny_H2O_TCM = 4.0;
else
#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  ny_H2O_TCM = 2.0;
else
#Potassiumcarbonate 1.5 hydrate:
if feed.TCM.ID1 <= 7.1 then
  ny_H2O_TCM = 1.5;
else
  ny_H2O_TCM = -1.;
#----------------------
# pressure balance

f_p1:               drain.p = feed.p - delta_p;
f_p2:               drain.p = drain_water.p;

# energy balance

# temperature

f_t1:               drain_water.t = t_reac;
f_t2:               drain.t = t_reac;

# heat transfer

f_q_trans:          q_trans = q_reac + q_preheat;

ifl ref(q_in) then   # the hot side is connected

  # conductive heat transfer through the wall
  f_q_q_trans:      q_trans = q_in.q_trans;

  # transfer of prevailing temperatures at the heat transfer surface
  # the reactor is isothermal conditions
  f_t_cold_in:      q_in.t_cold_in = t_reac;
  f_t_cold_out:     q_in.t_cold_out = t_reac;
endifl

# preheating, considering there is no water in the feed
```

```
f_q_preheat:       q_preheat = drain.h_Material1 * feed.mass_Material1 + drain.h_Material2 *
feed.mass_Material2 + drain.h_Oil * feed.mass_Oil - feed.h * feed.mass;

# reaction of dehydration

f_q_reac:q_reac = (hWater - 285.83 / 18.0153 * 1000) * ((ny_H2O_TCM * conv_TCM *
feed.mass_Material1 / molarmass_Material1 * 18.0153) ) +
                  ((drain.mass_Material2 - feed.mass_Material2) ) * (drain.h_Material2 +
H_298_15_Material2 / molarmass_Material2 * 1000) -
                  ((feed.mass_Material1 - drain.mass_Material1) ) * (drain.h_Material1 +
H_298_15_Material1 / molarmass_Material1 * 1000);

f_hWater:                  hWater = drain_water.fhpt(drain.p, drain.t, 1,0,0,0,0,0,0,0,0,0,0,0) -
drain_water.fhpt(1, 25, 1,0,0,0,0,0,0,0,0,0,0,0);

# enthalpies of formation of the working pair materials

f_H_298_15_Material1:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  H_298_15_Material1 = - 1093.99;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  H_298_15_Material1 = -2276.512;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  H_298_15_Material1 = - 1403.9;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  H_298_15_Material1 = -1612.930;
else
  H_298_15_Material1 = 0.0;
#--------------------------

f_H_298_15_Material2:
#Metaboric acid:
if feed.TCM.ID2 <= 2.1 then
  H_298_15_Material2 = - 802.78;
else

#Coppersulphate monohydrate:
if feed.TCM.ID2 <= 4.1 then
  H_298_15_Material2 = -1082.818;
else

#Calciumchloride:
if feed.TCM.ID2 <= 6.1 then
```

```
  H_298_15_Material2 = - 795.8;
else

#Potassiumcarbonate:
if feed.TCM.ID2 <= 8.1 then
  H_298_15_Material2 = -1151.499;
else
  H_298_15_Material2 = 0.0;

# minimum and maximum temperatures of the reactions that can be used as range
temperatures for the charging reactor

f_t_min:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  t_min = 145.0;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  t_min = 80.0;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  t_min = 175.0;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  t_min = 135.0;
else
  t_min = 10.0;
#--------------------------

f_t_max:
#Metaboric acid:
if feed.TCM.ID2 <= 2.1 then
  t_max = 165.0;
else

#Coppersulphate monohydrate:
if feed.TCM.ID2 <= 4.1 then
  t_max = 130.0;
else

#Calciumchloride:
if feed.TCM.ID2 <= 6.1 then
  t_max = 210.0;
else

#Potassiumcarbonate:
if feed.TCM.ID2 <= 8.1 then
  t_max = 150.0;
```

```
else
  t_max = 0.0;


#--------------------------
f_delta_H_reac:   delta_H_reac = (drain.h_Material2 * molarmass_Material2 / 1000 +
H_298_15_Material2 + ny_H2O_TCM * (hWater * 18.0153 / 1000 - 285.83) -
                        (feed.h_Material1 * molarmass_Material1 / 1000 + H_298_15_Material1)) /
molarmass_Material1 * 1000;

# equilibrium curves
# See Deliverable 2.4 eq. (2) and Table 2 for details. Data from experiments are used.

f_delta_H_reac_0:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  delta_H_reac_0 = 125.2;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  delta_H_reac_0 = 124.2;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  delta_H_reac_0 = 60.5;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  delta_H_reac_0 = 158.6;
else
  delta_H_reac_0 = 0.0;

f_delta_S_reac_0:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  delta_S_reac_0 = 298.1;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  delta_S_reac_0 = 328.0;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  delta_S_reac_0 = 135.5;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  delta_S_reac_0 = 375.6;
else
```

```
  delta_S_reac_0 = 0.0;

# stochiometric coefficient for the equilibrium reactions (as they are different from
ny_H2O_TCM):
f_ny_equilib:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  ny_equilib = 1.0;
else
#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  ny_equilib = 2.0;          # reaction product is trihydrate
else
#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  ny_equilib = 1.0;          # reaction product is monohydrate
else
#Potassiumcarbonate 1.5 hydrate:
if feed.TCM.ID1 <= 7.1 then
  ny_equilib = 1.5;
else
  ny_equilib = -1.;


# Van't Hoff curve gives the equilibrium temperature-pressure relationship
# The reference pressure p+ is 1 bar and is cancelled out of the equation
# reactor pressure is drain TCM outlet pressure
f_t_equilib: ln(drain.p) = delta_S_reac_0/8.314/ny_equilib -
delta_H_reac_0*1000/8.314/ny_equilib/(t_equilib+273.15);

# test conditions
t_conv_TCM1:          test (conv_TCM >= 0.0) error "Negative conversion rate!";
t_conv_TCM2:          test (conv_TCM <= 1.0) error "Conversion rate higher than 1!";
t_q_reac:             test (q_reac >= 0.0)          error "Heat consumed by reaction is
negative!";

# temperature range for charging
t_t_min:  test (t_reac >= t_min) warning "Temperature in reactor is too low!";
t_t_max:  test (t_reac <= t_max) warning "Temperature in reactor is too high!";

# pressure and temperature must be on the right side from the equilibrium curve
t_t_equilib:      test (t_reac > t_equilib) warning "Charging conditions not met, decrease p or
increase t in reactor!";

# No water in feed allowed
t_Water_feed:     test (feed.mass_Water < 1e-10) error "No water in feed allowed!";

# Using overall mass balance for testing correct balancing
t_mass:   test (drain.mass + drain_water.mass - feed.mass < 1e-6) warning "Overall mass
balance incorrect!";

# no change of substances
t_drain_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
```

```
                    error "Different material 1 at feed and drain!";
t_drain_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                    error "Different material 2 at feed and drain!";
t_drain_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
                error "Different oil at feed and drain!";
```

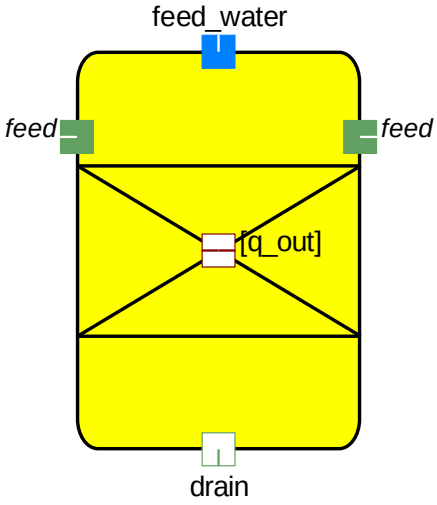
Variables

delta_p	pressure drop in the reactor
conv_TCM	conversion fraction of TCM
t_reac	temperature of the reaction
q_trans	transferred heat
q_reac	heat consumed by the reaction
q_preheat	heat required for a preheating of the reactor to a certain temperature
delta_H_reac	heat reaction of dehydration of TCM
hWater	water enthalpy at reaction temperature
molarmass_Material1	molar mass of TCM 1
molarmass_Material2	molar mass of TCM 2
H_298_15_Material1	enthalpy of formation of TCM1
H_298_15_Material2	enthalpy of formation of TCM2
ny_H2O_TCM	stochiometric coefficient of the extracted water
t_min	minimum allowed temperature at the reactor inlet Tstart
t_max	maximum allowed temperature in the reactor Tmax
t_equilib	reaction equilibrium temperature
delta_H_reac_0	standard enthalpy of reaction Spec from D2.5 chapter 3.2
delta_S_reac_0	standard entropy of reaction Spec from D2.5 chapter 3.2
ny_equilib	stochiometric coefficient of the equilibrium reactions

3.8. TCM_Reactor_Discharging

Purpose

Reactor for discharging step of thermochemical material (TCM). Low temperature side receiving heat from the reactor is optional. Different heat receiving working fluids (OR_ or heat transfer (T_) fluids) can be connected.



Connections

TCM_Stream: feed
TCM_Stream: drain
W_Stream: feed_water
q_cond_trans: q_out

3.8.1. TCM_Reactor_Discharging

Purpose

Design model for isothermal reactor to discharge the TCM.

Model equations

mass balances

f_Oil: feed.mass_Oil = drain.mass_Oil;
f_Material2: if (lambda > 1) then
drain.mass_Material2 = (1 - conv_TCM) * feed.mass_Material2;
else
drain.mass_Material2 = (1 - conv_TCM * lambda) * feed.mass_Material2;
f_Material1:
if (lambda > 1) then
drain.mass_Material1 = feed.mass_Material1 + conv_TCM * feed.mass_Material2 / molarmass_Material2 * molarmass_Material1;
else
drain.mass_Material1 = feed.mass_Material1 + lambda * conv_TCM * feed.mass_Material2 / molarmass_Material2 * molarmass_Material1;

stochiometric coefficient for the extracted water:
f_ny_H2O:

```
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  ny_H2O_TCM = 1.0;
else
#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  ny_H2O_TCM = 4.0;
else
#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  ny_H2O_TCM = 2.0;
else
#Potassiumcarbonate 1.5 hydrate:
if feed.TCM.ID1 <= 7.1 then
  ny_H2O_TCM = 1.5;
else
  ny_H2O_TCM = -1.0;

f_Water:  if (lambda > 1) then
                    if (feed_water.mass > ny_H2O_TCM * conv_TCM * feed.mass_Material2 /
molarmass_Material2 * 18.0153) then
                        drain.mass_Water = feed_water.mass - ny_H2O_TCM * conv_TCM *
feed.mass_Material2 / molarmass_Material2 * 18.0153;
                        else
                        drain.mass_Water = 0;
                    else
                    drain.mass_Water = feed_water.mass - ny_H2O_TCM * conv_TCM * lambda
* feed.mass_Material2 / molarmass_Material2 * 18.0153;

f_lambda:         lambda = (feed_water.mass / 18.0153) / (ny_H2O_TCM *
feed.mass_Material2 / molarmass_Material2);


f_molarmass_Material1: molarmass_Material1 = feed.TCM.TCM_M(feed.TCM.ID1);
f_molarmass_Material2: molarmass_Material2 = feed.TCM.TCM_M(feed.TCM.ID2);

# pressure drops
f_delta_p_TCM: feed.p - delta_p_TCM = drain.p;
f_delta_p_water: feed_water.p - delta_p_water = drain.p;

# pressure in the reactor
f_p2:       p = drain.p;

# energy balance

# calculation of reaction temperature t_reac (mixing temperature)

f_mix:              0.0 = (h_Material2_treac - feed.h_Material2) * feed.mass_Material2 +
                                (h_Material1_treac - feed.h_Material1) * feed.mass_Material1
+
                                (h_Oil_treac - feed.h_Oil) * feed.mass_Oil +
                                (feed_water.fhpt(p, t_reac, 1,0,0,0,0,0,0,0,0,0,0,0) -
feed_water.h) * feed_water.mass;
```

```
# enthalpies at t_reac
f_hOil_treac:       h_Oil_treac = feed.T_Composition.htf_h_t(t_reac) -
feed.T_Composition.htf_h_t(25);

f_hWater_treac: h_Water_treac = feed_water.fhpt(p, t_reac, 1,0,0,0,0,0,0,0,0,0,0,0) -
feed_water.fhpt(1.0, 25.0, 1,0,0,0,0,0,0,0,0,0,0,0);

f_h_Material1_t_reac: h_Material1_treac = feed.TCM.TCM_h_t(t_reac, feed.TCM.ID1) -
feed.TCM.TCM_h_t(25.0, feed.TCM.ID1);

f_h_Material2_t_reac: h_Material2_treac = feed.TCM.TCM_h_t(t_reac, feed.TCM.ID2) -
feed.TCM.TCM_h_t(25.0, feed.TCM.ID2);

f_H_298_15_Material1:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  H_298_15_Material1 = - 1093.99;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  H_298_15_Material1 = -2276.512;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  H_298_15_Material1 = - 1403.9;
else

#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  H_298_15_Material1 = -1612.930;
else
  H_298_15_Material1 = 0.0;
#--------------------------

f_H_298_15_Material2:
#Metaboric acid:
if feed.TCM.ID2 <= 2.1 then
  H_298_15_Material2 = - 802.78;
else

#Coppersulphate monohydrate:
if feed.TCM.ID2 <= 4.1 then
  H_298_15_Material2 = -1082.818;
else

#Calciumchloride:
if feed.TCM.ID2 <= 6.1 then
  H_298_15_Material2 = - 795.8;
else
```

```
#Potassiumcarbonate:
if feed.TCM.ID2 <= 8.1 then
  H_298_15_Material2 = -1151.499;
else
  H_298_15_Material2 = 0.0;

# minimum and maximum temperatures of the reactions that can be used as range
temperatures for the charging reactor
###
# These are preliminary values until more realistic data is made available!
###

f_t_min:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  t_min = 145.0;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  t_min = 80.0;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  t_min = 175.0;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  t_min = 135.0;
else
  t_min = 10.0;
#--------------------------

f_t_max:
#Metaboric acid:
if feed.TCM.ID2 <= 2.1 then
  t_max = 165.0;
else

#Coppersulphate monohydrate:
if feed.TCM.ID2 <= 4.1 then
  t_max = 130.0;
else

#Calciumchloride:
if feed.TCM.ID2 <= 6.1 then
  t_max = 210.0;
else

#Potassiumcarbonate:
if feed.TCM.ID2 <= 8.1 then
  t_max = 150.0;
```

```
else
  t_max = 0.0;


#-------------------------

# reaction of rehydration

f_q_reac:          q_reac = -((drain.mass_Material1 - feed.mass_Material1) * (h_Material1_treac
+ H_298_15_Material1 / molarmass_Material1 * 1000) -
                                    (feed_water.mass - drain.mass_Water) * (h_Water_treac -
285.83 / 18.0153 * 1000) -
                                    (feed.mass_Material2 - drain.mass_Material2) *
(h_Material2_treac + H_298_15_Material2 / molarmass_Material2 * 1000));

f_delta_H_reac:  delta_H_reac =          (h_Material2_treac * molarmass_Material2 / 1000 +
H_298_15_Material2 + h_Water_treac * 18.0153 / 1000 - 285.83 -
                                    (h_Material1_treac * molarmass_Material1 / 1000 +
H_298_15_Material1)) / molarmass_Material1 * 1000;

# isothermal discharging
# outlet temperature of the TCM from the reactor:
f_drain_t:         drain.t = t_reac;


# for discharging, q_reac minus any losses is transferred to the heat receiving side
f_q_trans:                 q_trans = q_reac - q_loss;

ifl ref(q_out) then # the cold side is connected

  # conductive heat transfer through the wall
  f_q_q_trans:     q_trans = q_out.q_trans;

  # transfer of prevailing temperatures at the heat transfer surface
  # the reactor is isothermal conditions
  f_t_hot_in:      q_out.t_hot_in = t_reac;
  f_t_hot_out:     q_out.t_hot_out = t_reac;
endifl

# equilibrium curves
# See Deliverable 2.4 eq. (2) and Table 2 for details. Data from experiments are used.

f_delta_H_reac_0:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  delta_H_reac_0 = 125.2;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  delta_H_reac_0 = 124.2;
else
```

```
#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  delta_H_reac_0 = 60.5;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  delta_H_reac_0 = 158.6;
else
  delta_H_reac_0 = 0.0;

f_delta_S_reac_0:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  delta_S_reac_0 = 298.1;
else

#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  delta_S_reac_0 = 328.0;
else

#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  delta_S_reac_0 = 135.5;
else
#Potassiumcarbonate 1.5H2O:
if feed.TCM.ID1 <= 7.1 then
  delta_S_reac_0 = 375.6;
else
  delta_S_reac_0 = 0.0;

# stochiometric coefficient for the equilibrium reactions (as they are different from
ny_H2O_TCM):
f_ny_equilib:
#Boric acid:
if feed.TCM.ID1 <= 1.1 then
  ny_equilib = 1.0;
else
#Coppersulphate pentahydrate:
if feed.TCM.ID1 <= 3.1 then
  ny_equilib = 2.0;          # reaction product is trihydrate
else
#Calciumchloride dihydrate:
if feed.TCM.ID1 <= 5.1 then
  ny_equilib = 1.0;          # reaction product is monohydrate
else
#Potassiumcarbonate 1.5 hydrate:
if feed.TCM.ID1 <= 7.1 then
  ny_equilib = 1.5;
else
  ny_equilib = -1.;
```

```
# Van't Hoff curve gives the equilibrium temperature-pressure relationship
# The reference pressure p+ is 1 bar and is cancelled out of the equation
# reactor pressure is drain TCM outlet pressure
f_t_equilib: ln(drain.p) = delta_S_reac_0/8.314/ny_equilib -
delta_H_reac_0*1000/8.314/ny_equilib/(t_equilib+273.15);

# test conditions

t_X1:          test (conv_TCM >= 0.0) error "Negative conversion rate!";
t_X2:          test (conv_TCM <= 1.0) error "Conversion rate higher than 1!";
t_q_reac: test (q_reac >= 0.0)        error "Heat of reaction is negative!";

# temperature range for discharging
t_t_min:   test (t_reac >= t_min) warning "Temperature in reactor is too low!";
t_t_max:  test (t_reac <= t_max) warning "Temperature in reactor is too high!";

# pressure and temperature must be on the left side from the equilibrium curve
t_t_equilib:      test (t_reac < t_equilib) warning "Discharging conditions not met, increase p or
decrease t in reactor!";

# substances must be the same
t_feed_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
                        error "Different material 1 at feed and drain!";
t_feed_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                        error "Different material 2 at feed and drain!";
t_feed_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
                  error "Different oil at feed and drain!";
```

Variables

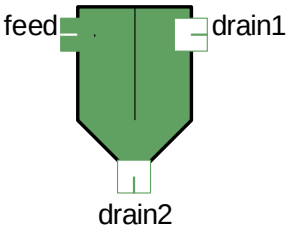
p	pressure
delta_p_TCM	pressure drop of TCM into the reactor
delta_p_water	pressure drop of water into the reactor
conv_TCM	conversion fraction of TCM Was variable X
t_reac	temperature of rehydration reaction
q_trans	transferred heat
q_reac	heat produced by the reaction (< 0)
delta_H_reac	heat reaction of rehydration of TCM
lambda	hydration factor
q_loss	heat loss to environment lost heat (<0)
h_Oil_treac	thermal oil enthalpy after mixing with water (at t_reac)
h_Water_treac	water enthalpy at treac
h_Material1_treac	enthalpy at reaction temperature of material1
h_Material2_treac	enthalpy at reaction temperature of material2
molarmass_Material1	molar mass of TCM 1
molarmass_Material2	molar mass of TCM 2
H_298_15_Material1	enthalpy of formation of TCM1
H_298_15_Material2	enthalpy of formation of TCM2
ny_H2O_TCM	stochiometric coefficient of the extracted water
t_min	minimum allowed temperature at the reactor inlet

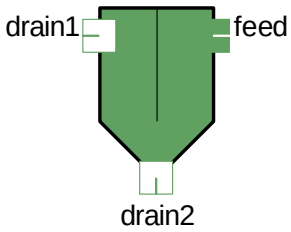
	Tstart
t_max	maximum allowed temperature in the reactor
	Tmax
t_equilib	reaction equilibrium temperature
delta_H_reac_0	standard enthalpy of reaction
	Spec from D2.5 chapter 3.2
delta_S_reac_0	standard entropy of reaction
	Spec from D2.5 chapter 3.2
ny_equilib	stochiometric coefficient of the equilibrium reactions

3.9. TCM_Separator

Purpose

Separator for an incoming thermochemical material stream.





Connections

TCM_Stream: feed
TCM_Stream: drain1
TCM_Stream: drain2

3.9.1. TCM_Separator

Purpose

allows solids-liquid separation of an incoming TCM stream

Model equations

```
# mass balances

# overall
f_mass:                 feed.mass = drain1.mass + drain2.mass;

# one of the individual mass balances is redundant and hence omitted
#f_mass_Oil:            feed.mass_Oil = drain1.mass_Oil + drain2.mass_Oil;
f_mass_Material1:       feed.mass_Material1 = drain1.mass_Material1 +
drain2.mass_Material1;
f_mass_Material2:       feed.mass_Material2 = drain2.mass_Material2 +
drain1.mass_Material2;
f_mass_Water:           feed.mass_Water = drain1.mass_Water + drain2.mass_Water;

# separation, individual split ratios for the different substances
f_phi_Oil:              drain1.mass_Oil = phi_Oil * feed.mass_Oil;
f_phi_Material1:  drain1.mass_Material1 = phi_Material1 * feed.mass_Material1;
f_phi_Material2:  drain1.mass_Material2 = phi_Material2 * feed.mass_Material2;
f_phi_Water:            drain1.mass_Water = phi_Water * feed.mass_Water;

# process conditions do not change, no pressure drops, no thermal losses
```

```
# pressures constant
f_p1:               feed.p = drain1.p;
f_p2:               feed.p = drain2.p;

# temperatures constant
f_t1:               feed.t = drain1.t;
f_t2:               feed.t = drain2.t;

# no change of substances
t_drain1_material1: test (feed.TCM.ID1 == drain1.TCM.ID1)
                          error "different material 1 at feed and drain1!";
t_drain1_material2: test (feed.TCM.ID2 == drain1.TCM.ID2)
                          error "different material 2 at feed and drain1!";
t_drain1_oil: test (feed.T_Composition.FluidID == drain1.T_Composition.FluidID)
                    error "different oil at feed and drain1!";
t_drain2_material1: test (feed.TCM.ID1 == drain2.TCM.ID1)
                          error "different material 1 at feed and drain2!";
t_drain2_material2: test (feed.TCM.ID2 == drain2.TCM.ID2)
                          error "different material 2 at feed and drain2!";
t_drain2_oil: test (feed.T_Composition.FluidID == drain2.T_Composition.FluidID)
                    error "different oil at feed and drain2!";
```

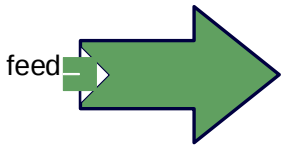
Variables

phi_Oil	splitting ratio of extracted oil going to drain1
phi_Material1	splitting ratio of extracted Material1 going to drain1
phi_Material2	splitting ratio of extracted Material2 going to drain1
phi_Water	splitting ratio of water going to drain1

3.10. TCM_Sink

Purpose

Sink for a thermochemical material stream.



Connections

TCM_Stream: feed

3.10.1. TCM_Sink

Purpose

sink modelling the release of a stream

Model equations

f_mass: mass = feed.mass;
f_p: p = feed.p;
f_t: t = feed.t;

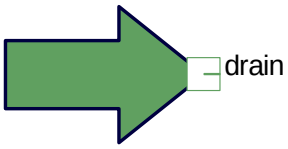
Variables

mass	mass flow
p	outlet pressure
t	outlet temperature

3.11. TCM_Source

Purpose

Source for a thermochemical material stream.



Connections

TCM_Stream: drain

3.11.1. TCM_Source

Purpose

source modeling the inlet of a stream

Model equations

f_mass: mass = drain.mass;
f_p: p = drain.p;
f_t: t = drain.t;

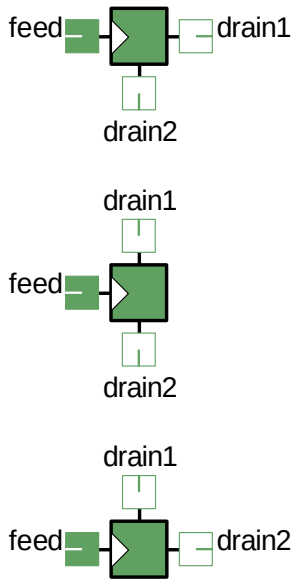
Variables

mass	mass flow
p	pressure
t	temperature

3.12. TCM_Splitter

Purpose

Splitter for an incoming thermochemical material stream.



Connections

TCM_Stream: feed
TCM_Stream: drain1
TCM_Stream: drain2

3.12.1. TCM_Splitter

Purpose

splits an incoming TCM stream into two outgoing TCM streams with equal thermodynamic properties

Model equations

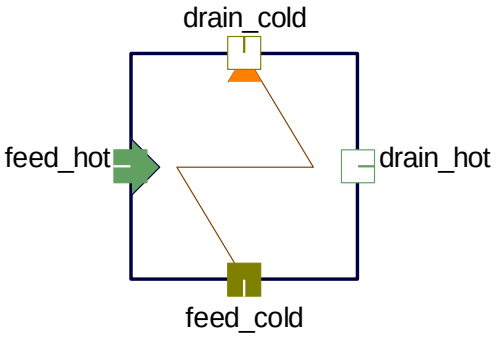
```
# mass balances

# overall
f_mass:                 feed.mass = drain1.mass + drain2.mass;

# one of the individual mass balances is redundant and hence omitted
#f_mass_Oil:            feed.mass_Oil = drain1.mass_Oil + drain2.mass_Oil;
f_mass_Material1:       feed.mass_Material1 = drain1.mass_Material1 +
drain2.mass_Material1;
f_mass_Material2:       feed.mass_Material2 = drain2.mass_Material2 +
drain1.mass_Material2;
f_Mass_Water:           feed.mass_Water = drain1.mass_Water + drain2.mass_Water;

# composition (material fractions) and process conditions do not change

# one of the fractions is redundant and hence omitted
#f_frac_Oil:        feed.w_Material1 = drain1.w_Material1:
f_frac_Material1: feed.w_Material1 = drain1.w_Material1;
```

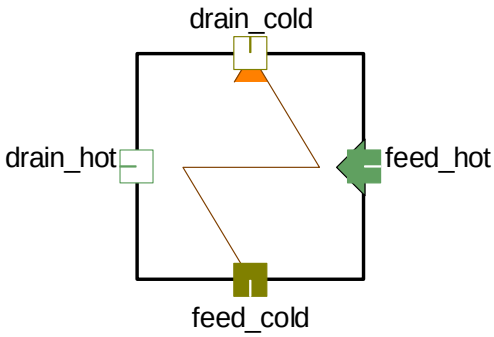
```
f_frac_Material2: feed.w_Material2 = drain1.w_Material2;
f_frac_Water:           feed.w_Water = drain1.w_Water;

# pressures identical
f_p1:           feed.p = drain1.p;
f_p2:           feed.p = drain2.p;

# temperatures identical
f_t1:           feed.t = drain1.t;
f_t2:           feed.t = drain2.t;

# tests

# no change of substances
t_drain1_material1: test (feed.TCM.ID1 == drain1.TCM.ID1)
                        error "different material 1 at feed and drain1!";
t_drain1_material2: test (feed.TCM.ID2 == drain1.TCM.ID2)
                        error "different material 2 at feed and drain1!";
t_drain1_oil: test (feed.T_Composition.FluidID == drain1.T_Composition.FluidID)
                    error "different oil at feed and drain1!";
t_drain2_material1: test (feed.TCM.ID1 == drain2.TCM.ID1)
                        error "different material 1 at feed and drain2!";
t_drain2_material2: test (feed.TCM.ID2 == drain2.TCM.ID2)
                        error "different material 2 at feed and drain2!";
t_drain2_oil: test (feed.T_Composition.FluidID == drain2.T_Composition.FluidID)
                    error "different oil at feed and drain2!";
```

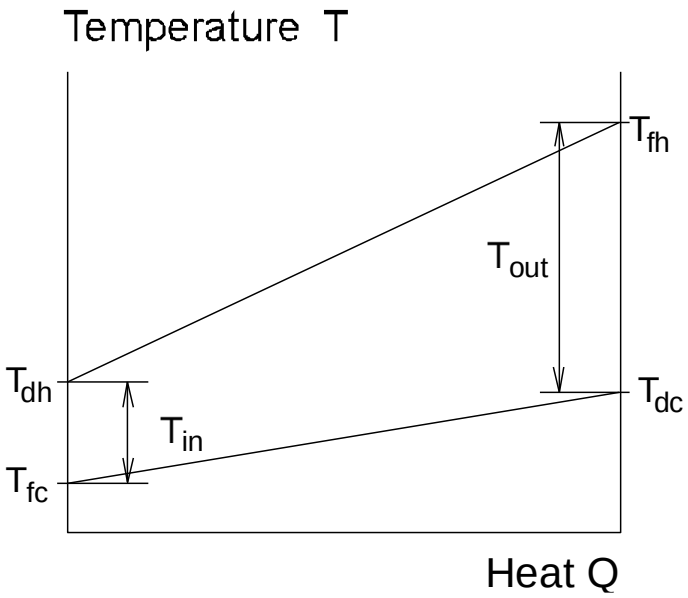
3.13. TCM_T_Htex

Purpose

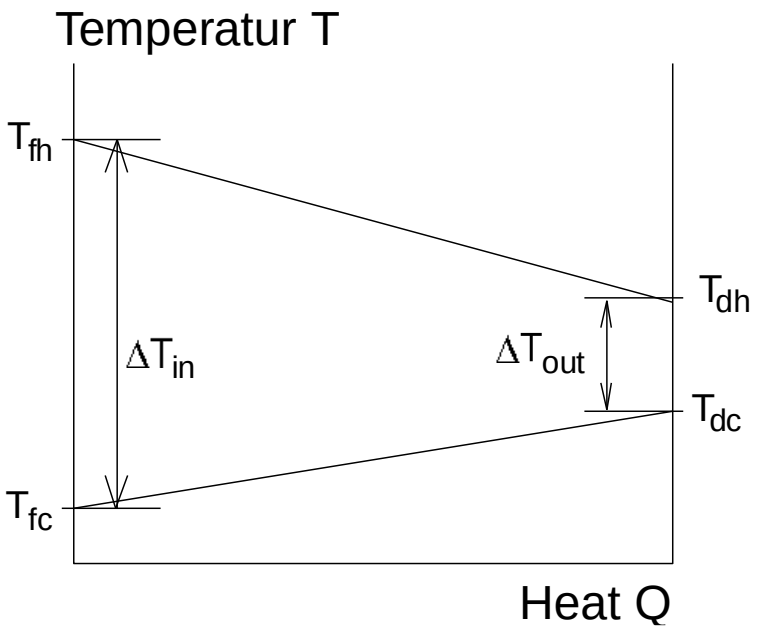
Heat exchanger for transfer from TCM fluid on the hot side to thermofluid on the cold side. Cocurrent or counter current flow.







QT-diagram, counter current



QT-diagram, cocurrent

Connections

TCM_Stream: drain_hot
TCM_Stream: feed_hot
T_Stream: feed_cold
T_Stream: drain_cold

3.13.1. TCM_T_Htex

Purpose

design model

Model equations

```
# mass balances

# hot side total and individual mass balances (one is redundant)
f_mass_hot:                 feed_hot.mass = drain_hot.mass;
#f_mass_Oil_hot:        feed_hot.mass_Oil = drain_hot.mass_Oil;
f_mass_Material1_hot:       feed_hot.mass_Material1 = drain_hot.mass_Material1;
f_mass_Material2_hot:       feed_hot.mass_Material2 = drain_hot.mass_Material2;
f_mass_Water_hot:           feed_hot.mass_Water = drain_hot.mass_Water;

# cold side mass balance
f_mass_cold:                feed_cold.mass = drain_cold.mass;

# pressure drops

f_delta_p_hot:          feed_hot.p - delta_p_hot = drain_hot.p;
f_delta_p_cold:         feed_cold.p - delta_p_cold = drain_cold.p;

# energy balance

f_e_hot:  feed_hot.mass * (feed_hot.h-drain_hot.h) * (1.0 - heat_loss/100) = q_trans;
f_e_cold: feed_cold.mass * (drain_cold.h - feed_cold.h) = q_trans;
```

```
# temperature differences
# They are differently defined for co- and counter current heat exchangers.

ifl Type == cocurrent then
  f_dt_in_co:       feed_hot.t - dt_in = feed_cold.t;
  f_dt_out_co:      drain_hot.t - dt_out = drain_cold.t;
endifl

ifl Type == counter_current then
  f_dt_in_counter:          drain_hot.t - dt_in = feed_cold.t;
  f_dt_out_counter:         feed_hot.t - dt_out = drain_cold.t;
endifl

# Mean temperature difference is the logarithmic temperature difference.
# For nearly equal inlet and outlet temperature differences, it is approximated
# by the arithmetic mean (which is the asymptotic approximation).
f_MTD: if abs(dt_in/dt_out) >=1.2 || abs(dt_out/dt_in) >=1.2 then
            MTD * ln(dt_in/dt_out) = (dt_in-dt_out);
   else
            MTD = (dt_in+dt_out)/2.0;

f_htc_area:         q_trans = htc_area * MTD;

# tests
t_dt_in:    test(dt_in > 0.0) error "dt_in <= 0.0";
t_dt_out: test(dt_out > 0.0)          error "dt_out <= 0.0";
t_q_trans:          test(q_trans > 0.0)         error "q_trans <= 0.0";

# no change of substances
t_drain_hot_material1: test (feed_hot.TCM.ID1 == drain_hot.TCM.ID1)
                                    error "Different material 1 at feed_hot and drain_hot!";
t_drain_hot_material2: test (feed_hot.TCM.ID2 == drain_hot.TCM.ID2)
                                    error "Different material 2 at feed_hot and drain_hot!";
t_drain_hot_oil: test (feed_hot.T_Composition.FluidID == drain_hot.T_Composition.FluidID)
                                error "Different oil at feed_hot and drain_hot!";

# feed and drain thermooil must use the same fluid
t_conn_cold:        test (feed_cold.T_Composition.FluidID == drain_cold.T_Composition.FluidID)
                    error "different thermofluid at feed and drain cold temperature side!";
```

Parameters

delta_p_hot	pressure drop of the high temperature side
delta_p_cold	pressure drop of the low temperature side
heat_loss	percentage of heat lost to the surroundings

Variables

dt_in	temperature difference at the inlet of the low temperature side
dt_out	temperature difference at the outlet of the low temperature side
htc_area	heat transfer coefficient x transfer area
MTD	mean temperature difference
q_trans	transferred heat

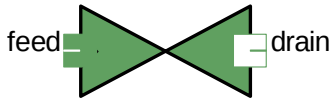
Switches

Type (cocurrent, counter_current)	defines the type of heat exchanger

3.14. TCM_Valve

Purpose

Valve for thermochemical material stream.



Connections

TCM_Stream: feed
TCM_Stream: drain

3.14.1. TCM_Valve

Purpose

adiabatic expansion in a valve

Model equations

```
# mass balances

# overall
f_mass:   feed.mass = drain.mass;

f_mass_Material1:       feed.mass_Material1 = drain.mass_Material1;
f_mass_Material2:       feed.mass_Material2 = drain.mass_Material2;
f_mass_Water:           feed.mass_Water = drain.mass_Water;
# f_mass_Oil:           drain.mass_Oil = drain.mass_Oil;

# pressure drop
f_delta_p_rel:    (1.0-pressure_drop) * feed.p = drain.p;
f_delta_p_abs:    feed.p - delta_p = drain.p;

# adiabatic expansion, enthalpy is constant
f_h:        feed.h = drain.h;

# test conditions
t_delta_p:          test (delta_p>=0.0) warning "delta_p < 0.0";

# no change of substances
t_conn_material1: test (feed.TCM.ID1 == drain.TCM.ID1)
                        error "different material 1 at feed and drain!";
t_conn_material2: test (feed.TCM.ID2 == drain.TCM.ID2)
                        error "different material 2 at feed and drain!";
t_conn_oil: test (feed.T_Composition.FluidID == drain.T_Composition.FluidID)
                    error "different oil at feed and drain!";
```

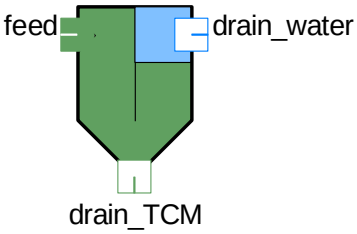
Variables

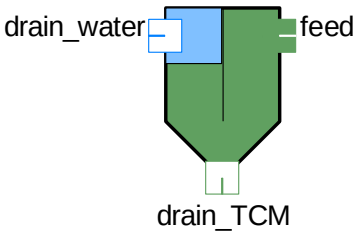
pressure_drop	relative pressure drop
delta_p	absolute pressure drop

3.15. TCM_W_Separator

Purpose

Separator for pure water from an incoming thermochemical material stream.





Connections

TCM_Stream: feed
TCM_Stream: drain_TCM
W_Stream: drain_water

3.15.1. TCM_W_Separator

Purpose

allows separation of water from an incoming TCM stream

Model equations

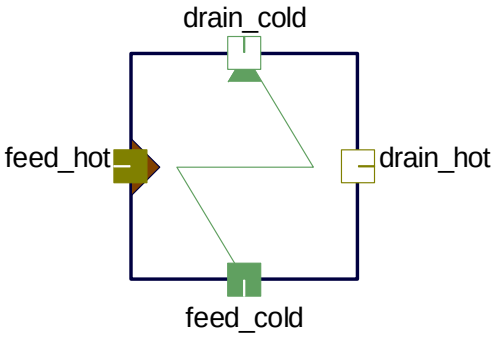
```
# mass balances

# overall
f_mass:                 feed.mass = drain_water.mass + drain_TCM.mass;

# one of the individual mass balances is redundant and hence omitted
#f_mass_Oil:            feed.mass_Oil = drain_TCM.mass_Oil;
f_mass_Material1:       feed.mass_Material1 = drain_TCM.mass_Material1;
f_mass_Material2:       feed.mass_Material2 = drain_TCM.mass_Material2;

f_mass_Water:           feed.mass_Water = drain_water.mass;

# water is 100% separated
f_sep_Water:            drain_TCM.mass_Water = 0.0;

# process conditions do not change, no pressure drops, no thermal losses

# pressures constant
f_p1:           feed.p = drain_water.p;
f_p2:           feed.p = drain_TCM.p;
```

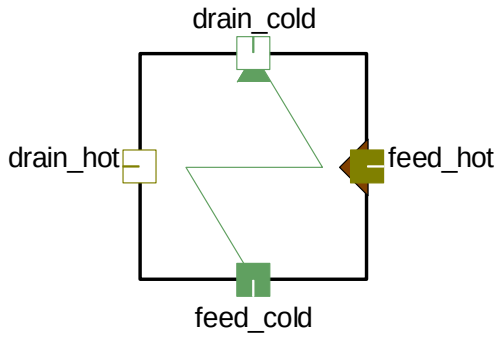
```
# temperatures constant
f_t1:                feed.t = drain_water.t;
f_t2:                feed.t = drain_TCM.t;

# no change of substances
t_drain_TCM_material1: test (feed.TCM.ID1 == drain_TCM.TCM.ID1)
                           error "different material 1 at feed and drain_TCM!";
t_drain_TCM_material2: test (feed.TCM.ID2 == drain_TCM.TCM.ID2)
                           error "different material 2 at feed and drain_TCM!";
t_drain_TCM_oil: test (feed.T_Composition.FluidID == drain_TCM.T_Composition.FluidID)
                   error "different oil at feed and drain_TCM!";
```

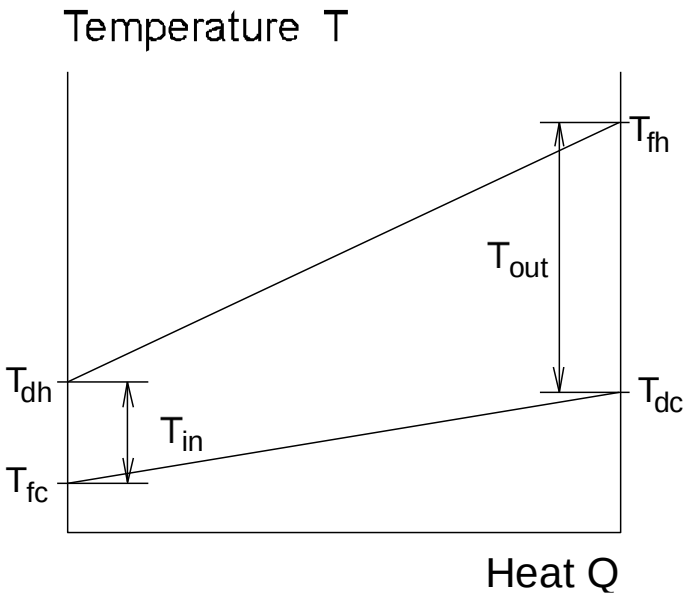
3.16. T_TCM_Htex

Purpose

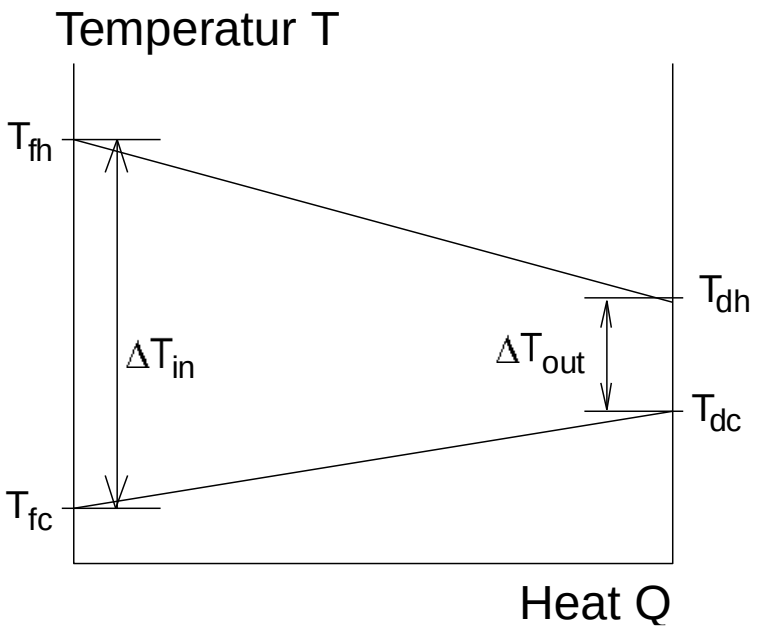
Heat exchanger for transfer from thermofluid on the hot side to TCM fluid on the cold side. Cocurrent or counter current flow.







QT-diagram, counter current



QT-diagram, cocurrent

Connections

TCM_Stream: drain_cold
TCM_Stream: feed_cold
T_Stream: feed_hot
T_Stream: drain_hot

3.16.1. T_TCM_Htex

Purpose

design model

Model equations

```
# mass balances

# hot side mass balance
f_mass_hot:                 feed_hot.mass = drain_hot.mass;

# cold side total and individual mass balances (one is redundant)
f_mass_cold:                feed_cold.mass = drain_cold.mass;
#f_mass_Oil_cold:           feed_cold.mass_Oil = drain_cold.mass_Oil;
f_mass_Material1_cold:  feed_cold.mass_Material1 = drain_cold.mass_Material1;
f_mass_Material2_cold:  feed_cold.mass_Material2 = drain_cold.mass_Material2;
f_mass_Water_cold:          feed_cold.mass_Water = drain_cold.mass_Water;

# pressure drops

f_delta_p_hot:          feed_hot.p - delta_p_hot = drain_hot.p;
f_delta_p_cold:         feed_cold.p - delta_p_cold = drain_cold.p;

# energy balance

f_e_hot:  feed_hot.mass * (feed_hot.h-drain_hot.h) * (1.0 - heat_loss/100) = q_trans;
f_e_cold: feed_cold.mass * (drain_cold.h - feed_cold.h) = q_trans;
```

```
# temperature differences
# They are differently defined for co and counter current heat exchangers.

ifl Type == cocurrent then
  f_dt_in_co:       feed_hot.t - dt_in = feed_cold.t;
  f_dt_out_co:      drain_hot.t - dt_out = drain_cold.t;
endifl

ifl Type == counter_current then
  f_dt_in_counter:          drain_hot.t - dt_in = feed_cold.t;
  f_dt_out_counter:         feed_hot.t - dt_out = drain_cold.t;
endifl

# Mean temperature difference is the logarithmic temperature difference.
# For nearly equal inlet and outlet temperature differences, it is approximated
# by the arithmetic mean (which is the asymptotic approximation).
f_MTD: if abs(dt_in/dt_out) >=1.2 || abs(dt_out/dt_in) >=1.2 then
            MTD * ln(dt_in/dt_out) = (dt_in-dt_out);
   else
            MTD = (dt_in+dt_out)/2.0;

f_htc_area:         q_trans = htc_area * MTD;

# tests
t_dt_in:    test(dt_in > 0.0) error "dt_in <= 0.0";
t_dt_out: test(dt_out > 0.0)          error "dt_out <= 0.0";
t_q_trans:          test(q_trans > 0.0)         error "q_trans <= 0.0";

# no change of substances
# feed and drain thermooil must use the same fluid
t_conn_hot:         test (feed_hot.T_Composition.FluidID == drain_hot.T_Composition.FluidID)
                    error "different thermofluid at feed and drain hot temperature side!";

t_drain_cold_material1: test (feed_cold.TCM.ID1 == drain_cold.TCM.ID1)
                                    error "Different material 1 at feed_cold and drain_cold!";
t_drain_cold_material2: test (feed_cold.TCM.ID2 == drain_cold.TCM.ID2)
                                    error "Different material 2 at feed_cold and drain_cold!";
t_drain_cold_oil: test (feed_cold.T_Composition.FluidID == drain_cold.T_Composition.FluidID)
                            error "Different oil at feed_cold and drain_cold!";
```

Parameters

delta_p_hot	pressure drop of the high temperature side
delta_p_cold	pressure drop of the low temperature side
heat_loss	percentage of heat lost to the surroundings

Variables

dt_in	temperature difference at the inlet of the low temperature side
dt_out	temperature difference at the outlet of the low temperature side
htc_area	heat transfer coefficient x transfer area
MTD	mean temperature difference
q_trans	transferred heat

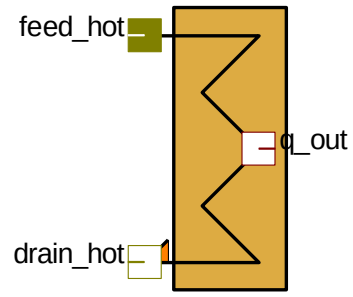
Switches

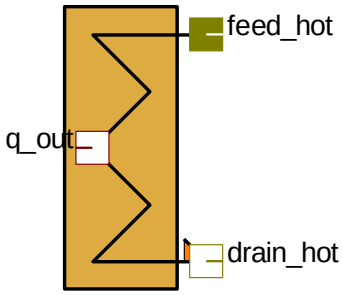
Type (cocurrent, counter_current)	defines the type of heat exchanger

3.17. T_wall_htex

Purpose

Heat exchanger with wall transferring heat from thermooil (T) to another side.





Connections

q_cond_trans: q_out
T_Stream: feed_hot
T_Stream: drain_hot

3.17.1. T_wall_htex

Purpose

design model

Model equations

```
# heat transfer from fluid inside the tubes through the wall

# mass transfer
f_mass:   feed_hot.mass = drain_hot.mass;

# pressure drop through the wall tubes
f_delta_p:          feed_hot.p - delta_p = drain_hot.p;

# hot fluid is cooled down
f_energy:feed_hot.mass*feed_hot.h - q_trans = drain_hot.mass*drain_hot.h;

# temperatue change
```

```
f_delta_t: feed_hot.t - delta_t = drain_hot.t;

# conductive heat transfer through the wall
f_q_q_trans:       q_trans = q_out.q_trans;

# transfer of prevailing temperatures
ifl Type == cocurrent then
  f_t_hot_in_co:   q_out.t_hot_in = feed_hot.t;
  f_t_hot_out_co: q_out.t_hot_out = drain_hot.t;
endifl

ifl Type == counter_current then
  f_t_hot_in_counter:      q_out.t_hot_in = drain_hot.t;
  f_t_hot_out_counter:     q_out.t_hot_out = feed_hot.t;
endifl

# test  conditions
t_q_trans:          test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:                  test (delta_p >= 0.0) warning "delta_p negative";

# feed and drain thermooil must use the same fluid
t_conn_hot: test (feed_hot.T_Composition.FluidID == drain_hot.T_Composition.FluidID)
                  error "different thermofluid at feed and drain!";
```

Parameters

delta_p	absolute pressure drop

Variables

delta_t	temperature difference between feed and drain mass flow
q_trans	transferred heat

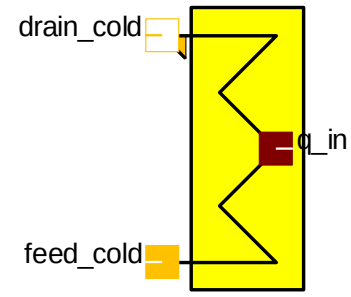
Switches

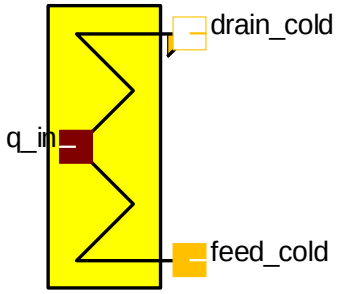
Type (cocurrent, counter_current)	defines the type of heat exchanger

3.18. wall_OR_htex

Purpose

Heat exchanger with wall transferring heat from wall to organic fluid (OR).





Connections

OR_Stream: feed_cold
OR_Stream: drain_cold
q_cond_trans: q_in

3.18.1. wall_OR_htex

Purpose

design model

Model equations

```
# heat transfer from wall to fluid inside the tubes

# mass transfer
f_mass:   feed_cold.mass = drain_cold.mass;

# pressure drop through the wall tubes
f_delta_p:          feed_cold.p - delta_p = drain_cold.p;

# cold fluid is heat up
f_energy: feed_cold.mass*feed_cold.h + q_trans = drain_cold.mass*drain_cold.h;

# temperatue change
f_delta_t: feed_cold.t + delta_t = drain_cold.t;
```

```
# conductive heat transfer through the wall
f_q_q_trans:        q_trans = q_in.q_trans;

# transfer of prevailing temperatures
# for the cold side cocurrent/counter current does not make a difference
f_t_cold_in:        q_in.t_cold_in = feed_cold.t;
f_t_cold_out:       q_in.t_cold_out = drain_cold.t;

# additional calculations and data for operating points of the working fluid with phase change
(inside the tubes)
# heat is flowing in, working fluid is heated and may evaporate

p_crit:     p_crit = feed_cold.Composition.o_pcrit();
s_crit:     s_crit = feed_cold.Composition.o_scrit();

# mean evaporation pressure
p_evap:   p_evap = feed_cold.p - delta_p;

# discriminating subcritical and supercritical boiler conditions
f_x_feed: if (p_evap <= p_crit) then
                    feed_cold.h - feed_cold.Composition.o_h_px( p_evap, 0) - x_feed*
(feed_cold.Composition.o_h_px( p_evap, 1.0) -  feed_cold.Composition.o_h_px( p_evap, 0)) =
0.0;
          else
                    x_feed = 999.0;

f_x_drain:          if (p_evap <= p_crit) then
                    drain_cold.h - drain_cold.Composition.o_h_px( p_evap, 0) - x_drain*
(drain_cold.Composition.o_h_px( p_evap, 1.0) -  drain_cold.Composition.o_h_px( p_evap, 0)) =
0.0;
          else
                    x_drain = -999;

# start of evaporation resp.. intermediate point I

f_s_I_aux: s_I_aux = feed_cold.s + (drain_cold.s - feed_cold.s) * 4.0/9.0;

f_s_I:      if (x_feed <= 0.0) then
                    s_I = feed_cold.Composition.o_s_px(p_evap, 0.0);
          else
                    if(s_I_aux <= s_crit) then
                              s_I = s_crit;
                    else
                              s_I = feed_cold.s + (drain_cold.s - feed_cold.s) * 4.0/9.0;

f_p_I:      if (p_evap <= p_crit && x_feed <= 1.0) then
                    p_I = p_evap;
          else
                    p_I = feed_cold.p - delta_p*8/9;

f_t_I:      t_I = feed_cold.Composition.o_t_ps(p_I, s_I);

f_h_I:      s_I = drain_cold.Composition.o_s_ph(p_I, h_I);
```

```
# end of evaporation resp. intermediate point II
f_s_II_aux: s_II_aux = feed_cold.s + (drain_cold.s - feed_cold.s) * 5.0/9.0;

f_s_II:     if (x_drain >= 1) then
                    if (x_feed <= 1.0) then
                            s_II = drain_cold.Composition.o_s_px(p_evap, 1.0);
                    else
                            if(s_II_aux < s_crit) then
                                    s_II = s_crit;
                            else
                                    s_II = feed_cold.s + (drain_cold.s - feed_cold.s) * 5/9;
            else
                    if (x_feed <= 1.0) then
                            s_II = feed_cold.Composition.o_s_px(p_II, 0.0);
                    else
                            if(s_II_aux < s_crit) then
                                    s_II = s_crit;
                            else
                                    s_II = feed_cold.s + (drain_cold.s - feed_cold.s) * 5/9;

f_p_II:     if (p_evap <= p_crit && x_feed <= 1.0) then
                    p_II = p_evap;
            else
                    p_II = drain_cold.p + delta_p*8.0/9.0;

f_t_II:     t_II = drain_cold.Composition.o_t_ps(p_II, s_II);

f_h_II:     s_II = drain_cold.Composition.o_s_ph(p_II, h_II);

# calculating intermediate data points to obtain a temperature profile  (6 in total for each
property)

# subcritical design gives additional points in liquid and vapour region (3 each)
# supercritical design evenly spreads entropy s between inlet and outlet

f_s_1:      s_1 = feed_cold.s + (s_l-feed_cold.s)*1/4;
f_p_1:      p_1 = feed_cold.p - (feed_cold.p-p_l)*1/4;
f_t_1:      t_1 = feed_cold.Composition.o_t_ps(p_1, s_1);

f_s_2:      s_2 = feed_cold.s + (s_l-feed_cold.s)*2/4;
f_p_2:      p_2 = feed_cold.p - (feed_cold.p-p_l)*2/4;
f_t_2:      t_2 = feed_cold.Composition.o_t_ps(p_2, s_2);

f_s_3:      s_3 = feed_cold.s + (s_l-feed_cold.s)*3/4;
f_p_3:      p_3 = feed_cold.p - (feed_cold.p-p_l)*3/4;
f_t_3:      t_3 = feed_cold.Composition.o_t_ps(p_3, s_3);

f_s_4:      s_4 = s_II + (drain_cold.s-s_II)*1/4;
f_p_4:      p_4 = p_II - (p_II-drain_cold.p)*1/4;
f_t_4:      t_4 = feed_cold.Composition.o_t_ps(p_4, s_4);
```

```
f_s_5:     s_5 = s_II + (drain_cold.s-s_II)*2/4;
f_p_5:     p_5 = p_II - (p_II-drain_cold.p)*2/4;
f_t_5:     t_5 = feed_cold.Composition.o_t_ps(p_5, s_5);

f_s_6:     s_6 = s_II + (drain_cold.s-s_II)*3/4;
f_p_6:     p_6 = p_II - (p_II-drain_cold.p)*3/4;
f_t_6:     t_6 = feed_cold.Composition.o_t_ps(p_6, s_6);

# test  conditions
t_q_trans:          test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:                  test (delta_p >= 0.0) warning "delta_p negative";

# feed and drain terminal must reference the same composition
ifl ref(feed_cold.Composition) != ref(drain_cold.Composition) then
  t_Comp:          test(1!=1) error "Composition must be the same at feed and drain";
endifl
```

Parameters

delta_p	absolute pressure drop

Variables

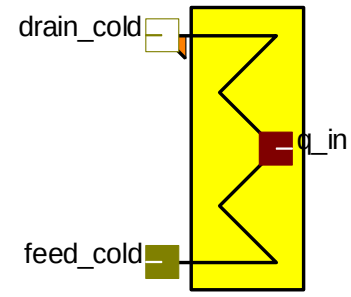
delta_t	temperature difference between feed and drain mass flow
q_trans	transferred heat
p_evap	--- profile calculation: mean evaporation pressure
p_I	--- profile calculation: intermediate presssure
t_I	--- profile calculation: intermediate temperature (sat liq resp. inlet)
h_I	intermediate enthalpy (sat liq resp. inlet)
s_I	--- profile calculation: intermediate entropy (sat liq resp. inlet)
s_I_aux	--- profile calculation: auxilary variable (entropy)
p_II	--- profile calculation: intermediate pressure (sat vap resp. outlet)
t_II	--- profile calculation: intermediate temperature (sat vap resp. outlet)
h_II	intermediate enthalpy (sat vap resp. outlet)
s_II	--- profile calculation: intermediate entropy (sat vap resp. outlet)
s_II_aux	--- profile calculation: auxilary variable (entropy)
x_feed	--- profile calculation: steam quality at feed
x_drain	--- profile calculation: steam quality at drain
p_1	--- profile calculation: intermediate pressure #1
t_1	--- profile calculation: intermediate temperatures #1
s_1	--- profile calculation: intermediate entropy #1
p_2	--- profile calculation: intermediate pressure #2
t_2	--- profile calculation: intermediate temperature #2
s_2	--- profile calculation: intermediate entropy #2
p_3	--- profile calculation: intermediate pressure #3
t_3	--- profile calculation: intermediate temperature #3
s_3	--- profile calculation: intermediate entropy #3
p_4	--- profile calculation: intermediate pressure #4
t_4	--- profile calculation: intermediate temperature #4
s_4	--- profile calculation: intermediate entropy #4
p_5	--- profile calculation: intermediate pressure #5
t_5	--- profile calculation: intermediate temperature #5
s_5	--- profile calculation: intermediate entropy #5

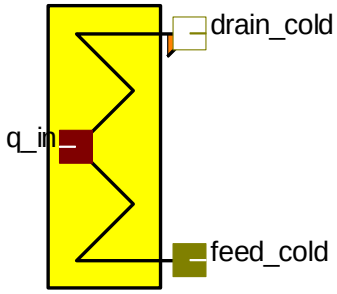
p_6	--- profile calculation: intermediate pressure #6
t_6	--- profile calculation: intermediate temperature #6
s_6	--- profile calculation: intermediate entropy #6
p_crit	--- profile calculation: critical pressure of the working fluid
s_crit	--- profile calculation: entropy at critical point

3.19. wall_T_htex

Purpose

Heat exchanger with wall transferring heat from wall to thermooil (T).





Connections

q_cond_trans: q_in
T_Stream: feed_cold
T_Stream: drain_cold

3.19.1. wall_T_htex

Purpose

design model

Model equations

```
# heat transfer from wall to fluid inside the tubes

# mass transfer
f_mass:   feed_cold.mass = drain_cold.mass;

# pressure drop through the wall tubes
f_delta_p:          feed_cold.p - delta_p = drain_cold.p;

# cold fluid is heat up
f_energy: feed_cold.mass*feed_cold.h + q_trans = drain_cold.mass*drain_cold.h;

# temperatue change
f_delta_t: feed_cold.t + delta_t = drain_cold.t;
```

```
# conductive heat transfer through the wall
f_q_q_trans:      q_trans = q_in.q_trans;

# transfer of prevailing temperatures
# for the cold side cocurrent/counter current does not make a difference
f_t_cold_in:      q_in.t_cold_in = feed_cold.t;
f_t_cold_out:     q_in.t_cold_out = drain_cold.t;

# test  conditions
t_q_trans:        test (q_trans >= 0.0) warning "q_trans negative";
t_delta_p:              test (delta_p >= 0.0) warning "delta_p negative";

# feed and drain thermooil must use the same fluid
t_conn_cold:      test (feed_cold.T_Composition.FluidID == drain_cold.T_Composition.FluidID)
                  error "different thermofluid at feed and drain!";
```

Parameters

delta_p	absolute pressure drop

Variables

delta_t	temperature difference between feed and drain mass flow
q_trans	transferred heat

3.20. OR_Stream

Purpose

Transfer of mass. The transferred fluid is represented by an OR_Composition object.

Global objects

OR_Composition: Composition defines the ORC working fluid in use

3.20.1. OR_Stream

Purpose

Transfer of mass. The transferred fluid is represented by an OR_Composition object.

Model equations

```
ft: if blocksize() == 1.0 && isconverged(p) && isconverged(t) then
        h = Composition.o_h_pt(p, t);
    else
        t = Composition.o_t_ph(p, h);

fs:     s = Composition.o_s_ph(p ,h);

fv:     v = Composition.o_v_ph(p, h);


t1:      test (mass >=0.0) warning "mass flow negative";
```

Variables

p	pressure
t	temperature
h	enthalpy
s	entropy
v	specific volume
mass	mass flow

3.21. TCM_Stream

Purpose

thermochemical storage material (TCM) stream

Global objects

TCM: TCM Thermochemical material working pair
T_Composition: T_Composition Heat transfer fluid in use

3.21.1. TCM_Stream

Purpose

thermochemical storage material (TCM) stream

Model equations

```
# thermochemical storage material stream

# It is a mixture of the working pair material suspended in oil plus water

# mass fraction balances
```

```
f_mass_frac:       w_Material1 + w_Material2 + w_Oil + w_Water = 1.0;

f_mass_Material1:          mass_Material1 = mass * w_Material1;
f_mass_Material2:          mass_Material2 = mass * w_Material2;
f_mass_Oil:                mass_Oil = mass * w_Oil;
f_mass_Water:              mass_Water = mass * w_Water;

# various fractions for additional information

#fSolid_fraction:  Solid_fraction = Material1 + Material2 ;
f_w_Solids:                w_Solids = w_Material1 + w_Material2;

#fX_Material1_Oil:         X_Material1_Oil = Material1 / Oil;
#fX_Material2_Oil:         X_Material2_Oil = Material2 / Oil;
f_X_Material1_Oil:         X_Material1_Oil * w_Oil = w_Material1;
f_X_Material2_Oil:         X_Material2_Oil * w_Oil = w_Material2;

# Global thermodynamic properties

#fh:                h = h_Oil * Oil + h_Material1 * Material1 + h_Material2 * Material2 + water *
h_Water;
#fv:                v = v_Oil * Oil + v_Material2 * Material2 + v_Material1 * Material1 + water *
v_Water;

f_h:               h = h_Oil * w_Oil + h_Material1 * w_Material1 + h_Material2 * w_Material2 +
h_Water * w_Water;
f_v:               v = v_Oil * w_Oil + v_Material1 * w_Material1 + v_Material2 * w_Material2 +
v_Water * w_Water;

# individual specific volumes
f_v_Oil:           v_Oil = T_Composition.htf_v_t(t);
f_v_Water:                 v_Water = fv(p, h_Water_int, 1, 0,0,0,0,0,0,0,0,0,0,0,0);
f_v_Material1:     v_Material1 = TCM.TCM_v_t(t, TCM.ID1);
f_v_Material2:     v_Material2 = TCM.TCM_v_t(t, TCM.ID2);

# density
f_rho:      rho * v = 1.0;

# enthalpies of the TCM stream

# Using enthalpies with different zero point definitions and making them consistent
# All enthalpies are defined here to be zero at 25°C

f_h_Oil:           h_Oil = T_Composition.htf_h_t(t) - T_Composition.htf_h_t(25);

f_h_Material1:     h_Material1 = TCM.TCM_h_t(t, TCM.ID1) - TCM.TCM_h_t(25, TCM.ID1);
f_h_Material2:     h_Material2 = TCM.TCM_h_t(t, TCM.ID2) - TCM.TCM_h_t(25, TCM.ID2);
#f_h_Material1:    h_Material1 = TCM.TCM_h_t(t, TCM.ID1);
#f_h_Material2:    h_Material2 = TCM.TCM_h_t(t, TCM.ID2);


f_h_Water:                 h_Water = h_Water_int - fhpt(1, 25, 1, 0,0,0,0,0,0,0,0,0,0,0,0);
```

```
# for internal use (to allow using fv(p,h,...) above
f_h_Water_int:    h_Water_int = fhpt(p, t, 1, 0,0,0,0,0,0,0,0,0,0,0,0);

# Water phase and water vapour pressure

#f_Water_Phase:p_H2Os = (exp(73.649 - 7258.2/(t+273.15) - 7.3037*ln(t+273.15) +
(4.1653*10^(-6))*(t+273.15)^2))/100000;
f_p_H2Os:         p_H2Os*1e5 = exp(73.649 - 7258.2/(t+273.15) - 7.3037*ln(t+273.15) +
(4.1653*10^(-6))*(t+273.15)^2);

# -1 -> liquid /// 1 -> gas
f_Water_Phase:  if (p > p_H2Os) then
                        Water_Phase = -1;
                else
                        Water_Phase = 1;

# Pressure for liquid water
f_p_required:     p_required = p_H2Os + 0.1;
```

Variables

p	pressure
t	temperature
h	enthalpy
v	specific volume
rho	density
mass	total mass flow
mass_Material1	mass flow material 1
mass_Material2	mass flow material 2
mass_Oil	mass flow oil
mass_Water	mass flow water
w_Material1	mass fraction material 1
w_Material2	mass fraction material 2
w_Oil	mass fraction Oil
w_Water	mass fraction Water
w_Solids	solid fraction
X_Material1_Oil	mass ratio of material1 to oil
X_Material2_Oil	mass ratio of material2 to oil
h_Material1	enthalpy of material 1
h_Material2	enthalpy of material 2
h_Oil	enthalpy of oil
h_Water	enthalpy of water with the same zero point as the other substances
h_Water_int	enthalpy of water (for internal processing)
v_Material1	specific volume material 1
v_Material2	specific volume material 2
v_Oil	specific volume oil
v_Water	specific volume water
Water_Phase	-1 liquid // +1 gas
p_H2Os	saturation pressure of water
p_required	minimal required pressure for liquid water

3.22. q_cond_trans

Purpose

conductive heat transfer through a wall (solid boundary)

3.22.1. q_cond_trans

Purpose

conductive heat transfer through a wall (solid boundary)

Model equations

```
# temperature differences, MTD and heat transfer area*heat transfer coefficient

# definition of prevailing temperature differences
f_dt_in:    dt_in = t_hot_in - t_cold_in;
f_dt_out:  dt_out = t_hot_out - t_cold_out;

# Mean temperature difference is the logarithmic temperature difference.
# For nearly equal inlet and outlet temperature differences, it is approximated
# by the arithmetic mean (which is the asymptotic approximation).
f_MTD: if abs(dt_in/dt_out) >=1.2 || abs(dt_out/dt_in) >=1.2 then
            MTD * ln(dt_in/dt_out) = (dt_in-dt_out);
  else
            MTD = (dt_in+dt_out)/2.0;

# heat transfer equation
f_q_trans:          q_trans = htc_area * MTD;

# For now, there is no explicit variable which represents the heat transfer area.
# If there is an explicit area, also a heat transfer coefficient is required.
```

Variables

q_trans	transferred heat
dt_in	temperature difference at the inlet of the cooling stream
dt_out	temperature difference at the outlet of the cooling stream
MTD	mean temperature difference
htc_area	heat transfer coefficient x heat transfer area
t_hot_in	temperature of the hot side at the inlet
t_hot_out	temperature of the hot side at the outlet
t_cold_in	temperature of the cold side at the inlet
t_cold_out	temperature of the cold side at the outlet

3.23. OR_Composition

Purpose

Represents an organic carbon-based working fluid and some other alternatives used in organic rankine cycles, refrigeration and heat pump processes.

3.23.1. OR_Composition

Purpose

Represents an organic carbon-based working fluid and some other alternatives used in organic rankine cycles, refrigeration and heat pump processes.

Model equations

```
##################################################################################
# The following are all coming from the REFPROP database

# pure fluids

# Ammonia
ifl FluidName == Ammonia then
  f_Ammonia: FluidID = 11;
endifl

ifl FluidName == Butane then
  f_Butane: FluidID = 12;
endifl

ifl FluidName == Butene then
  f_Butene: FluidID = 13;
endifl

# Carbon Dioxide
ifl FluidName == CO2 then
  f_CO2: FluidID = 14;
endifl

# Ethanol
# Other names: ethyl alcohol
# molecular formula C2H5OH
ifl FluidName == Ethanol then
  f_Ethanol: FluidID = 15;
endifl

ifl FluidName == Isobutane then
  f_Isobutane: FluidID = 16;
endifl

ifl FluidName == Isobutene then
  f_Isobutene: FluidID = 17;
endifl

# Three isomers exist for pentane:
# n-pentane (linear molecule), isopentane and neopentane
```

```
# All three have the molecular formula C5H12


# Isopentane
# Other name: methylbutane, 2-methylbutane
ifl FluidName == Isopentane then
  f_Isopentane: FluidID = 18;
endifl

# Neopentane
# Other name: dimethylpropane, 2,2-dimethylpropane
ifl FluidName == Neopentane then
  f_Neopentane: FluidID = 19;
endifl

# Pentane
# Other name: n-Pentane
ifl FluidName == Pentane then
  f_Pentane: FluidID = 20;
endifl

ifl FluidName == Propane then
  f_Propane: FluidID = 21;
endifl

# Toluene
# Other names: toluol, methylbenzene, phenylmethane
# molecular formula C7H8
ifl FluidName == Toluene then
  f_Toluene: FluidID = 22;
endifl

# Water
ifl FluidName == Water then
  f_Water: FluidID = 23;
endifl

# Cyclopropane
# molecular formula C3H6
ifl FluidName == Cyclopropane then
  f_Cyclopropane: FluidID = 24;
endifl

# Cyclopentane
# molecular formula C5H10
ifl FluidName == Cyclopentane then
  f_Cyclopentane: FluidID = 25;
endifl

# Cyclohexane
# molecular formula C6H12
ifl FluidName == Cyclohexane then
  f_Cyclohexane: FluidID = 26;
```

```
endifl

# Siloxanes
ifl FluidName == D4 then
  f_D4: FluidID = 31;
endifl

ifl FluidName == D5 then
  f_D5: FluidID = 32;
endifl

ifl FluidName == D6 then
  f_D6: FluidID = 33;
endifl

ifl FluidName == MDM then
  f_MDM: FluidID = 34;
endifl

ifl FluidName == MD2M then
  f_MD2M: FluidID = 35;
endifl

ifl FluidName == MD3M then
  f_MD3M: FluidID = 36;
endifl

ifl FluidName == MD4M then
  f_MD4M: FluidID = 37;
endifl

ifl FluidName == MM then
  f_MM: FluidID = 38;
endifl

# refrigerants
ifl FluidName == R123 then
  f_R123: FluidID = 51;
endifl

ifl FluidName == R1234yf then
  f_R1234yf: FluidID = 52;
endifl

ifl FluidName == R1234ze then
  f_R1234ze: FluidID = 53;
endifl

ifl FluidName == R124 then
  f_R124: FluidID = 54;
endifl

ifl FluidName == R125 then
```

```
  f_R125: FluidID = 55;
endifl

ifl FluidName == R134a then
  f_R134a: FluidID = 56;
endifl

ifl FluidName == R142b then
  f_R142b: FluidID = 57;
endifl

ifl FluidName == R143a then
  f_R143a: FluidID = 58;
endifl

ifl FluidName == R152a then
  f_R152a: FluidID = 59;
endifl

ifl FluidName == R227ea then
  f_R227ea: FluidID = 60;
endifl

ifl FluidName == R236ea then
  f_R236ea: FluidID = 61;
endifl

ifl FluidName == R236fa then
  f_R236fa: FluidID = 62;
endifl

ifl FluidName == R245ca then
  f_R245ca: FluidID = 63;
endifl

# R245fa
# Other name: 1,1,1,3,3-pentafluoropropane
# molecular formula CF3CH2CHF2
ifl FluidName == R245fa then
  f_R245fa: FluidID = 64;
endifl

# R365mfc
# Other name: 1,1,1,3,3-pentafluorobutane
# molecular formula CF3CH2CF2CH3
ifl FluidName == R365mfc then
  f_R365mfc: FluidID = 160;
endifl

# Novec649 (3M brand name)
# Other name: 1,1,1,2,2,4,5,5,5-nonafluoro-4-(trifluoromethyl)-3-pentanone
# Other name: FK-5-1-12
# molecular formula C6F12O
```

```
ifl FluidName == Novec649 then
  f_Novec649: FluidID = 153;
endifl

# mixtures (treated as pseudo-pure fluids)

ifl FluidName == R407C then
  f_R407C: FluidID = 83;
endifl

ifl FluidName == R410A then
  f_R410A: FluidID = 86;
endifl

ifl FluidName == R507A then
  f_R507A: FluidID = 91;
endifl

ifl FluidName == Air then
  f_Air: FluidID = 95;
endifl

# end of REFPROP section
################################################################################

# user defined fluid uses the REFPROP calculation methods and file format
#ifl FluidName == _userdefined then    # *.FLD is copied to userdefined1.fld file
#    f_userdefined1: FluidID = -1;
#endifl
```

Variables

FluidID	fluid identifier

Switches

FluidName (Air, Ammonia, Butane, Butene, CO2, Cyclohexane, Cyclopentane, Cyclopropane, D4, D5, D6, Ethanol, Isobutane, Isobutene, Isopentane, MD2M, MD3M, MD4M, MDM, MM, Neopentane, Novec649, Pentane, Propane, R123, R1234yf, R1234ze, R124, R125, R134a, R142b, R143a, R152a, R227ea, R236ea, R236fa, R245ca, R245fa, R365mfc, R407C, R410A, R507A, Toluene, Water)	working fluid to be used

3.24. TCM

Purpose

thermochemical material working pair

3.24.1. TCM

Purpose

thermochemical material working pair

Model equations

```
# Selection of the thermochemical material system.
# Note that only certain selected working pairs of the available materials make sense.
```

```
# H3BO3 and HBO2
ifl Material_System == Boric_Acid then
  # boric acid H3BO3
  f_BA_1:          ID1 = 1;
  # metaboric acid HBO2
  f_BA_2:          ID2 = 2;
endifl

# CuSO4.5H2O and CuSO4.H2O
ifl Material_System == Copper_Sulfate then
  # copper sulfate pentahydrate CuSO4.5H2O
  f_CS_1:          ID1 = 3;
  # copper sulfate monohydrate CuSO4.H2O
  f_CSA_2:         ID2 = 4;
endifl

# CaCl2.2H2O and CaCl2
ifl Material_System == Calcium_Chloride then
  # calcium chloride dihydrate CaCl2.2H2O
  f_CC_1:          ID1 = 5;
  # anhydrous calcium chloride CaCl2
  f_CC_2:          ID2 = 6;
endifl

# K2CO3.3/2_H2O and K2CO3
ifl Material_System == Potassium_Carbonate then
  # potassium carbonate sesquihydrate K2CO3.3/2_H2O
  f_PC_1:          ID1 = 7;
  # anhydrous potassium carbonate K2CO3
  f_PC_2:          ID2 = 8;
endifl
```

Variables

ID1	identifier material 1
ID2	identifier material 2

Switches

Material_System (Boric_Acid, Calcium_Chloride, Copper_Sulfate, Potassium_Carbonate)
Selection of the thermochemical material system

4. Economic modelling

In order to carry out an economic analysis, CENER has developed and implemented a specific financial model adapted to the requirements of the disruptive RESTORE concept. This model takes into account a wide variety of financial features and parameters.

The model presents a high flexibility, making possible the users to modify a high number of parameters in order to customize their analysis and fulfil their requirements. The main financial parameters are shown in Table 4, together with some default values, that could be modified by the users:

Table 4 – Economic parameters and default values.

Inputs		Default value
General	Useful life (years)	28
	Yearly inflation rate (%)	2,50%
Heat charge	Overall heat feed-in tariff (€/kWh) Pucharse	0.1
Heat discharge	Overall heat feed-in tariff (€/kWh) Sales	0.2
	Residual Value	0
El. Charge	El. overall feed-in tariff (€/kWh) Pucharse	0.3
El. Discharge	El. overall feed-in tariff (€/kWh) Sales	0.7
	Residual Value	0
Debt	Debt (% total investment)	75%
	Debt repayment period (years)	15
	Yearly debt interest (%)	5,00%
	Grace period - Interest-only period (years)	3
Equity	Equity (% total investment)	25%
	Yearly rate of return on equity capital (%)	12%
Other Expenses	O&M costs per unit of net electricity (€/kWh)	0,0250
	Insurance cost (%)	0,50%
	Land occupation (m2)	0
	Land cost per unit area (€/m2)	0,000
	Decommissioning Cost	0%
Taxes	Tax rate (%)	35,00%
	MAT - Minimum Alternative Tax (%)	0,00%
	Tax Credit Expiration Period (years)	10
	Depreciation Schedule (yearly %)	----

In addition, the economic model receives the following inputs from other models and sub-models of the system, as shown in Table 5.

Table 5 – Economic input parameters from other models of the system.

Inputs		From submodel
Investment	Plant capital cost (€)	CAPEX estimation module
Heat charge	Yearly net electricity Charged (kWh/year)	Technical model
Heat discharge	Yearly net electricity discharged (kWh/year)	Technical model
El. charge	Yearly net electricity Charged (kWh/year)	Technical model
El. discharge	Yearly net electricity discharged (kWh/year)	Technical model

The following aspects have been considered for the financial model:

- The inflation rate is considered to be constant over the investment life, and it is directly applied to the energy prices, O&M costs, land costs, etc.
- The Yearly Net Electricity Production and Consumption can be varied for each year by assigning a percentage of the reference production.
- The Yearly Net Heat Production and Consumption can be varied for each year by assigning a percentage of the reference production.
- The Plant Capital Cost investing schedule can be described by assigning a percentage of the total Plant Capital Cost for a number of years.
- The debt payment amortization system considered for the analysis is the French System (constant instalment).
- In addition to the corporate tax, a Minimum Alternate Tax, MAT (See description later in this document) and tax credit can be applied (set 0 years in the Tax Credit Expiration Period for not considering tax credit).
- The depreciation schedule considered for the capital assets can be provided as a percentage of the total investment for each year of the plant life. The appropriate calculation of the depreciation schedule has to be done by the user.
- Costs related to grid connection and energy deliver to the energy networks have not been considered.

The financial model calculates the cash flows for the project and, from these, the most common financial indicators, such as the Net Present Value (NPV), the Internal Rate of Return (IRR), and the Debt Service Coverage Ratio (DSCR) for each year. Besides, due to the particularities of the RESTORE concept, able to store and deliver energy in both forms: electricity and heat, the cost of the energy is analyzed from two different perspectives:

- Electricity approach, estimating the Levelized Cost Of Stored for Electricity (LCOS_el) and considering the heat as a side product.

- Heat approach, estimating the Levelized Cost Of Heat (LCOS_heat) and considering the electricity as a side product.

In summary, the main outputs of the economic model are shown in Table 6 below

Table 6 – Main Outputs of the Economic input parameters from other models of the system

Output	Units
Project Net Present Value (NPV)	€
Project Internal Return Rate (IRR)	%
Levelized Cost of Storage for Electricity	€/kWh
Levelized Cost of Storage for Heat	€/kWh
Plant capital cost	(€)

In next sections both approaches, LCOS for electricity and for heat are described in detail.

4.1.1. The Levelized Cost of Storage for Electricity

The Levelized Cost Of Storage for Electricity (LCOS_el) is an economic indicator of great usefulness when comparing technological options, concerning energy storage systems, from an economic point of view.

The LCOS_el can be defined, in actual euros (or dollars), as the value that would have to be assigned to every unit of dispatchable electricity delivered by the energy storage system throughout a determined period in order to equal the total costs incurred during this period, also expressed in actual euros (or dollars).

In this approach, the electricity is considered the main product purpose of the study meanwhile the heat delivered is considered as a side product, thus representing an income in the calculations.

That means that, if *Qn* is the total electric energy produced by the power plant in the year *n*, *d* is the discount rate and Cn is the total cost incurred at the power plant in the year n, according to the definition of LCOS_el, the following equality:

$$\sum_{n=1}^{N} \frac{Q_n \times LCOS_{el}}{(1+d)^n} = \sum_{n=1}^{N} \frac{C_n}{(1+d)^n}$$

From this equality, the following mathematical expression can be obtained for the calculation of the LCOS_el:

$$LCOS_{el} = \frac{\sum_{n=1}^{N} \frac{C_n}{(1+d)^n}}{\sum_{n=1}^{N} \frac{Q_n}{(1+d)^n}}$$

Where:

- *LCOS_el* is the Levelized Cost Of Storage for Electricity
- *n* is the number of years (period) for carrying out this calculation, which usually coincides with the expected useful life of the power plant.
- *Cn* is the net cost (cost minus benefits from side products) incurred during the year *n*, which includes, among others, the investment cost in the first years of construction and commissioning of the plant, O&M, cost associated to energy, as well as financial costs, including main depreciation and interest payment.
- *Qn* is the yearly electricity delivered by the storage system in the year *n*.
- *d* is the discount rate, which indicates the present value of the future cash flows that depend on, among others, the debt interest, inflation rate and expected investment profitability.

The next step is to calculate the different factors required by the LCOS_el expression.

As a discount rate, the Weighted Average Cost of Capital (WACC) needed to finance the construction of the energy storage system has been used. The WACC is the minimum return that a company must earn on an existing asset base to satisfy its creditors, owners, and other providers of capital, so that they will not invest elsewhere. It can be calculated by using the following expression:

$$d = WACC = \frac{E_I \times R_{EI} + D_I \times R_{DI}}{E_I + D_I}$$

Where:

- E_I is the self-financing (equity capital) of the investment
- R_{EI} is the yearly rate of return on equity capital (profitability expected from self-financing)
- D_I is the bank debt of the investment
- R_{DI} is the yearly debt interest

In addition, the annual net costs include the initial investment, all type of costs derived from the achievement of a proper operation of the plant (operation and maintenance costs), energy cost, as well as financial costs, etc.

In addition, the value of these annual costs, as well as the relative weight of specific costs composing them, varies along the years. Next expression shows the way these costs can be calculated:

$$\sum_{n=1}^{N} \frac{C_n}{(1+d)^n} = \sum_{n=1}^{N} \frac{I_n + OM_n + FC_n + EC_n - TS_n}{(1+d)^n}$$

Where:

- I_n is the investment of equity capital in the year n. As the LCOS value is calculated from a promoter point of view of the project regarding construction, commissioning and operation of system, only the equity capital is considered as investment; the rest of the capital is considered as a debt. This magnitude is typically higher than cero during the first years of the analysis, in which the system is under project and construction, and cero for the rest of the period considered for the calculation of the LCOS. Even if the simplest way to consider the investment is to consider that the total equity capital is invested in the initial instant, this simplification has not been considered as it underestimates the value of the money flow during the construction period. Instead, for each case, more realistic criteria can be rather considered according to the schedule of equity capital contributions during the whole useful lifetime of the power plant.
- OM_n is the anual operation and maintenance (O&M) cost in the year n. These costs include both O&M costs of the central (operation labour, maintenance, etc.) and land renting costs.
- EC_n is the annual energy cost in the year n. These costs include, expenses such as the cost of the electricity and heat used for charging the system, as well as incomes, associated to the side products (in this case the heat delivered).
- FC_n is the annual financial cost in the year n. This cost includes both depreciation of the principal payment and the debt interest payment. As it has been already explained, the calculation of the LCOS has been carried out from the stockholder point of view, so the bank financing (debt) is considered as an expense throughout the years and not as an initial investment. Logically, in order to obtain the financial costs and their distribution during the years, a repayment period and/or a grace period has to be considered.
- TS_n is the tax saving in the year n, which is calculated for each year as the tax rate considered multiplied by the sum of the deductible expenses: O&M expenses, depreciation of investments and the corresponding part of the financial costs interest payment. This term is sometimes not considered depending on the type of analysis performed (that means considering a tax rate value = 0%)

4.1.2. The Levelized Cost of Heat

Similarly, an estimation of the Levelized Cost of Storage for Heat can be conducted following a similar procedure. In this scenario, the calculation process is analogous to the aforementioned approach. However, the underlying assumption that differentiate both approaches is the consideration of the Heat as the primary product, meanwhile the delivered electricity represents the side product which represent an income when calculate the energy costs.

4.1.3. The investment cost module

The investment cost module estimates the total investment cost. The estimated value is then considered for its use in the overall economic model. The calculation of the total investment cost is based on the estimation of the cost of the main components which compose the overall energy storage system of RESTORE. This is considering the main components of the organic cycles as well as the main components which are involved in the thermochemical energy storage. The cost of each component is estimated based on its specific correlation which takes as input one or two characteristic parameters which represent the nominal conditions and size of the component. Each correlation is unique and represents how the cost scales with the size of the component. As example, the logarithmic equation for the estimation of component cost is shown below:

$$\log(\mathrm{C_{correlation}}) = \mathrm{K_1} + \mathrm{K_2}\log(\mathrm{x}) + \mathrm{K_3}(\log \mathrm{x})^2$$

Where C_{base} is the resulting cost, $K_{1,2,3}$ represents the constants of the equation and x is the representative parameter of the component.

After the calculation, the cost is updated considering the CEPCI factor using the following equation:

$$\mathrm{C_{present}} = \frac{\mathrm{CEPCI_{present}}}{\mathrm{CEPCI_{correlation}}} \cdot \mathrm{C_{correlation}}$$

Where $C_{present}$ is the updated cost, $\mathrm{CEPCI_{present}}$ represents the index in the present year, and $\mathrm{C_{correlation}}$ and $\mathrm{CEPCI_{correlation}}$, cost resulting from the correlation and the CEPCI in the year of the correlation, respectively. constants of the equation and x is the representative parameter of the component.

Furthermore, the model has the possibility of including a contingency factor in order to consider additional costs related to auxiliary systems or other unknown aspects by the user which could add uncertainties. As default value, this parameter is set in 15%

5. Conclusion

This document (D5.1) reported about the modelling of individual components of the overall RESTORE system, presenting also the economic model for the RESTORE project, including results and software-models related to the work carried out in Task T5.1, concerning the “Modelling of individual components of the overall RESTORE system”.

Individual models to represent the overall RESTORE system have been developed. These models are now contained in the dedicated model library RESTORE_Lib. The component models specifically developed for the RESTORE project have been described in detail. The model equations and variables representing individual models are listed, and the icons representing the individual units are displayed.

An extensive and detailed economic model has been implemented. The model allows to the user to modify a broad number of details and considerations for the economic analysis. In addition, the model receives inputs from the technical simulations and the investment cost from the CAPEX calculation submodule. As result, the module shows to the user results such as the Net Present Value, the Internal Return Rate, the Investment Cost and the Levelized Cost of Storage for Heat and Electricity.

The results presented in this deliverable will directly feed the development of the WP5 tasks: T5.2 (Techno-Economic Modelling of the Integrated Systems), T5.3 (Web-Platform Adaptation for RESTORE Dynamic and Techno-economic Modelling to represent the Use-Cases), and T5.4 (Implementation, Optimization, Management & Validation of RESTORE Use-Cases using the Simulation Web Platform), and it will also influence the further development of task T5.5 (Replication Strategy via Stakeholders additional Cases).

6. References

[1] European Commission, Horizon 2020 Project RESTORE “Renewable Energy based seasonal Storage Technology in Order to Raise Environmental sustainability of DHC”, https://cordis.europa.eu/project/id/101036766,Grant Agreement GA Nr.101036766, August 207th, 2021.

[2] SIMTECH GmbH, “The Process Simulation Environment IPSEpro”, https://www.simtechnology.com/cms/ipsepro/process-simulation-and-heat-balance-software, © 2023 SimTech GmbH.

[3] SIMTECH GmbH, “IPSE GO: The Future of Simulation”, https://about.ipsego.app/, © 2023 SimTech GmbH.

[4] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D1.1 - Report on Requirements and Specifications of the Overall Concept, ed.: F. Cabello (CENER), March 2023.

[5] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D1.4 - Specifications of RESTORE Use-Cases and Models, ed.: F. Dargam (SIMTECH), E. Perz (SIMTECH), September 30th, 2023.

[6] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D2.3 - Report on TCES Task D2.3 - Small-scale (1-2kW) TCES reactor tested and optimized, ed.: S. Denner (TU Wien), January 2024.

[7] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D2.4 - Design report of the 30 kW/150 kWh TCES reactor, ed.: G. Wedl (TU WIEN), L. Schmieder (TU WIEN), F. Winter (TU WIEN), January 2024.

[8] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D2.5 – Dedicated models for the reactor simulation, ed.: A. Werner (TU Wien), L. Schmieder (TU Wien), January 2024.

[9] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D3.1 - Numerical model for HPORC systems optimization and application to different Test Cases, ed.: M. Astolfi (POLIMI), D. Alfani (POLIMI), October 2023.

[10] European Commission, RESTORE Horizon 2020 Project GA Nr.101036766. Deliverable D5.10 - RESTORE Replication Strategy V1, ed.: F. Dargam (SIMTECH), E. Perz (SIMTECH), November 2023.