# EBSILON® *Professional*

Release Notes (English)

Release 17.0

iqony

# Release Notes

# EBSILON®*Professional*

Release 17.0

## Contents

# 1. Calculation Kernel

## 1.1. New Components

### 1.1.1. Component 171: Fly Wheel Energy Storage

Component 171 allows to model a fly wheel energy storage system. Here the energy is stored in mechanical form as rotational energy of a rotor. The component has both shaft pins and electric pins. The energy content of the storage system is proportional to the rotational speed of the rotor. An acceleration thus causes the storage system to charge and a decrease in the rotational speed causes it to discharge. Generally, an acceleration of a rotating mass like a turbine rotor, e.g., can be balanced with the help of this component as well.
Please refer to the Online Help for further details.

## 1.2. Extensions of Existing Components

### 1.2.1. General

For all components that explicitly prevent an end of the calculation by setting the internal state to "non-converged", a warning message will now be output if the simulation is aborted because the maximum number of iterations has been reached. Affected components are usually iteration controllers or components with internal closed-loop control that additionally use a closed-loop control reaction time. However, they can also be user-defined components with EbsScript code or external Dll using the Signal-Not-Converged mechanism.

### 1.2.2. Components 12 and 39: Result Values for Controllers

The result values for the controllers have been extended to facilitate an analysis of the closed-loop control procedure.

The result values RSCV for the setpoint value, RACT for the achieved target value, and RCORR for the value of the controlled variable have already been available. These have been retained. However, please note that RCORR specifies the value of the controlled variable that the controller has calculated in the current iteration step. It is not identical to the line value for the controlled variable as the controller does not set the value directly but only calculates a gradient that is included in the equation system. The value of the controlled variable that is actually reached therefore does not only depend on the controller itself but results from the solution of the equation system.

Therefore, a new result value CORR has been added that specifies the value of the controlled variable resulting after the equation system has been solved, as well as a result value CORR0 that shows the value of CORR in the previous iteration step, and the change of the correction variable CHCORR (= CORR-CORR0).

The result value DEV (= RACT–RSCV) specifies the difference between setpoint value and actual value. CHDEV specifies how far DEV has changed compared to the previous iteration step.

In what follows, there are some result values internally used by the controller to determine the gradient between target value and corrected variable that is included in the equation issued by the controller:

REF is the reference variable determined by the course of iteration of target value and setpoint value which is used for the internal normalization of the deviation of setpoint value and actual value; RELACT (= CHDEV/REF) is the change of the actual value related to this reference variable, and RELCORR (= CHCORR/CORR) is the relative change of the controlled variable in the last iteration step. The ratio of these two quantities is RATREL (= RELCORR/RELACT).

By means of the specification value CHL2 and CHL3 respectively and the characteristic line CCHL2 and CCHL3 respectively, the user can limit the relative change of the controlled variable calculated by the controller. The value of this characteristic line is now shown as result value RCCHL, and the overall resulting change limit as result value RCHL.

Due to this limitation as well as the damping and the limitation by the minimum and maximum values for the controlled variable, often a modified value of RATREL, which is displayed in the result value RATRELUSED, is used for the calculation of the gradient.

The following result values show the effect of individual damping and limiting mechanisms: CHCORRORIG specifies the originally calculated change of the controlled variable, CHCORRDAMP specifies the value after the damping, and CHCORRCHL specifies the value after applying the change limit.

GRADMAT shows the gradient that is included in the equation system. It specifies how the controlled variable has to be modified in order to change the actual value by one unit.

Due to measures for improving the convergence, however, the mentioned values are modified in various places as well.

The previous result value RCHL3 is identical with CHCORRCHL. RCHL3 has therefore been admitted as alias name for CHCORRCHL so that scripts and expressions where RCHL3 occurs continue working as before.

### 1.2.3. Components 15 and 16: Heat Extraction and Injection

For these components, the pressure drop in off-design is limited to 10 times the nominal value in analogy to Component 13 (Piping) due to excessive pressure drops when the design mass flow is too low. Other than in the case of Component 13, the factor 10 is fixed here and cannot be set by means of a specification value (DPDPNMAX). When this limitation activates, it is advisable to select another design point anyway, e.g., to carry out a local design in the current subprofile.

### 1.2.4. Components 20 and 70: Pressure Calculation for Drums

The drum components now allow to have the drum pressure calculated by the components in off-design.
In this way, counterclockwise processes can be displayed in EBSILON with similarly good convergence properties as clockwise processes.

In the case of clockwise processes, it is the condenser component that allows to directly calculate the precise pressure whose condensation enthalpy allows to have the nascent condensation heat dissipated by the cooling water.

In the case of the drums, it was necessary to use a controller to set the respective precise pressure where the evaporation enthalpy allows the heat exchanger to provide the required evaporation heat. Now the components can calculate this directly.

However, an internal iteration is required for this (as in the case of the condenser). The following parameters are available for the fine-tuning of this internal iteration:

- START (start value for the internal iteration)
  The specification is optional. If no start value is specified, Component 20 assumes a value of 1 bar and Component 70 assumes the nominal value P1N.
- CHL (max. relative change of the pressure from one iteration step to the next)
- ITSTEP (frequency of the recalculation of the pressure)
  For instance, at ITSTEP = 1 the pressure is recalculated in each iteration step (default value) and at ITSTEP = 5 only in every fifth iteration step.

Moreover, the parameter DERIVLEV is available for the convergence tuning. It allows to adjust when the partial derivatives are to be included in the equation system in the mode Pressure Calculation. These will be activated if, during the iteration, the calculated pressure differs by less than DERIVLEV compared to the previous iteration step. As the dependency is very sensitive (in the sample model, the pressure changes by 100 mbar if the mass flow only changes by 1 g/s), the default value has been set to 0. However, there may be models where this parameter leads to improvements.

There are two variants for the internal iteration for the pressure:

- At FSPEC = 3 (Component 20) and FSPECP = 3 (Component 70) respectively, the pressure variation is implemented in such a way that the steam volume flow remains constant (i.e., the mass flow changes with the pressure). This variant is recommended if the steam quantity attunes by a compressor that can transport a certain volume flow.
- At FSPEC = 4 and FSPECP = 4 respectively, the pressure variation is implemented at a constant steam mass flow.

In Component 20, this feature has partially been available already since Patch 16.03.

### 1.2.5. Component 30: Display of Temperature Differences with the Difference Meter

At FTYP = 2 (temperature difference), the result of the difference is now output as line value DELTA_T on the line.

The background: internally, EBSILON uses the variables Pressure, Mass flow, and Enthalpy. At FTYP = 2, the numerical value of the difference (in Kelvin) is written on the variable Enthalpy onto the line. Previously, the result was also displayed as enthalpy (in kJ/kg). However, this was unpleasant, especially when other units were to be used. Thus, at first it seemed natural to write the result onto the line as a temperature. This, however, also leads to an error in a unit conversion. With an identical temperature on both sides, T = 0°C would be displayed, leading to 32°F when switching over to Fahrenheit. Therefore, the value has now been written onto DELTA_T, which enables a correct unit switchover.

### 1.2.6. Component 34: Saltwater as Injection for the Flashbox

In this component, saltwater can now be used as injection (Pin 4) as well. Also, saltwater in the Universal fluid is now treated correctly (Please note: in the Universal fluid, it was possible to select saltwater before. However, as the component was unable to treat salt in the injection, the salt fraction from the injection was ignored).

### 1.2.7. Component 45 (Value Indicator)

#### 1.2.7.1. Joule-Thomson Coefficient (FTYP = 99)

With FTYP = 99, the Joule-Thomson coefficient can now be accessed. It is defined as

$$\mu = \partial T/\partial p|_h$$

and is a measure for the behavior of a gas when exiting a throttle:

- If $\mu = 0$ (as a rule for ideal gas), the temperature will not change.
- If $\mu > 0$ (e.g., air), the temperature will decrease in the event of expansion
- If $\mu < 0$ (e.g., hydrogen), the temperature will increase in the event of expansion.

#### 1.2.7.2. Internal Information (FTYP = 100)

FTYP = 100 serves to output internal information exchanged between the components via logic lines into the result arrays RAINFO and RAAINFO. These only serve for diagnostic purposes for the Hotline and should not be used by the user as this exchange of information may change from Release to Release.

#### 1.2.7.3. Fractions at 2-Phase States (FTYP = 101)

Just as FTYP = 10, FTYP = 101 serves to output the fraction of a phase in a 2-phase state. While FTYP = 10 usually refers to the “steam mass fraction”, i.e., the fraction of the gaseous phase in the phase transition liquid / vapor, FTYP = 101 can also be used for other phase transitions. The result value RPHASE indicates which phase transition it is:

0: no phase
1: liquid
2: vapor
3: vapor / liquid (VLE)
4: solid
5: solid / liquid (SLE)
6: solid / vapor (SVE)
7: vapor / liquid/solid

8: supercritical
9: liquid / liquid
10: solid / solid
11: vapor / liquid / liquid
12: solid / liquid / liquid
13: solid / solid / liquid
14: liquid / liquid / supercritical

Please note: With the libraries LibIce and LibCO2, FTYP = 10 can be used for the phase transitions solid/liquid and solid / vapor too as these libraries yield X-values between 10 and 11 for solid / liquid and X-values between 100 and 101 for solid / vapor.

#### 1.2.7.4. Total H + NCV

At FTYP = 102, the total of enthalpy and NCV is output. This total can be particularly useful for energy balance considerations.

#### 1.2.7.5. Innere Energie

At FTYP = 103, the internal energy is output. In the case of physical properties libraries with no implemented function for the internal energy, it is calculated according to

$$U = H - P * V$$

#### 1.2.7.6. Diagnostics: Course of Iteration

There is now a result array RARESULT in which the course of iteration of the RESULT value is displayed. The display is activated by means of the flag FITLOG. By default, the output is disabled in order not to produce unnecessary amounts of data.

#### 1.2.7.7. Isochoric Specific Heat Capacity in the 2-Phase Range

While the isobaric specific heat capacity cp is not defined for pure substances in the 2-phase range as the temperature does not change during the phase transition at constant pressure, the isochoric specific heat capacity cv is defined at all times: if you keep the volume constant during the phase transition, the pressure will change and thus the boiling temperature too, so that the derivative

$$cv = \partial U/\partial T|_v$$

can be formed. Alternatively, the calculation can be performed via the relation

$$cv = T * \partial S/\partial T|_v$$

This value is now calculated in the 2-phase range both by Component 45 and by Component 168 (Quantity converter).

#### 1.2.7.8. Sonic Speed in the 2-Phase Range

Even though the actual expansion of a sound wave depends on the details of the geometry (bubble / droplet size, formation of a fluid film on the pipe wall) in the 2-phase range, a “sonic speed” can also be calculated in the 2-phase range via the thermodynamic relation

$w^2 = \partial p / \partial \rho | s$

This is done in this way both by Component 45 and by Component 168 (Quantity converter).

In fact, this quantity has practical significance for steam turbines as well:
Also in the 2-phase range, it is just the speed at which the flow rate through the turbine reaches its maximum. For this, however, it is necessary to calculate with static quantities, which Component 168 allows to do.

## 1.2.8. Component 48 (Value Transmitter Switch)

### 1.2.8.1. On / Off Flag FFU

This component now features a flag FFU that allows to switch the transmission on or off.

### 1.2.8.2. FOUT = 0

As in the case of Component 36 (Value transmitter), the setting FOUT = 0 is now available here too; it serves to set the output signal to the same value as the input signal. This is now the default setting too.

As in most cases the input and output signals are of the same type, it is thus not necessary to switch FOUT when switching FIN.

### 1.2.8.3. Temperature Transmission (FIN = 2)

It is now possible to transmit the temperature as well. This makes sense when the involved lines have different compositions as otherwise it is easier to transmit the enthalpy.

The options for scaling, use of characteristic lines, and transmission onto another quantity, however, are not implemented for the temperature transmission in Component 48. Due to the peculiarities of the unit conversion of the temperature, additional specifications would be required for this purpose; these, however, are implemented in Component 36.

Therefore, when setting FIN, the flag FOUT and the flags and specification values for the scaling will become inactive.

## 1.2.9. Components 77: Calculator

The FOPR switch allows two new modes:

- FOPR = 14: Always forward data from port 1
- FOPR = 15: Always forward data from port 2

## 1.2.10. Components 111 (Natural Draft Cooling Tower), 112 (Forced Draft Cooling Tower): Transient Extension

Components 111 (Natural draft cooling tower) and 112 (Forced draft cooling tower) have been extended by transient functionality in Release 17.

In transient states, the cooling towers have a great thermal inertia. This applies to the collecting water basin in particular; it usually has a large volume and thus acts as a large water reservoir. But also, the fill and packing materials have a corresponding mass that charges or discharges the heat in transient state. From Release 17 on, these effects can be considered in Components 111 and 112. Please refer to the Online Help for further details.

## 1.2.11. Component 118 (Direct storage / mass storage)

### 1.2.11.1. Specification and Calculation of the Area-Specific Heat Losses

Component 118 now allows to specify the area-specific heat losses. For this purpose, a new flag FQLOSS has been added. For

- FQLOSS = 0, the heat loss is specified by the temperature-dependent value QLOSSR (kW/K) (as before)
- FQLOSS = 1, by contrast, the area and temperature difference-specific heat loss (kW/m$^2$K) is used

For the area and temperature difference-specific heat losses, a vertical cylinder is assumed, and a difference is made between the specific loss values

- QLS – towards the side of the cylinder (the loss-relevant area depends on the filling level of the storage system)
- QLT – up
- QLB – down.

Specifying the area-specific heat losses (FQLOSS = 1) is only possible with a clear definition of the cylinder geometry and of the area relevant for the loss respectively (FFILL = 0 and FLEV = 0 or FLEV = 1).

The area-specific heat losses can be reasonably estimated based on the assumption that in the heat transfer to the ambiance, the highest resistance arises in the insulation of the storage system. For a known insulation thickness Delta (m) and a known heat conductivity of the insulation LAMBDA (W/mK), the area- and temperature-specific loss value can be estimated as

QLOSS = LAMBDA/DELTA

### 1.2.11.2. Calculation of the Fluid Pressure

Component 118 now allows to calculate the fluid pressure, PSTO, PNEW as well as the pressure at Pin 2 (outlet) from the geodesic height of the fluids, which in turn depends on the filling level of the storage system. In addition, a superimposed pressure PLOAD is considered above the fluid column. For this purpose, a new value has been introduced on the flag FPSTO. For

- FPSTO = 2

the new fluid pressure PNEW is calculated from PLOAD and the current filling level. The pressure on Pin 2 results as the mean value between the pressure at the beginning of the time step – PSTO – and the pressure at the end of the time step – PNEW.
The described pressure calculation mode (FPSTO = 2) is only possible with a clear definition of the cylinder geometry and of the height relevant for the calculation respectively (FFILL = 0 and FLEV = 0 or FLEV = 1).

### 1.2.12. Component 117 (The Sun): Horizon Data Added

Component 117 has been extended by the characteristic line CHORIZON and the flag FHORIZON. At

- FHORIZON = 0 the horizon is horizontal (= 0°)
- FHORIZON = 1 the angle of elevation of the horizon is determined from the characteristic line CHORIZON. This is a function of the azimuth.

For reasons of compatibility, the horizon angle is only made available. It is up to the solar components to use this value.

### 1.2.13. Component 119 (Indirect Storage): Geometry-Based Pressure Drop Calculation

Component 119 has been extended by a geometry-based pressure drop calculation. The flag FDP has been introduced for this purpose. At

- FDP = 0 the pressure drop is calculated from the value DP12N and the corresponding off-design scaling (as before Release 17)
- FDP = 1 the geometry of the pipe is used for calculating the pressure drop.

The geometry-based pressure drop calculation only makes sense when Component 119 models a real pipe. For calculating the pressure drop in this case, the internal diameter DIAI, the pipe length LSTO, and the wall roughness KS are used besides the volume flow of the fluid.

### 1.2.14. Components 119, 145, 165, 166: New Specification Value FNUMSC

In Components 119 and 166, the numerical scheme for the calculation of the fluid temperatures depended on the flag FSPECM. For FSPECM = 1 (fluid mass neglected), the scheme CDS (central differences) was always used. For FSPECM > 1 (fluid mass considered), the upwind scheme was used in these components. In Components 145 and 165, the upwind scheme was always used. The CDS scheme has a higher accuracy compared to the upwind scheme. The upwind scheme, by contrast, is more stable for the internal convergence of the component.

From Release 17 on, there is a new flag FNUMSC in Components 119, 145, 165, and 166. It allows the user to select the numerical scheme independently of the FSPECM value. FNUMSC = 0 corresponds to the upwind scheme and FNUMSC = 1 corresponds to the CDS scheme.

If a switchover from CDS to upwind occurs in the internal iteration due to detected convergence problems, the user will be warned at the end of the simulation.

In models saved with EBSILON versions earlier than Release 17, the value of the FNUMSC flag will be set depending on the flag FSPECM when loading to reproduce the old results. However, this automatic setting of the flag will not work if FSPECM does not have a simple valid numerical value but e.g. contains a reference (Kernel expression) to other components. In such a case, the user him / herself may want to set the flag FNUMSC in such a way that the old results are reproduced.

## 1.2.15. Components 119, 138, 165, 166: Alternative Reference Value for the Pressure Drop Calculation in Off-Design, New Specification Value FDPBASE

Up to Release 17, the pressure drop calculation in Components 119, 138, 165, and 166 always referred to the specific volume at the inlet of the component. This may lead to higher inaccuracies if the temperature change between inlet and outlet is high, and the flowing fluid is compressible/gaseous.

From Release 17 on, there is a flag FDPBASE, where the following applies:

- FDPBASE = 0 – here the mean value of the specific volume between inlet and outlet is used as reference quantity for the pressure drop calculation in the off-design calculation
- FDPBASE = 1 – here the specific volume at the inlet is used as reference quantity for the pressure drop calculation in the off-design calculation (like up to Release 17)

In addition, a new nominal value V12N (the mean specific volume between inlet and outlet) has been added to the components as reference quantity.

## 1.2.16. Components 119, 165, 166: Consideration of the Heat Conductivity in the Fluid, New Specification Values FHC, CLMFL

Up to Release 17, the heat conductivity in the fluid was only considered in Component 145 and was neglected in Components 119, 165, and 166. This assumption is justified for the case of the flowing fluid in the storage system. For calculating a static fluid (e.g., standby operation of the storage system), however, it is necessary to consider the heat exchange between the fluid elements in the storage system. For this purpose, new specification values have been added in Components 119, 165, and 166.

The flag FHC controls the consideration of the heat conduction in the fluid:

- FHC = 0 – here the heat conduction in the fluid is completely neglected (like up to Release 17)
- FHC = 1 – here the heat conduction in the fluid is considered. In doing so, the physical properties tables of the fluid are used for calculating the heat conductivity.

In addition, the heat conductivity can be multiplied by a factor CLMFL. For instance, this may be required in the case of the free convection in the storage system.

## 1.2.17. Components 119, 165: Mode FSPECM = 4 Also Works with NFLOW > 1

Up to Release 17, the mode FSPECM = 4 – Mass flows specified, pressure calculated – was only available in Component 119. This mode only worked with one element in the flow direction NFLOW = 1.

From Release 17 on, the mode FSPECM = 4 is available both in Component 119 and in Component 165. Moreover, this mode works for any desired number of elements in the flow direction (NFLOW >= 1) in both components.

### 1.2.18. Component 137: Photovoltaic System

Methods for considering shading losses have been added. These can be activated by means of the flag FSHADING > 0.

- FSHADING = 1: The input matrix MXSHADINGLOSS is used.
- FSHADING = 2: The input value SHADINGLOSS is used.
- FSHADING = 3: The characteristic line CSHADINGLOSS is used.
- FSHADING = 4: The Kernel expression ESHADINGLOSS is used.

The output is reduced by these relative losses.

### 1.2.19. Component 146: Gearbox / Bearing

For the gearbox, the option to specify the rotational speed on both sides on the shaft has been created (FSPEC = -1). This facilitates the use of the new Component 171 (fly wheel energy storage).

### 1.2.20. Component 160: Alternative Specification of the Start Value for the Filling Level instead of the Fluid Enthalpy for 2-Phase Fluids

Up to Release 17, the value HFLSTART (start value for the fluid enthalpy in the storage system) had to be specified in the component. The start value for the filling level of the storage system for 2-phase fluids (e.g., steam gradient storage system) resulted from this.

Alternatively, it is possible to specify the start value for the filling level of the 2-phase fluid in the storage system from Release 17 on. For this purpose, there is a flag FENTHFL and a specification value XFLSTART.

- At FENTHFL = 0 the value HFLSTART is used (as before).
- At FENTHFL = 1 the value XFLSTART (Start value for volume fraction liquid – filling level) is used directly.

In the case of FENTHFL = 1, the corresponding fluid enthalpy will result from the value XFLSTART and the specified pressure PFLSTART. Using XFLSTART does not make sense for pure gas storage systems.

### 1.2.21. Component 165 (Thermal Regenerator / Bulk Material Storage)

#### 1.2.21.1. Pin 6 for Connecting an Additional External Heat Source / Heat Sink

Component 165 has been extended by another logic pin (pin 6). At this pin, a heat flow can be specified that is included in the energy balance of the component. For instance, it allows to model an additional external heater refrigeration system (the heat flow can also be negative) of the storage system.

#### 1.2.21.2. New Mode of Calculation, Flag FCALC

Component 165 has been extended by another mode of calculation. In the calculation of the component, up to Release 17 the internal storage mass and the shell were always balanced separately and in doing so, the entire fluid mass flow was distributed among two parts according to the area fractions. From Release 17 on, it is possible to consider both parts, the internal storage mass and the shell, simultaneously. This is a more realistic consideration and is recommended from Release 17 on. For reasons of compatibility, the previous separate consideration of the internal storage mass and the shell will be retained. A new flag FCALC has been introduced to choose between two modes of consideration.

- FCALC = 0 corresponds to the separate solution of the internal storage mass and the shell.
- FCALC = 1 corresponds to the simultaneous solution of the internal storage mass and the shell.

## 1.2.22. Component 168 (Quantity Converter)

**1.2.22.1. Specification of Speed from the Outside**
The component has been equipped with a logic inlet (Pin 3) to allow specification of the flow rate to be used for calculating the statistical quantities from outside. The flag FIN3 serves to activate the use of Pin 3. The desired speed then must be specified as enthalpy, e.g., by means of a measured value with FTYP = 33.

Also, a result value RVEL has been added that displays which flow rate the calculation has been carried out with.

**1.2.22.2. Joule-Thomson Coefficient**
Like in Component 45, the Joule-Thomson coefficient is now available in Component 168 as well. In Component 168, FMAPx = 415 must be set for this purpose.

## 1.2.23. Component 170: Header adapter

This component is an adapter that can be used on a header together with Components 148-150. To do so, the "external" pins 1 and 2 are installed into the header. Any Ebsilon components can be installed between the "internal" pins 3 and 4.

To perform correct calculations almost all components, require that the mass flow on pins representing real pipes is non-negative.

The header adapter ensures (cf. explanation of FCALCDIR) that the mass flow from Pin 3 to 4 is positive. Depending on the flow direction in the header, either the line data of Pin 1→3 and 4→2 are passed through (in the case of positive flow direction) or 2→3 and 4→1 are passed through (in the case of negative flow direction).

Simple constructions that do not change the mass flow on the connection from Pin 3 to 4 already work quite reliably (e.g. pressure / heat sinks and sources respectively).

The specification-value FCALCDIR is used to specify mapping between outer (1 and 2) and inner (3 and 4) ports:

- "Like larger mass flow: 0": the flow on the outer-port with the larger absolute mass flow determines the flow-direction

- “Like M1: 1”: the flow on port 1 determines the flow-direction
- “Like M2: -1”: the flow on port 2 determines the flow-direction
- “From higher to lower pressure: 2”: the outer port with the higher pressure determines the flow (e.g. for tubes, components with pressure drop)
- “From lower to higher pressure: -2”: the outer port with the lower pressure determines the flow (e.g. for pumps, compressors, components raising the pressure)
- “Use kernelexpression ECALCDIR: 3”: the result of kernelexpression ECALCDIR determines the flow-direction:
    - `evalexpr >= 0.0`: port 1→3 and port 4→2
    - `evalexpr < 0.0`: port 2→3 and port 4→1

In most cases the default specification-value “Derive from outside : 0” for FSPECM/FSPECP works fine. In case you get missing mass flow/pressure errors near the header-adapter changing those settings may help.

In the event of changes of the mass flow, it may be necessary to work with controllers.
In addition, the component possesses a logic pin (5) on which the mass flow takes the value of 1 (flow goes port 1→3 and port 4→2) or -1 (flow goes port 2→3 and port 4→1).

This information can e.g. be forwarded to Pin 4 of storage component 119 in order to inform it about the direction of the flow (cf. component119.FDIR = “According to mass flow port 4...”).

The default values FSPECM and FSPECP determine which of the variables M3, M4 or P3, P4 is viewed as externally defined. The setting “derive from outside” usually works. For constructions inside component 170 that require certain mass flow or pressure specifications, adjustments can be made here.

PLEASE NOTE: This component was in development during the patches for Release 16. Therefore, changes in the behavior of the component in Release 17 (e.g. new / other specification values, different behavior during the simulation, etc.) may occur.

## 1.3. Physical Properties

### 1.3.1. FRAGOLTHERM Oils

The following heat transfer oils of the company Fragol have been included for the line type oil / molten mass:

- Fragoltherm DPO
- Fragoltherm 660
- Fragoltherm HT
- Fragoltherm 620
- Fragoltherm 550
- Fragoltherm 590
- Fragoltherm 500
- Fragoltherm Q-HTF

- Fragoltherm Q-32-N
- Fragoltherm FG-35
- Fragoltherm Q-20
- Fragoltherm Q-7

As the manufacturer stated that Fragoltherm DPO is also usable in the gaseous phase, both "Fragoltherm DPO liquid" and "Fragoltherm DPO gaseous" have been included. However, it is necessary to keep in mind that no additional data for the pressure dependency were specified for the gaseous phase. For modeling the heat transfer, neglecting the pressure dependency is certainly accurate enough. The density, however, is only correct near the saturated steam point, and the approximation as incompressible fluid might lead to a distinctive underestimation of pump and compressor outputs respectively.

### 1.3.2. UserProps Dll

For the line type Universal fluid, there is the new library "User-defined properties (dll)" (UserProps for short). For this, the user can create a Dll that carries out the calculation of the physical properties.

In contrast to the existing User2Phase-Dll interface, UserProps allows to specify compositions (physical properties vectors) and additional attributes.

The interface is described in the file "user_props.h" in the subfolder "<Ebsilon installation folder>\Data\Examples\user_props_dll" and uses the language C++. A sample project is located there as well. It is recommended to use "Microsoft VisualStudio 2022" for creating an own library.

Each entry "User-defined properties (dll)" of a Universal fluid has the two standard attributes:

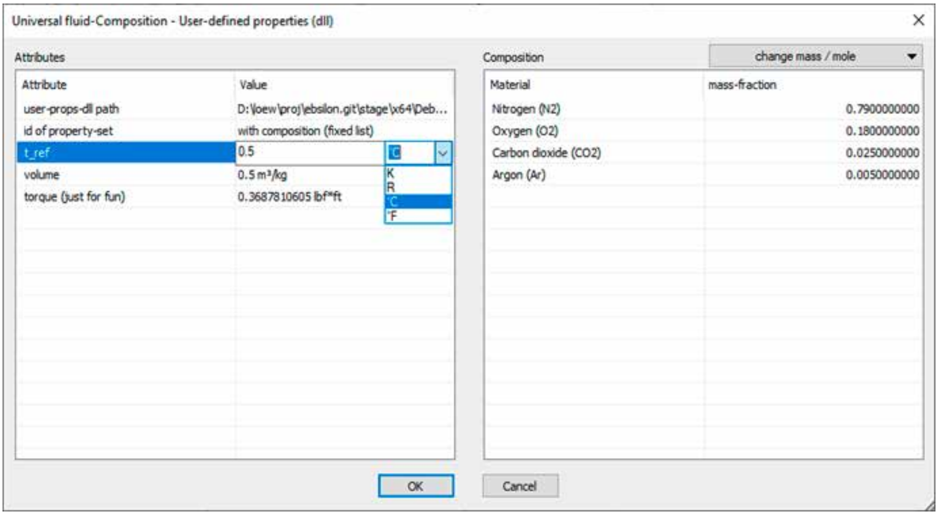
1. "user-props-dll path": path to the Dll. If applicable, several different Dlls can be used in a model or also fluid.
2. "id of property-set": Id of the selected property sets (physical property models). A Dll can have any number of different physical property models.

For each "property set", further attributes of the following types can be defined for a start:

- "double": floating point value (double precision)
- "value with unit"
- "list": list (combo box) for selection
- "string": text

Plus, a specification which substances may occur (if applicable, none at all for pure substance models).

Further information on the functions and types of the interface can be found in the file "user_props.h". Code for a sample Dll is located in the same directory.



Attributes

| Attribute | Value |
|---|---|
| user-props-dll path | D:\loew\proj\ebsilon.git\stage\x64\Deb... |
| id of property-set | with composition (fixed list) |
| t_ref | 0.5 °C |
| volume | 0.5 m³/kg |
| torque (just for fun) | 0.3687810605 lbf*ft |

Composition

| Material | mass-fraction |
|---|---|
| Nitrogen (N2) | 0.7900000000 |
| Oxygen (O2) | 0.1800000000 |
| Carbon dioxide (CO2) | 0.0250000000 |
| Argon (Ar) | 0.0050000000 |

## 1.3.3. VLMIX

The VLMIX property set extends the components available in FDBR with the ability to calculate vapor liquid equilibria. The methods used focus on robust calculations with reasonable speed at the cost of reduced accuracy.

It is also advised to be aware of the limitations of the methods used described below.

### 1.3.3.1. Thermal Properties

Thermal properties are either calculated with an ideal approach (pressure independent) or with a cubic equation of state (Peng-Robinson, Redlich-Kwong, Soave-Redlich-Kwong).

In the ideal approach for deriving the enthalpy and entropy of the liquid phase, a regression function for heat of vaporization was used.

$$H_{liq} = H_{gas} - dH_{vap}$$

$$S_{liq} = S_{gas} - \frac{dH_{vap}}{T}$$

The mixing rules use the classical linear approach.

### 1.3.3.2. Vapor / Liquid Equilibrium (VLE)

For VLE the following equation holds:

$$f_{i_{liq}} = f_{i_{gas}}$$

where $f_{i_{phase}}$ is the fugacity of the component *i* in the respective phase.

The fugacity of the liquid phase is calculated via Raoult's law:

$$f_{i_{liq}} = x_i * pvap_i$$

where $x_i$ is the molar fraction of componen *i* in the liquid phase and $pvap_i$ is the vapor pressure of the pure component *i* at the current temperature.

The fugacity of the gas phase is assumed to be the partial pressure defined in Dalton's law:

$$f_{i_{vap}} = pres_i = y_i * pres$$

$$\sum_i y_i = 1$$

$$\sum_i y_i * pres_i = pres$$

where $y_i$ is the molar fraction, $pres_i$ is the partial pressure of component *i* in the vapor phase and *pres* is the current pressure of the fluid.

The vapor pressure of each component is necessary for determining the VLE. For a few components, data for an extended Antoine equation were available. For the rest these were estimated as described below.

**1.3.3.3. Vapor Pressure Estimation**

The Antoine equation for the Vapor pressure has the following form:

$$ln(pvap_i) = A_i + \frac{B_i}{C_i + T}$$

Therefore, we need three data points to determine the parameters $A_i$, $B_i$ and $C_i$.

Candidates are the critical point, the triple point and the natural boiling point, which ensures that these points are exactly matched.

If we only have two data points, the following equation is used:

$$ln(pvap_i) = A_i + \frac{B_i}{T}$$

**1.3.3.4. Limitations**

The approach described can only determine simple VLE with only one liquid phase. Also, azeotropic behavior (e.g., ethanol-water) cannot be described and compositions near the azeotropic point will be off.

As the VLE calculations use the Antoine Equation for the Vapor pressure, which itself is only valid up to the critical point, this approach poorly describes the solubility of supercritical components in a liquid phase. In this case, an approach with Henry's law would be necessary, which is planned for a future release.

Additionally, the heat of vaporization for supercritical components when using the ideal gas formulation is not defined, and the liquid phase thermal properties of such a component are therefore off. Fortunately, the concentration of such components is low in the liquid phase.

For high pressures, when the compressibility factor $Z_i$ is away from 1, the method used for calculating the gas phase fugacities result in a bigger error for the phase split composition.

# 2. User Interface

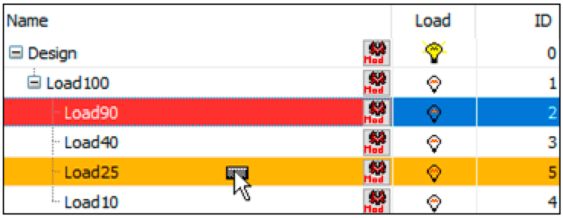
## 2.1. QT Carnot Factor Diagram: Display of the Exergy Losses

From Release 17 on, the exergy losses (corresponds to the SI unit of kW) can be displayed in the QT Carnot factor diagram. These losses are determined as integral area between the two curves ("cold" and "hot" side) of a heat exchanger. The display is activated / deactivated via the menu item "View" – "Show exergy loss values" in the Diagrams dialog. Activating the display is exclusively possible for the QT Carnot factor diagram type.

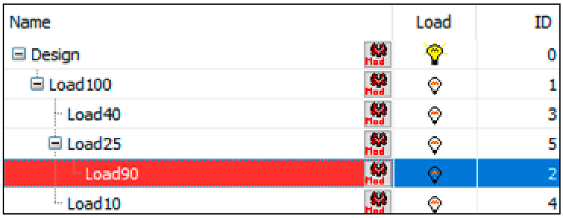
## 2.2. Profile Dialog Revised

The profile dialog now uses an improved control element for displaying the tree structure. This control element enables multi-selection by means of the mouse. Moreover, profiles can be shifted by drag&drop using the mouse or copied by additionally pressing CTRL and recursively copied with SHIFT+CTRL. Please note that the drop target (the element over which the moved elements are released) either

- is the new parent element (if no insertion points are displayed above or below the element)

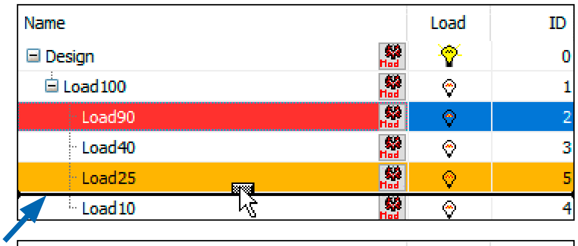


| Name | Load | ID |
|---|---|---|
| Design | | 0 |
| Load100 | | 1 |
| Load90 | | 2 |
| Load40 | | 3 |
| Load25 | | 5 |
| Load10 | | 4 |

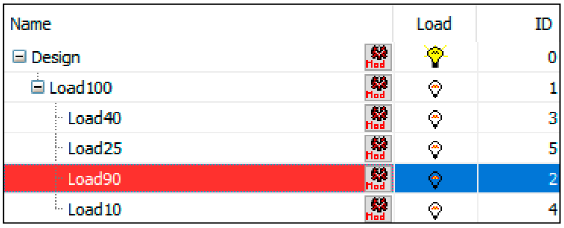


| Name | Load | ID |
|---|---|---|
| Design | | 0 |
| Load100 | | 1 |
| Load40 | | 3 |
| Load25 | | 5 |
| Load90 | | 2 |
| Load10 | | 4 |

- or (i.e., insertion points are displayed above or below the element) it is the predecessor /successor element.

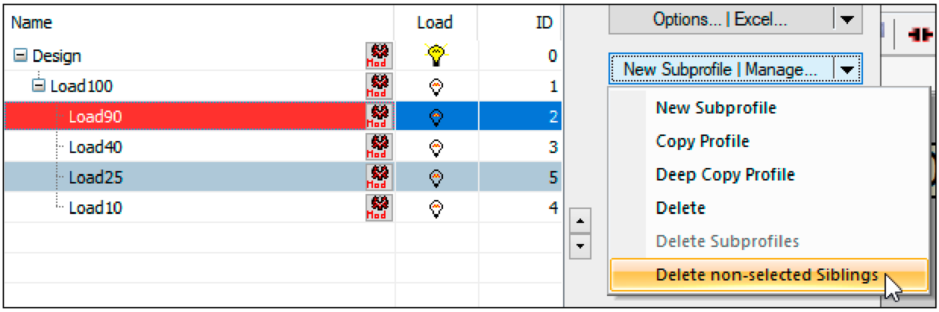


| Name | Load | ID |
|---|---|---|
| ⊟ Design | | 0 |
| ⊟ Load100 | | 1 |
| Load90 | | 2 |
| Load40 | | 3 |
| Load25 | | 5 |
| Load10 | | 4 |

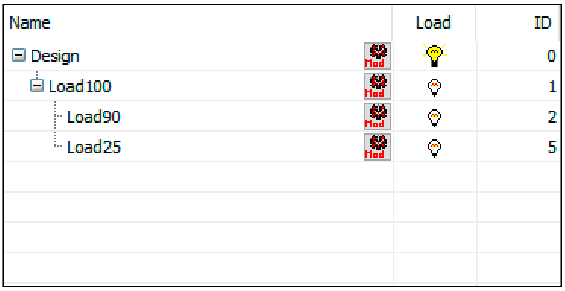


| Name | Load | ID |
|---|---|---|
| ⊟ Design | | 0 |
| ⊟ Load100 | | 1 |
| Load40 | | 3 |
| Load25 | | 5 |
| Load90 | | 2 |
| Load10 | | 4 |

There is now a function that deletes all non-selected sibling profiles, too:



| Name | Load | ID |
|---|---|---|
| ⊟ Design | | 0 |
| ⊟ Load100 | | 1 |
| Load90 | | 2 |
| Load40 | | 3 |
| Load25 | | 5 |
| Load10 | | 4 |



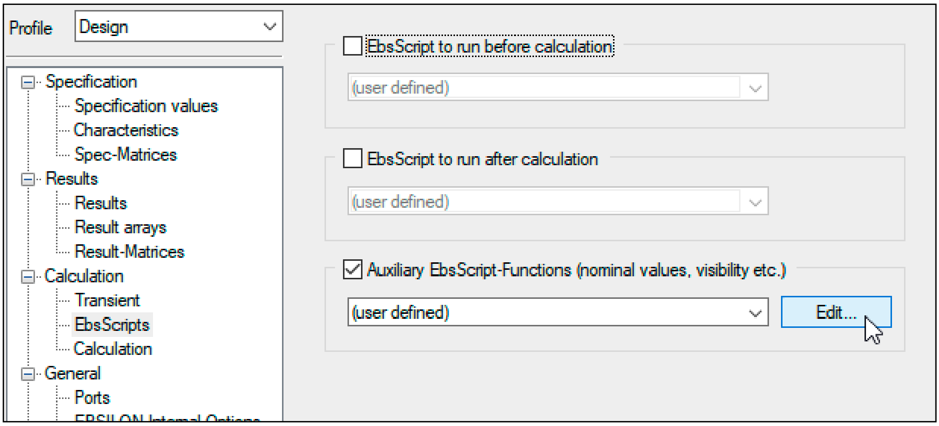
| Name | Load | ID |
|---|---|---|
| ⊟ Design | | 0 |
| ⊟ Load100 | | 1 |
| Load90 | | 2 |
| Load25 | | 5 |

The additional confirmation prompt before shifting or deleting profiles can be disabled by means of a checkbox.

## 2.3. Macro Objects: Dynamic Display of Specification and Result Values

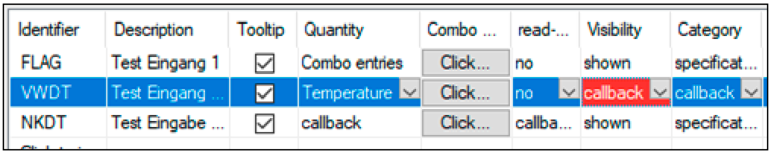
Up to now, a dynamic actualization of specification and result value was possible only via the XUI-Dll interface.

From Release 17 on, this is now also possible by an EbsScript-Code which is stored in the model. For this purpose, open the “Auxiliary EbsScript-Functions” in the Macro Interface:



and insert the requested code there.

Please note that additionally, in the definition of specification and result values, the corresponding property has to be set to “callback”:



| Identifier | Description | Tooltip | Quantity | Combo ... | read-... | Visibility | Category |
|---|---|---|---|---|---|---|---|
| FLAG | Test Eingang 1 | ☑ | Combo entries | Click... | no | shown | specificat... |
| VWDT | Test Eingang ... | ☑ | Temperature | Click... | no | callback | callback |
| NKDT | Test Eingabe ... | ☑ | callback | Click... | callba... | shown | specificat... |

The following functions are examples which may be used as template.
Of course, they have to be adapted to the respective model:

```
// dynamically change visibility
// will be called for every spec-/result-value with visibility set to “callback”
// return EbsValueVisibilityCallback to use default (see @System unit)
function getEbsValueVisibility(e:ebsvar):EbsValueVisibilityEnum;

begin
        // default behaviour for other values
        getEbsValueVisibility:=EbsValueVisibilityCallBack;

                if getEbsVarIdentifier(e) = “VWDT” then
                begin
                        if $.NKDT > 10 then
                        begin
                                getEbsValueVisibility:=EbsValueVisibilityGrayed;
                        end
                else
                        begin
                                getEbsValueVisibility:=EbsValueVisibilityShown;
                        end
                end;
end;

// dynamically change read-only
// will be called for every spec-/result-value with read-only set to “callback”
// return EbsValueReadonlyEnum to use default (see @System unit)
function getEbsValueReadOnly(e:ebsvar):EbsValueReadonlyEnum;

begin
        // default behaviour for other values
        getEbsValueReadOnly:=EbsValueReadonlyCallBack;

        // example code: changes spec-value “NKDT” to read-only, when FLAG is equal to 1
                if getEbsVarIdentifier(e) = “NKDT” then
                begin
                        if $.FLAG = 1 then
                        begin
                                getEbsValueReadOnly:=EbsValueReadonlyYes;
                        end
                else
                        begin
                                getEbsValueReadOnly:=EbsValueReadonlyNo;
                        end
                end;
end;
```

```
// dynamically change category
// will be called for every spec-/result-value with category set to “callback”
// return EbsValueCategoryEnum to use default (see @System unit)
function getEbsValueCategory(e:ebsvar):EbsValueCategoryEnum;

begin
    // default behaviour for other values
    getEbsValueCategory:=EbsValueCategoryCallBack;

    // example code: changes spec-value “VWDT” to nominal, when FLAG is equal to 1
        if getEbsVarIdentifier(e) = “VWDT” then
        begin
            if $.FLAG = 1 then
            begin
                getEbsValueCategory:=EbsValueCategoryNominalValue;
            end
        else
            begin
                getEbsValueCategory:=EbsValueCategorySpecificationValue;
            end
        end;
end;

// dynamically change dimension
// will be called for every spec-/result-value with category set to “callback”
// return DIM_ERROR to use default (see @System unit)
function getEbsValueDimension(e:ebsvar):DIM_Enum;

begin
    // default behaviour for other values
    getEbsValueDimension:=DIM_ERROR;

    // example code: changes spec-value “NKDT” to bar, when FLAG is equal to 4
        if getEbsVarIdentifier(e) = “NKDT” then
        begin
            if $.FLAG = 4 then
            begin
                getEbsValueDimension:=DIM_bar;
            end
        else
            begin
                getEbsValueDimension:=DIM_K;
            end
        end;
end;
```

# 3. Additional Modules

## 3.1. General

### 3.1.1. List of All Dimensions and Units

In the installation directory, in the subfolder "Docu", there is an HTML file called "dimensions_and_units.html". It contains a list of all dimensions used in EBSILON as well as the corresponding units. Dimensions and corresponding units are linked internally, and the conversion factors (and offsets respectively) used are listed.

## 3.2. EbsScript

### 3.2.1. Editor

Already during Release 16, the Editor received a new color scheme. In Release 17, the following changes and Extensions have now been added:

- The background of selected text is displayed in light blue; the foreground color remains unchanged.
- In the syntax options, it is possible to adjust the foreground and background color for the respective elements, and whether the text is to be displayed in bold.
- In the Editor menu, there is the entry "Edit -> View white space". With this option activated, blanks and tabs are displayed as visible characters.

### 3.2.2. New EbsVar functionality / functions

The following new functions are available for use with EbsScript variables / objects of type *ebsvar*:

- *Function setExpression(value:const ebsvar;*
  *expression:string):Boolean;*
  For the *ebsvar* value, sets the specified *expression*.
  Note: This function makes it possible to set an expression from EbsScript for specific/result values with dimensions "Text", "String" or similar.

- *Procedure copyEbsVar(source:const ebsvar; dest:const ebsvar;*
  *sourceProfileId:integer=-1; destProfileId:integer=-1);*
  Copies the contents of *source* to *dest*. If *sourceProfileId* or *destProfileId* is not specified or equal to -1, the current profile is used for the corresponding value. A call with *source* equal to *dest* is expressly permitted, for example to copy a specific/result value into another profile.

Furthermore, a direct assignment between the types *ebsvar* and *Variant* is now possible. This allows, for example, the value of an *ebsvar* to be saved and written back again (Attention: The evaluated value of the *ebsvar* is always saved. Any expressions are not saved.)

### 3.2.3. Hyperbolic and Area Functions

The mathematical functions of EbsScript have been extended by the hyperbolic functions

- *Function sinh(value:Real):Real;*
- *Function cosh(value:Real):Real;*
- *Function tanh(value:Real):Real;*

as well as their inverse functions (area functions)

- *Function asinh(value:Real):Real;*
- *Function acosh(value:Real):Real;*
- *Function atanh(value:Real):Real;*

### 3.2.4. “for-in” Loops

The *“for-in”* syntax can now be used for the iteration over arrays as well:

```
for x in arr do loop-instruction;
```

where *arr* is an expression of the type of an array (i.e., array with fixed size, dynamic array or open array) and the loop variable *x* is a variable that has the same type as the elements of the array.The loop is then performed once for each element in the array (in ascending order of the element index) an *x* is a copy of the array element with the current index in each case.

Informally, the *“for-in”* loop corresponds to the following construct:

```
for index:=low(arr) to high(arr) do
begin
        x:=arr[index];
        loop-instruction;
end;
```

The expression *arr* is evaluated **exactly once** (even if the loop is performed multiple times. For explanation: *arr* might e.g., be a function call that returns an array).

The key words *continue* and *break* can be used exactly as in index-based for-loops.

Alternatively, the following reference-binding syntax can be used:

*for **var** x in arr do loop-instruction;*

and

*for **const** x in arr do loop-instruction;* respectively.

In these cases, the loop variable *x* must be a new variable name in the local block. It behaves like a *var* and *const* parameter of a function respectively and represents a reference to the current array element.

In the case of *var*, the array element can also be changed by way of allocation to *x*.

Please note:
If *arr* is manipulated within the loop (with the exception of changing the values of the elements of the array), then the execution of **another** iteration step or the implicit checking of the condition of the loop abortion is **undefined behavior** (for instance, branching with break is admissible to prevent undefined behavior).
In the case of *"for **var**"* and *"for **const**"*, already accessing the loop variable after the manipulation is undefined behavior.

Examples:

| **Program** | **Output** |
|---|---|
| *var ar:array[1..3] of integer;*<br>        *x:integer;*<br>*begin*<br>        *ar:=[11,12,13];*<br>        *for x in ar do*<br>        *begin*<br>                *println(x);*<br>        *end;*<br>*end;* | 11<br>12<br>13 |
| *var x:integer;*<br>*begin*<br>        *for x in [11,12,13] do*<br>        *begin*<br>                *println(x);*<br>        *end;*<br>*end;* | 11<br>12<br>13 |
| *var ar:array[1..3] of integer;*<br>*begin*<br>        *ar:=[11,12,13];*<br>        *println("numbers");*<br>        *for var x in ar do*<br>        *begin*<br>                *println(x);*<br>                *x:=x*x;*<br>        *end;*<br>*println("squared in place");*<br>        *for const x in ar do*<br>        *begin*<br>                *println(x);*<br>        *end;*<br>*end;* | numbers<br>11<br>12<br>13<br><br>squared in place<br>121<br>144<br>169 |

## 3.2.5. Exceptions

EbsScript now also supports the triggering and interception of ***exceptions***. This is an option to represent code paths.

An exception is triggered with the command ***raise*** and the program control then moves to the ***except*** or ***finally*** handler of the ***try*** block entered last.

To intercept an exception, the triggering raise operation must be located within a ***try*** block or a function / procedure activated from there (i.e., try blocks protect any code executed between entering and leaving).
For instance, the following code

```
begin
        println(“before try block”);
        try
                println(“before raise”);
                raise Exception.Create(“Hello exceptional world!”);
                println(“after raise”);

        except on e:Exception do
                println(e.message);
        end;
        println(“after try block”);
end;
```

generates the output:

```
before try block
before raise
Hello exceptional world!
after try block
```

With *raise*, only objects that are additionally of the type *Exception* (in @System) or are derived from it can be thrown.

A try-except block has the following syntax:

```
try
        // protected code block
        Instructions; ...
except
        // exception handling code block
        Instructions; ...
end;
```

Here when triggering an exception, it is intercepted by the *except* block and its instructions are executed; subsequently, the program execution continues normally.

Alternatively, the following syntax can be used:

```
try
        // protected code block
        Instructions; ...
except
        on e1:<Exception-Type 1> do
                Instruction;
        on e2:<Exception-Type 2>do
                Instruction;
        on e3:<Exception-Type ...>do
                Instruction;
else
        // exception handling code block
        Instructions; ...
end;
```

Here when triggering an exception, the specified exception types are compared to the triggered one top-down. The **first type** (and only the first) that can be bound to the triggered exception (i.e., the exception has the same or a derived type) binds the exception to the specified variable and executes the instruction (which, of course, may also be a *begin*-instructions-*end* block).
If none of the specified types fits, the *else* block will be executed.
The specified exception variables can also be omitted (including the colon) in each case. In this case, no access to further information of the exception is possible.
The *else* block is optional too.
If none of the specified exception types fits and no *else* block exists, the exception will be forwarded to the next superordinate *try-except/try-finally* block.

If no such block exists, a possibly set ***unhandledExceptionFilter*** is activated or the program is terminated (a so-called ***uncaught exception***).
During the treatment of an exception treatment, the procedure *raise*; (without argument!) can be activated, which reactivates the exception just intercepted and forwards it to the next superordinate t*ry-except/try-finally* block (to the *raise*; instruction) (Please note: Subsequent handlers of the current *try* blocks are NOT considered anymore! Exception-/*finally* handlers are not regarded as part of the corresponding *try* block).

Another option to react to exceptions is by means of the *try-finally* block:

```
try
        // protected code block
        Instructions; ...
finally
        // code block that is ALWAYS executed
        Instructions; ...
end;
```

Here the code of the *finally* block is always executed. If an exception had been active at the beginning of the *finally* block, it is deactivated and automatically reactivated at the end of the block. *try-finally* blocks are suitable, e.g. for ensuring that external resources like files are closed correctly.

A *try* block with *except* and *finally* handler is not possible. However, such a behavior can easily be achieved by means of nested *try* blocks. For instance

```
try
        try
                // protected code block
                Instructions; ...
        finally
                // code block that is ALWAYS executed
                Instructions; ...
        end;
except
        // Exception handler
end;
```

Using *Try-Except-* and *Try-Finally* blocks respectively ONLY causes runtime costs when triggering exceptions. If no exception is triggered in such a block, no additional code will be executed during the runtime.

Warning: Exceptions should be used exclusively for the exception treatment in the event of errors or unexpected conditions. It is not advisable to use exceptions to, e.g., return valid results from deeply nested calls or to take shortcuts.

**Watch out, this is going to be VERY TECHNICAL:**
An exception triggered by means of *raise* is considered to be ***active***. During the treatment in an exception handler and *finally* block respectively, the (possibly) active exception is deactivated. It can be activated by means of *raise*; (without argument) or it becomes automatically active again at the end of the *finally* block (if an active exception exists at the entry).
If an exception leaves the range of a function/procedure when searching for a handler, its local varieties will be released (this is called ***stack unwinding***).

AND HERE IS THE LIMITATION: The program will be immediately **terminated** if

- another exception is triggered when executing the destructor due to a stack unwinding and it leaves the destructor
- or if during the execution of a *finally* handler at the beginning of which an active exception existed, another exception is triggered and it leaves the destructor.

Here the exception model of EbsScript is guided by that of C++.

Please note: When executing destructors and *finally* handlers, exceptions may be triggered, but usually those should be intercepted within this block. The function

```
function uncaughtException:integer;
```

defined in *@System* yields the number of active exceptions automatically deactivated in *finally* blocks, which allows to detect whether a newly triggered exception that leaves a destructor or *finally* block leads to an abortion or not.

In addition, there is the function

```
function currentException:Exception;
```

which returns the exception that is currently being handled – or *nil* if this does not exist.

Here is an example that produces several active exceptions at the same time:

```
procedure test(do_raise:integer);
begin
        if do_raise > 0 then
        begin
                raise Exception.Create("exception raised in test");
        end;
end;

var i:integer;
begin
        for i := 0 to 1 do
        begin
                // outer try-except
                try
                        // inner try-finally
                        try
                                println("i=", i);
                                test(i);
                        finally
                                println("running finally");
                                raise Exception.Create("exception raised in finally");
                        end;
                except
                        println("exception caught");
                end;
        end;
end.
```

Output:

```
i=0
running finally
exception caught
i=1
running finally

A runtime-error occured! Program execution stopped!    Time: 2/6/2024    10:24:38 AM

Exception: Runtime ERROR, more than one active exception! Terminated due to raising of an
exception during stack-unwinding (in “finally” or during class destruction)
```

### 3.2.6. Debugging

The Start-Debug button now allows to select the EbsScript to be started. To do so, please press the small dropdown arrow at the right edge of the button and select the desired script. Pressing the Start-Debug button will now open (if necessary) and start the script (and not the script active in the Editor). To always start the script active in the Editor, as previously, please select the entry “Active script”. The Start-without-Debugger button starts the same script as the Start-Debug button. The setting that defines which one is the start script is saved along with the model.

The list of the breakpoints is now saved along with the model.
Moreover, there is now a new breakpoint bar. In it, all breakpoints of the model are displayed. They can be activated, deactivated, and deleted there. When double-clicking on an entry, the respective EbsScript will be activated (if applicable, loaded) and the corresponding line will be jumped to.

### 3.2.7. EbsOpen Interface Unit

The interface unit @EbsOpen has been added. It allows to access the EbsOpen interface from EbsScript. This way, functions and properties of EBSILON and the model respectively may be used that are not directly accessible from EbsScript.

## 3.3. UserProps-Dll

See UserProps-Dll

# 4. Changes in the Results

## 4.1. Component 124: Result Values DP34ECO, DP34EVA, DP34SUP Removed

In Component 124, the result values DP34ECO, DP34EVA, and DP34SUP were not calculated on the basis of the surface area fractions in the flow path 3->4 (hot side) but on the basis of the surface areas in the flow path 1->2 (cold side). Due to a lack of relevance of these result values, they have been removed in Release 17.

# 5. Known Errors

## 5.1. Documentation

Unfortunately, it has not been possible to include all the new features of Release 17 into the Help. Please refer to these Release Notes until a Patch for the Help is available.

It is also possible to access the latest status of the Online Help available on the Internet by shifting the Help Preferences to “Launch in Browser”.

Iqony Solutions GmbH
Wetzbach 35
64673 Zwingenberg
Phone: +49 6251 1059-0
info@ebsilon.com
www.ebsilon.com