# EBSILON®*Professional*

Release Notes (English)

Release 16.0

steag
energy services

# Release Notes

# EBSILON®*Professional*

Release 16.0

## Contents

# 1. Calculation Kernel

## 1.1. New Components

### 1.1.1. Fuel Cell (Component 163)

Component 163 can be used to model the stack of a fuel cell of type SOFC (O--), PEM (H+) or AFC (OH-). It can use a manufacturer curve for a reference fluid or conductivities for the membrane. It calculates the ionic-flow, voltage, energy and mass balance over the membrane due to an imposed current.

### 1.1.2. Turbine (Map Based) (Component 164)

Component 164 can be used to model a map-based turbine, similar to the map based compressor (component 159). For this purpose, you need manufacturer data of the map.

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

The component 165 can be used to model a thermal regenerator (e.g. Cowper) / a bulk material storage. Thereby there is a difference between the internal storage and the shell. The shell may have cylinder shape or be a square channel. The shell geometry determines the overall storage volume in which the internal storage material is enclosed.
Please see Online Help for further details.

### 1.1.4. Phase-Change Material Storage (Component 166)

The component 166 models a **PCM** storage (here PCM stays for **phase change material**) including a heat exchanging part for charging / discharging the storage. It is important to clearly distinguish between the **PCM-Fluid** (the medium which changes its aggregate state and in which the latent heat is basically stored) and the **Fluid** (the working medium which flows through the heat exchanger charging / discharging the storage). Contrary to the working medium fluid the PCM-Fluid remains encapsulated within the storage. The working medium fluid flows through the component. In this way the component 166 is connected to the modelled cycle.
Please see Online Help for further details

### 1.1.5. Elektrolysis Cell (Component 167)

Component 167 can be used to model the stack of an electrolysis cell of type SOFC (O--), PEM (H+) or AFC (OH-). It can use a manufacturer curve for a reference fluid or conductivities for the membrane. It calculates the ionic-flow, voltage, energy and mass balance over the membrane due to an imposed current.

### 1.1.6. Quantity Converter (Component 168)

In logical constructions, this component serves to map quantities that are only available as result values or as derived quantities onto the three basic quantities mass flow, pressure, and enthalpy, which are used as variables in Ebsilon. This allows to access all these values during the calculation, e.g. for closed-loop controls or arithmetic operations.

The flags FMAPM, FMAPP, and FMAPH serve to set which input variable is to be mapped onto the mass flow / the pressure / the enthalpy of the outlet line. This mapping is always carried out in the Ebsilon default units: if e.g. a temperature is mapped onto a mass flow, 20°C is mapped onto 20 kg/s. In the result values RM, RP, and RH, however, the original dimensions are retained. There it would be possible to have 68°F displayed instead of 20°C.

The following input variables are available:

#### 1.1.6.1. Total and Static Quantities

As long as flow velocities are very small compared to sonic velocity, it is not necessary to differentiate between total and static quantities. For higher velocities, however, kinetic fractions are no longer negligible. Here it has to be noted that the quantities shown on the lines in Ebsilon are always total quantities, i.e. the kinetic fraction is always contained. This allows to allocate constant values to a line in Ebsilon, e.g. for the enthalpy, as the total enthalpy remains constant even if the pipe cross section and thus the flow velocity changes. The breakdown into static enthalpy and kinetic energy, by contrast, depends on the flow velocity at the respective location of the line.

Determining the static quantities therefore requires to know the flow velocity at the respective location of the line. Therefore it can be specified by means of Component 33 (General input value / start value) in Ebsilon; here it is possible to use several Components 33 with different velocities. The static quantities are then obtained as result values of the respective start value.

As the result values can only be further processed in EbsScripts, Component 168 now allows to transfer these quantities onto lines. FMAPx = 1 to 23 are available for this purpose. To facilitate the modeling of flight propulsion systems (which was the request that motivated the development of this component), FMAPx = 31 also allows to directly transfer the thrust (= mass flow * velocity).

#### 1.1.6.2. Masses, Energies ...

With FMAPx between 100 and 200, mass flows (absolute and molar) as well as energy flows (sensible, latent, total) can be displayed, and also the NCV, entropy, exergy, and geodetic height (the air pressure converted into height above sea level).

#### 1.1.6.3. Material Fractions

With FMAPx between 200 and 300, material fractions can be displayed. Here the substance whose fraction is to be displayed has to be selected in the respective FSUBSTx:

• 201 yields the mass fraction of the selected substance.
• 202 yields the molar fraction of the selected substance.
• 203 yields the emission load, i.e. the mass flow of the selected substance.

Moreover, an analysis according to chemical elements across all substances is possible, in the following variants:

• 204 yields the mass fraction of the desired element.
• 205 yields the mass flow of the desired element.

For this purpose, an individual element has to be selected at FSUBSTx, i.e. not a compound. In doing so, both conventional elements and NASA elements can be used. It is also possible to select “ash”, which then contains the fraction that does not consist of the conventional Ebsilon elements.

For instance: A binary fluid line with a water/LiBr mixture can conventionally only be broken down into $H_2O$ and ash. LiBr is treated as “ash” because it does not consist of conventional Ebsilon elements.

#### 1.1.6.4. Phase Information

301, 302, and 303 yield the fraction of the fluid that is in the gaseous, liquid, and solid phase respectively.

311 to 317 treat humid air:

- 311 the absolute humidity,
- 312 the relative humidity,
- 313 the saturation factor (which corresponds to the relative humidity for values <1 but can also become >1 if condensed water droplets are present),
- 314 the partial pressure of the water existing in the fluid,
- 315 the saturation pressure (i.e. at which partial pressure saturation would occur),
- 316 the wet bulb temperature (i.e. the temperature that a humid surface could achieve by evaporation),
- and 317 the dew point temperature (i.e. the temperature where water droplets initially form).

321 to 349 refer to a certain phase transition in each case:

- 321 to 329 refer to the phase transition solid / gaseous,
- 331 to 339 refer to the phase transition solid / liquid,
- 341 to 349 refer to the phase transition liquid / gaseous,

namely

- ..1 to the pressure and ..2 to the temperature of the beginning of the phase transition (when heating up)
- ..3 to the pressure and ..4 to the temperature at the end of the phase transition
- ..5 the enthalpy of the initial phase
- ..6 the enthalpy of the end phase
- ..7 the enthalpy difference between the two phases
- ..8 the current supercooling of the fluid up to the beginning of the phase transition
- ..9 the current superheating of the fluid since the end of the phase transition

351 and 352 yield the pressure and temperature of the triple point.

361 and 362 yield the pressure and temperature of the critical point.

#### 1.1.6.5. Physical Properties Calls

With FMAPx between 400 and 500, various physical properties functions can be displayed.

#### 1.1.6.6. Shafts and Electric Lines
FMAPx between 500 and 600 allow to access values of shafts and electric lines:

501 to 507 yield powers (real / idle/ apparent power) and quantities related to power like the power factor (cos(phi)), phase shift between current and voltage, and the number of phases.

511 to 512 yield rotational speed and frequency repectively and torque.

521 to 545 enable access to values regarding currents (521 to 525), voltages (531 to 535), and resistances (542 and 545). This information is partly determined by means of internal additional information that is not part of the equation system. Only 521 and 531 indicate the line value of the current and voltage respectively. FMAPx with the final digit 2 yields the real part and with the final number 3 the imaginary part of the respective quantity. With the final digit 4 you obtain the phase (related to the zero point "mass") and with the final digit 5 the amount. Normally, it should correspond to the line value. However, this additional information is not available for each electric line.

501 and 511 are available on shafts and electric lines, 512 only on shafts, and all others only on electric lines.

### 1.1.7. Biomass Gasifier (Component 169)
Component 169 is an extension of Component 96 which allows to define up to 5 tar substances and use kernel-expressions in all major input variables. The equilibrium calculations are also pressure dependent in contrary to component 96.

### 1.1.8. Header Adapter (Component 170)
PLEASE NOTE: This component is still under development and has not been completely tested. Over the course of the patches for Release 16, changes in the behavior of the component (e.g. new / other specification values, different behavior during the simulation, etc.) may occur.

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.

The header adapter ensures that the mass flow from Pin 3 to 4 is ALWAYS POSITIVE, i.e. 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 direction) or 2 -> 3 and 4 -> 1 are passed through (in the case of negative 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).

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 assumes the values of 1 and -1, depending on whether the current mass flow on the header is greater or equals 0 (value 1) or is negative (value -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.

## 1.2. Extensions of Existing Components

### 1.2.1. Boundary Value and Start Value (Components 1, 33, and 132)

#### 1.2.1.1. Specification Load Factor
There is now a specification value LOAD by which the value entered in the specification value M is multiplied in order to set the mass flow on a line.

This is helpful when you want to calculate different load cases in subprofiles. You do not have to calculate the mass flow in the respective profile then and only need to change M in the design profile when copying or changing the specification.

The default value for LOAD is 1.0, and as long as you leave LOAD at 1.0 in all profiles, everything will remain as before.

### 1.2.2. Steam Turbine (Component 6)

#### 1.2.2.1. FOUTETAP
As the calculation of the polytropic efficiency is very time-consuming (in some cases with complex physical properties, the total computing time of the model multiplied as a consequence), the result value ETAP is now no longer automatically calculated with each simulation. Just as previously in the case of the Compressor / Fan (Component 24), a flag FOUTETAP has now been implemented in Component 6 that allows to adjust whether ETAP is to be calculated or not.

### 1.2.3. Heat Extraction and Injection (Components 15 and 16)

#### 1.2.3.1. Options for Temperature Specification
In this component, the heat output extracted and injected respectively had to be specified on the logic pin before. The outlet temperature was then calculated from the heat balance.

To set a particular temperature at the outlet, a controller had to be used until now. Now this is no longer necessary as new options for temperature specification have been implemented. These are selected via the new flag FT:

FT = 0: T2 is calculated from the heat balance (as before).
FT = 1: T2 is defined by specification value T2SET.
FT = -1: T2 is defined on the outlet line.

In the cases FT=1 and FT=-1, the dissipated heat quantity and the required heat quantity respectively is put on the logic line as enthalpy.

As negative values are admissible on this logic line, the heat extraction can also be used as heat injection and vice versa.

### 1.2.4. Combustion Chamber with Heat Output (Component 21)

#### 1.2.4.1. Result Value QLCO

If a portion of the carbon is not completely oxidized into $CO_2$ but only CO, the latent heat existing in the carbon is not fully utilized. This loss is now separately shown in result value QLCO.
The loss due to CO has been contained in the result value QLNB so far. This has not been changed. Please note: The amount of CO formed is determined by the specification values COCON and ECOCON respectively. The combustion efficiency ETABN, however, must be selected small enough to enable the losses due to CO. Otherwise a warning message will be output and the NCV of the flue gas will be reduced accordingly in order to comply with the energy balance. This reduction is also contained in result value QLNB. The result value QLCO, however, contains the unchanged value. If the warning occurs, ETABN must be reduced until QLNB >= QLCO.

### 1.2.5. Value Indicator (Component 45)

#### 1.2.5.1. Freezing Temperature and Pressure

By analogy to FTYP = 50 (Boiling temperature) and 51 (Boiling pressure), this value indicator now offers two new FTYP variants for the freezing point:

- For the pressure prevailing on the line, FTYP = 93 yields the temperature where the the liquid starts freezing when cooling down. Below the triple pressure, the temperature where the gas begins to resublimate (i.e. to enter the solid phase) is provided.
- For the temperature prevailing on the line, FTYP = 94 yields the pressure where the transition to the solid phase begins. Depending on the fluid, this can happen at rising or falling pressure; while $CO_2$ solidifies at rising pressure, liquid water freezes when the pressure decreases.

#### 1.2.5.2. Melting Temperature and Pressure

For pure substances like e.g. water, the freezing point and melting point are the same. For mixtures, however, the melting point (beginning of the transition from the solid to the liquid phase when heating up) and freezing point (beginning of the transition from the liquid to the solid phase when cooling down) may be different. Therefore two further FTYP variants have been implemented for this purpose:

- For the pressure prevailing on the line, FTYP = 95 yields the temperature where the solid matter starts melting when heating up. Below the triple pressure, the temperature where the solid matter begins to sublimate (i.e. to enter the gaseous phase) is provided.
- For the temperature prevailing on the line, FTYP = 96 yields the pressure where the transition from the solid to the liquid phase begins.

### 1.2.5.3. Specific Gas Constant

FTYP = 97 yields the specific gas constant that is defined by

RS = R / MOLM

where R is the universal gas constant and MOLM is the molar mass.

### 1.2.5.4. Real Gas Factor

FTYP = 98 yields the real gas factor (also called compressibility factor) that is defined by

Z = (P * V) / (RS * T)

with P = pressure, V = specific volume, RS = specific gas constant, T = temperature[K]. Z = 1 for an ideal gas. The real gas factor therefore allows to estimate how accurate a calculation as an ideal gas is.

### 1.2.5.5. Normalized Volume Flow According to Ideal Gas Approximation

Previously, at FTYP = 52 the volume flow in the operating point was converted to normalized conditions according to the formula

V/VNORM = (T/TNORM) / (P/PNORM)

This, however, led to undesired results when the volume flow in the operating point was not calculated in ideal gas approximation but e.g. a real gas correction was used.

Therefore the calculation is now carried out independently of the physical properties in the operating point, i.e. the normalized volume flow is now always the volume flow that would flow through the line if the fluid was an ideal gas at normalized conditions:

VNORM = Ri * TNORM / PNORM,

where Ri is the specific gas constant with Ri = R/$m_{mol}$.

R is the universal gas constant R = 8.31441 kJ/(kmol*K) and mmol is the molar mass of the fluid.

The result then no longer depends on the operating point but only on the composition of the fluid.

Solid matters existing in the fluid are neglected. Whether water is to be included in the calculation or not can be adjusted via the flag FNORMW, as before, just as the possible conversion to a certain $O_2$ content.

The modification regarding FTYP = 52 also applies to Component 46 (Measured value input).

### 1.2.5.6. Zusammensetzung der Phasen

The types FTYP = 53 and FTYP = 90 (composition of the phases (mass fractions)) now behave identically. The index before the respective substance name in the listing corresponds to that index in the list of all available substances (see e.g. list of default value FSUBST).

The same applies to the types FTYP = 54 and FTYP = 91 (composition of the phases (mole fractions)).

### 1.2.6. Selective Splitter (Component 52)

#### 1.2.6.1. Substance-Specific Kernel Expression

For this component, the Kernel expression EADAPT could only be used as a common correction factor for all separation rates.

Now there is a new setting FADAPT = -2 where EADAPT is no longer interpreted as a correction factor but directly as a separation rate. Here values < 0 or > 1 are cut down to 0 and 1 respectively.

To allow to specify different separation rates for the individual substances, a KE_Internal-Variable subst_id has been implemented which can be used when programming the Kernel expression. The component evaluates the Kernel expression in a loop for each substance existing on the inlet line and, in doing so, hands over the substance currently under consideration in subst_id.

If, for instance, you want to separate half of the water and all of the $CO_2$ and all of the $SO_2$, you can use the following code:

```
function evalexpr:REAL;

var val:real;
        internals:array of InternalValue;
        n:integer;
        i:integer;
        subst_id:integer;

begin
        internals := keGetInternals();
        n := length( internals );
        if (n > 0) then subst_id :=internals[0].value;

        if (subst_id = 4) then // H2O
        begin
                val:=0.5;
        end
        else if (subst_id = 3 or subst_id = 5) then // CO2 and SO2
        begin
                val:=1.0;
        end
        else
        begin
                val:=0.0;
        end;
        evalexpr := val;
end;
```

#### 1.2.6.2. Separation with Defined Outlet Composition

To facilitate the modeling of chemical processes, the modes FM = 4 and FM = 5 have been implemented to allow to define the ratio of the substances at the outlet. Here the specification values J.. are not interpreted as separation rates but as the relative ratio of the substances among each other that is to be achieved on the outlet line. At FM = 4, the mass fractions are specified here, and at FM = 5 the molar fractions.

The separation quantity is defined by the specification value RR (reaction rate). For RR = 1, as much as possible is separated, i.e. so much that nothing is left of (at least) one substance.

### 1.2.7. Heat Exchanger (Component 61)

#### 1.2.7.1. Determining the Heat Transfer Coefficients in the Design Case

To design a heat exchanger, optionally specific temperatures or temperature differences (FSPECD = 1 to 5) or the effectivity EFFN (FSPECD = 0) or the heat exchanger surface area AN (FSPECD = 9) are specified to Ebsilon. From this, Ebsilon determines the characteristic value KA, i.e. the product of the heat transfer coefficient k and the surface area A, in the design calculation.

Here the heat transfer coefficient was always calculated from the two heat transfer coefficients AL12N and AL34N for the cold and hot side respectively, which had to be specified by the user. At FSPECD = 0 to 5, an inaccuracy in the specification of AL12N and AL34N has no effect on the heat exchange in the design case as it only depends on the product k*A. To determine the heat exchanger surface area AN from k*A, however, and also for the off-design behavior, a precise knowledge of k is required, and at FSPECD=9 also in the design case.

Release 16 now allows to have the heat transfer coefficients determined by Ebsilon. This is effected by means of the new flag FAA:

- FAA = 0 determines AL12N and AL34N and also the corresponding off-design exponents EX12 and EX34 on the basis of the fluids connected to the component:
    - If the respective fluid combination is contained in the database, the data of this combination will be used.
    - If the respective fluid combination is not contained in the database but both fluids are, average values will be formed for each fluid that this fluid has in the existing combinations. The user will be notified about this in a comment.
    - If only one of the fluids is available in the database, the database values will be used for this fluid and the specification values will be used for the other fluid. The user will be notified about this in a comment.
    - If none of the fluids is available in the database, the specification values will be used for both fluids. In this case, a warning message will be output because the mode FAA = 0 has no effect in this case.

  This mode is the default setting for newly inserted heat exchangers.

- FAA = 1 uses the specification values AL12N, AL34N, EX12, and EX34. This corresponds to the previous behavior.
  Therefore heat exchangers created with an older version of Ebsilon are automatically set to this mode so that the results remain unchanged.

- FAA = 2 can be used for calculating the heat transfer coefficient AL12N, provided that the surface area AN is known. However, this option is only available at FSPECD = 0 to 5 as another specification value is required in addition to the surface area. As far as possible, AL34N and both exponents are taken from the database.
- FAA = 3 can be used for calculating the heat transfer coefficient AL34N, provided that the surface area AN is known. However, this option is only available at FSPECD = 0 to 5 as another specification value is required in addition to the surface area. As far as possible, AL12N and both exponents are taken from the database.
- FAA = 4 can be used for calculating the heat transfer coefficient AL12N, provided that the surface area AN is known. However, this option is only available at FSPECD = 0 to 5 as another specification value is required in addition to the surface area. Unlike at FAA = 2, at FAA = 4 the specification values are used for AL34N and the two exponents.
- FAA = 5 can be used for calculating the heat transfer coefficient AL34N, provided that the surface area AN is known. However, this option is only available at FSPECD = 0 to 5 as another specification value is required in addition to the surface area. Unlike at FAA = 3, at FAA = 5 the specification values are used for AL12N and the two exponents.

#### 1.2.7.2. Expansion by Transient Calculation

From Release 16 on, Component 61 can be used in transient calculations too. However, a transient calculation with Component 61 is only possible in off-design mode.

You can choose between two approaches for the parameterization of the component:

- The simplified approach with the specification of the ratio of the heat exchange surface area to the mass of the pipes (specification value ATM) and the ratio of the heat exchange surface area to the internal volume of the pipes (specification value ATV)
- The more accurate approach with the specification of the pipe parameters and, if applicable, of the fin parameters

In both cases, the surface area (AN) determined in the steady-state simulation is used to determine the thermal mass of the pipes and the pipe volume for the transient calculation.

Component 61 uses the combined physical and numerical calculation model of Component 126. A flag FINST is used to switch over between the steady-state calculation (as previously) and the transient calculation. Please refer to the EBSILON Online Help for further details.

### 1.2.8. Gas Turbine (Component 106)

- Additional water / steam ports (11,12) are added in order to allow to model the effects of fogging, power augmentation and water needed for NOx reduction
- An additional heat rejection port (13) is added to model a second heat rejection source
- Extension of the library dataset to save information on the maximum allowed $H_2$ volume fraction in the fuel and allow to filter the dataset-list for it.

Additional result variables:

- RALAM: Air ratio
- RGTCSET: gas turbine curve set used in calculations
- RQCOOL2: 2nd heat rejection duty

### 1.2.9. Indirect Storage (Component 119)

PLEASE NOTE: The expansion for Component 119 described here is still under development and has not been completely tested. Over the course of the patches for Release 16, changes in the behavior of the expansion described here may occur.

In Component 119, the following new values are available for the specification value "FDIR":

2: Forward flow  
3: Backward flow  
4: According to mass flow at Pin 4

In addition, it also has logic pin (4) to display the direction at FDIR = 4 (mass flow greater than or equal to 0, then in forward direction; in the case of negative mass flow in backward direction).

In contrast to FDIR = 1 (switched over according to the previous time step), the values 2, 3, and 4 always state the absolute flow direction in the component. Furthermore, the result fields and matrices regarding the absolute component orientation are displayed at FDIR = 2, 3 or 4. This means that if e.g. the cold Component 119 is flown through backwards (FDIR = 3) with a hot fluid, then the temperature matrix will usually display higher temperatures in the area on the right after the simulation.

These new values and Pin 4 can e.g. be used in connection with Component 170 (Header adapter).

Additionally, the component possesses a logic pin (5) on which the mass flow assumes the values 1 and -1, depending on whether the current mass flow on the header is greater than or equal to 0 (value 1) or is negative (value -1).

This information can e.g. be forwarded to Pin 4 of storage component 119 to inform it about the flow direction.

### 1.2.10. Steam Turbine (Component 122)

- Stodola correction for small number of stages is added (FP1OD = 3).
- Matrix MXPCRIT is added to allow to model the critical pressure ratio for the Stodola relation
- Additional exhaust loss method (FSECT = 7) added (refined ELEP calculations)
- AEN method as alternative to the Stodola flow-inlet pressure relation is added (FP1OD = 4). PB = crit_flux(PB,HB)*AEN
- Additional FIDENT modes (11,12,13) added which give feedback on logic port 11
- FGSOD=3 added which allows the valve point on port 11 P (FIDENT = 0 only)
- FGSNOZZLETYPE added for distinguishing between converging and converging-diverging nozzles. Up to version 15 only converging Nozzles were considered.
- Performance factors can be values, characteristic lines, values on port 11, expressions and a combination of all
- Allow non-$H_2O$ fluids. This disables SCC methods, which are regression formulas, based on measured steam turbine data. Depending on the property set used, long calculation times might occur.

### 1.2.11. Shaft Seal (Component 123)

- Allow reverse flow, when Pout > Pin (FDIR = 1)

### 1.2.12. Heat Exchanger with Phase Transition (Component 124)

#### 1.2.12.1. Specification of the Heat Transfer Coefficients for the Numerical Solution for Mixtures

Component 124 uses various specification values for specifying the heat transfer coefficients. In the case of fluids of the type water-steam or 2-phase fluid, it is possible to determine the state of phase of the entire fluid. Therefore if FTYPHX > 0, the specification values AL12ECON, AL34ECON, AL12EVAN, AL34EVAN, AL12SUPN, AL34SUPN and the corresponding exponents will be used for these fluids.

For the other fluids (mixtures), independently of the flag FTYPHX it was only possible to use the specification values AL12N, AL34N and the corresponding exponents. From Release 16 on, the values

- AL12PHTN
- EX12PHT
- AL34PHTN
- EX34PHT

are provided for such fluids. If at FTYPHX > 0 a component of the mixture undergoes a phase change within the heat exchanger (numerical solution FALG = 1), these specification values will be used instead of AL12N, EX12, AL34N, EX34. This may happen e.g. in the case of the condensation of the water fraction in the gas.

#### 1.2.12.2. Alternative Specification of the Heat Transfer Coefficients for the Numerical Solution via Kernel expressions

Alternatively to the specifications of the nominal values and exponents, the heat transfer coefficients within the heat exchanger (numerical solution FALG = 1) can be calculated by means of the Kernel expressions

- EALPH12
- EALPH34

To differentiate between the specification of the nominal values, exponents, and the Kernel expressions, the flags

- FALPH12
- FALPH34

are provided from Release 16 on.

### 1.2.13. Air cooled Condenser (Component 127)

- Distinguish between forced and induced draft (FAIRPATH). Before Version 16, only a forced draft ACC was modeled.
- Characteristic CWINDCORR added. This enables to model the influence of wind on the air flow.

### 1.2.14. Transient Separator (Component 131)

#### 1.2.14.1. New Calculation Mode FFU = 2

Previously, Component 131 could only calculate an output signal (Pin 2) from an input signal (Pin 1). The new mode FFU = 2 now allows to calculate the input signal from the known output signal on the basis of the same equations. This can be interesting if e.g. a time-attenuated measurement signal is available and you want to calculate the chronological sequence of an unattenuated value from it. The attenuation can e.g. be caused by a thick plating of a measurement sensor.

### 1.2.15. Stratified Storage (Component 145)

#### 1.2.15.1. Additional Pins 3, 4 for the Fluid

From Release 16 on, Component 145 has new pins for the fluid. Up to Release 16, Pins 1 (fluid inlet), 2 (fluid outlet), and 3 (logic pin for the amount of heat exchanged during the time step) were available. Pin 1 was intended for discharging in the lower (“cold”) part of the storage system and for charging in the upper (“hot”) part of the storage system. Pin 2 was intended for discharging in the upper (“hot”) part of the storage system and for charging in the lower (“cold”) part of the storage system.

From Release 16 on, the pins are as follows:

- Pin 1 – Fluid inlet when discharging the storage system
- Pin 2 – Fluid outlet when discharging the storage system
- Pin 3 – Fluid inlet when charging the storage system
- Pin 4 – Fluid outlet when charging the storage system
- Pin 5 – Logic pin for the amount of heat exchanged during the time step (former Pin 3)

The flag FCHARGE has a new value 2 to allow to calculate with the new pins. To calculate with the old logic, you can either work with FCHARGE = 0 or with FCHARGE = 1. Existing models do not need to be adapted.

### 1.2.16. Reciprocating Engine (Component 153)

- Extension of the library dataset to save information on the maximum allowed $H_2$ volume fraction in the fuel and allow to filter the dataset-list for it.

Additional result variables:

- RALAM: Air ratio

### 1.2.17. Map based compressor (Component 159)

- Polytropic head method for efficiency added.
- Flag to take compressibility into consideration, when calculating corrected flow and corrected speed (F_USE_Z_REF)
- Flag for shaft speed handling in IGV mode (FY = 3) added

### 1.2.18. Flag FDPNUM: Handling the Pressure Drop in the Numerical Calculation in Components 7, 10, 61, 119, 124, 126

For the numerical calculation (transient calculation in Components 7, 10, 61, 119, 126 as well as calculation at FALG = 1 in Component 124), the mean value of the fluid pressure between inlet and outlet was used up to Release 15. From Release 16 on, this calculation is refined. For this purpose, a new flag FDPNUM has been implemented in the above-mentioned components.

At FDPNUM = 0, the calculation is carried out with a mean value of the fluid pressure between inlet and outlet, as before.

At FDPNUM = 1, however, a linear pressure distribution between inlet and outlet is assumed and the calculation is carried out with corresponding pressure values in the individual fluid elements (number controlled via NFLOW).

## 1.3. Material Properties

### 1.3.1. Effective Material Properties Functions for Phase Smoothing

When modeling the latent heat storage system, it has proven reasonable to develop effective properties functions that have a smoothed behavior at the phase transition as singularities at the phase transition point cause numerical problems and a special treatment of such cases should be avoided.

Outside a small range around the phase transition point (0.05 K up or down), these functions coincide with the real properties functions. Within this range, there is a smoothed transition from one phase to the other.

The following effective properties functions are available:

- heff (p, T)
- cpeff (p,T)
- rhoeff (p,T)
- lambdaeff (p,T)
- xeff (p,T)
- phaseeff (p,T)

In addition, there are the functions

- heff (p, x1, x2): phase transition enthalpy between the state x1 and the state x2
- teff (p,x): yields the effective temperature associated with the state x. For the phase transition solid (x = 10) / liquid (x = 11), for instance, for x = 10.5 you receive the actual phase transition temperature, for x = 10.0 the lower and for x = 11.0 the upper limit of the smoothing range.

### 1.3.2. PCM Fluids

PCM (phase change material) fluids are substances which store or give off the supplied or released energy by means of a phase change. Energy is needed for the phase change. Due to this, a large part of the energy exists as latent heat and the operation temperature ranges become smaller compared to sensible heat storage systems. The substances described here undergo a phase transition from solid to liquid (melting) or from liquid to solid (freezing). The phase change takes place within a melting interval. To be able to carry out calculations in the two-phase range, an effective specific heat capacity is defined so that the integral over the melting interval corresponds to the effective specific melting enthalpy (sum of melting enthalpy and sensible enthalpy):

$$\Delta h_{eff} = \Delta h_{mel} + \Delta h_{Sens} = \int_{T_0}^{T_1} (c_{p,eff}(T)dT)$$

In Ebsilon, the PCM fluids can be found among the "oil / melt" fluids. Four predefined fluids are available altogether. Moreover, it is possible to create a user-defined fluid on the basis of a JSON format. Although the pressure has to be formally specified when activating the physical properties functions, the calculations exclusively depend on the temperature. The following physical properties functions are available:

- h(p,T)
- h(p,s)
- s(p,T)
- s(p,h)
- t(p,h)
- t(p,s)
- x(p,T)
- x(p,h)
- cp(p,T)
- cp(p,h)
- v(p,T)
- v(p,h)
- eta(p,T)
- eta(p,h)
- lambda(p,T)
- lambda(p,h)
- rho(p,T)
- rho(p,h)
- nue(p,T)
- nue(p,h)
- t_min(p)
- t_max(p)
- t_freeze(p)
- t_melt(p)
- p_min(T)
- p_max(T)
- h_freeze(T)
- h_melt(T)

#### 1.3.2.1 Predefined Fluids

The predefined fluids of the manufacturer Rubitherm are substances with an effective melting interval of about 15K, where the mean temperature roughly corresponds to the number in the brand name of the PCM. In addition, there are lower and upper melting interval temperatures, designated as t_trans_lower and t_trans_upper in the JSON interface, and the melting temperature t_melt and the solidifying temperature t_freeze. The melting interval temperatures indicate the limits of the temperature range in which the effective specific heat capacity contains the fractions of the melting enthalpy (see above) and thus rises. In the melting interval, the PCM fluid undergoes a phase transition from solid to liquid. When the melting temperature t_melt is exceeded, it is assumed that the PCM fluid is in the liquid state (important e.g. when considering the convection in Component 166). When the solidifying temperature t_freeze is fallen below, it is assumed that the PCM fluid is in the solid state.

In what follows, the most important properties of the fluids are summarized:

- RT44HC: paraffin, operation temperatures 25-70°C, effective melting interval 37-52°C, effective specific melting enthalpy 250.51 kJ/kg, melting temperature 41°C, solidifying temperature 44°C
- RT64HC: paraffin, operation temperatures 25-95°C, effective melting interval 57-72°C, effective specific melting enthalpy 243.66 kJ/kg, melting temperature 64°C, solidifying temperature 65°C
- RT80HC: fatty acid, operation temperatures 25-110°C, effective melting interval 73-88°C, effective specific melting enthalpy 198.11 kJ/kg, melting temperature 77°C, solidifying temperature 80°C
- SP58p: salt hydrate (sodium acetate trihydrate), operation temperatures 25-80°C, effective melting interval 51-66°C, effective specific melting enthalpy 242.42 kJ/kg, melting temperature 58°C, solidifying temperature 58°C

#### 1.3.2.2 User-Defined Fluids

Please note: All predefined fluids are already available in the required JSON format and can be used as templates for user-defined fluids by copy-and-paste.

The user-defined PCMs are created by means of the JSON format. The following details are required for the full functionality of the PCMs:

- Name (“name”)
- Minimum and maximum operation temperatures (“t_min” and “t_max”)
- Melting temperature “t_melt” and solidifying temperature “t_freeze” (solidus and liquidus temperature), used e.g. when considering the convection in Component 166
- The limits of the effective melting interval “t_trans_lower” and “t_trans_upper”, required e.g. in the context of “region_polynomial” and “region_cp_with_h_eff”
- Specific heat capacity (“cp”)
- Density (“rho”)
- Thermal conductivity (“lambda”)
- Dynamic viscosity (“eta”)

Moreover, further optional details are available:

- Minimum and maximum operation pressures (“p_min” and “p_max”)
    - set to 0.001-1000 bar by default
- Zero point definition by pressure, temperature, specific entropy and enthalpy
    - “t0”, “p0”, “h0”, and “s0”
    - Set to 0 kJ/kg and 0 kJ/(kgK) at 1 bar and 25°C by default

Most characteristic values are described by means of primitive types. The material properties heat capacity, density, thermal conductivity, and dynamic viscosity have to be defined by various more complex JSON objects, which in turn presuppose one of the predefined types. In what follows, these types are listed and the specifications for the JSON format are described:

- „step_points” → depend on the physical material properties
    - Heat capacity: piecewise constant function; points define the beginning of the constant value that is valid up to the next point
        - Advantage: enthalpy becomes piecewise linear function
    - Other physical material properties: piecewise linear function between the initial point and the terminal point

        - Definied as an array of points that, in turn, are arrays
- "polynomial"
    - The entire operation temperature range is defined by an individual polynomial of any degree
    - Defined as an array of the coefficients of the polynomial, beginning with the smallest coefficient $a_0$ (absolute term)
- "region_polynomial"
    - The solid and the liquid phase are described with an own polynomial in each case; the melting interval is interpolated in a linear way
    - Defined as an object with the properties "liquid" and "solid", which are assigned to the respective arrays of the polynomial coefficients
- "piecewise_polynomial"
    - The entire operation temperature range is defined by any number of polynomials
    - The entire operation temperature range must be covered and the polynomials must be specified sorted according to the definition range
    - Consists of an array of the piecewise defined polynomials
    - Polynomials defined as objects with upper and lower limit of the definition range ("t_min" and "t_max") as well as the array of the polynomial coefficients ("polynomial")
- "expression"
    - Definition of a random temperature-dependent function by means of all common functions and operators by a simple string
    - Use of the conditional operator for the definition of different functions as a function of the temperature range
        - condition ? true case : false case
- "region_cp_with_h_eff" → only available for the heat capacity
    - Solid and liquid phase defined by one of the previous types (except "region_polynomial")
        - „solid" and „liquid" as objects with the respective type
    - The behavior in the two-phase range is calculated by specifying the effective specific melting enthalpy ("h_eff")
        - Function in the two-phase range: $c_p\,(T) = a + b \cdot (1\text{-}tanh^2(c \cdot T + d)\,)$
        - *c* can be specified by the user (by default 0.26)
        - The remaining coefficients are determined from the boundary conditions
            - → $\Delta h_{\text{eff}}$ adhered to and
            - → continuity of the function at both margins

Below is a syntactically correct definition of a PCM fluid where each of the existing types is used. **This is only an example of the different types and not a real fluid for calculations.**

```
{
        “name” : “MyFluid”,
        “p_min” : 0.1,
        “t_min” : 20,
        “t_max” : 60,
        “t_melt” : 38,
        “t_freeze” : 38,
        “t_trans_lower” : 35,
        “t_trans_upper” : 45,
        “t0” : 35,
        “p0” : 1,
        “h0” : 1000,
        “s0” : 0.8,
        “cp” : {
                “region_cp_with_h_eff” : {
                        “h_eff” : 250,
                        “c” : 0.4,
                        “liquid” : {
                                “step_points” : [
                                        [45, 1.6],
                                        [50, 1.7],
                                        [60, 1.8]
                                ]
                        },
                        “solid” : {
                                “expression” : “t>30 ? 1.5E-2*t : sin(t+3)”
                        }
                }
        },
        “rho” : {
                “polynomial” : [1700, 3.5, 4E-06]
        },
        “lambda” : {
                “region_polynomial” : {
                        “liquid” : [0.2],
                        “solid” : [0.1, 0.004]
                }
        },
        “eta” : {
                “piecewise_polynomial” : [
                        {
                                “t_min” : 45,
                                “t_max” : 55,
                                “polynomial” : [30]
                        },
                        {
                                “t_min” : 55,
                                        “t_max” : 60,
                                        “polynomial” : [40]
                                }
                        ]
                }
}
```

### 1.3.3. Helisol Fluids

The thermo oils Helisol 5A, Helisol XA, and Helisol XLP of the company Wacker have been included into the standard database, in each case in an unused and a used (i.e. after 750 hours of operation) variant and in each case for the four pressure stages 1 bar, 10 bar, 20 bar, and 30 bar.

In contrast to the thermo oils integrated so far, the Helisol data are not interpolated with polynomials but with characteristic lines.

In the calculation, only the dependence on the temperature is considered but not the dependence on the pressure. This can only be be considered by selecting the most suitable data record. A plausibility check regarding the line pressure does not take place.

### 1.3.4. TREND Becomes TREND 4. New: TREND 5

The line type “Universal fluid” allows to use the property library “TREND” of Ruhr-Universität Bochum. Here Ebsilon 15 supports Version 4. In Ebsilon 16, Version 5 is supported in addition. To differentiate between the two, the library “TREND” has been renamed to “TREND 4”. The library “TREND 5” has been added.

TREND 5 supports the following additional substances:

- 1-Hexene (hexylene)
- POE5 (pentaerythritol tetrapentanoate)
- POE7 (pentaerythritol tetraheptanoate)
- POE9 (pentaerythritol tetranonanoate)

In models saved with versions prior to Release 16, “TREND” libraries are loaded as “TREND 4”. When changing from TREND 4 to 5 and back, all compositions and options are retained to the greatest degree possible (see also Section 1.3.4).

### 1.3.5. Library Change for Universal Fluid

When a library is changed in the case of the line type “Universal fluid”, the composition is now retained (of course, only for substance fractions that are available in the new library).

### 1.3.6. New Options for the Composition Definition

In the case of the line types air, flue gas, oil, coal, crude gas, and user-defined cp (also known as FDBR line types), it is now possible to use each of the composition definitions for each line type.

In addition, there are four new composition definitions:

• Mass without solid matters
• Mol without solid matters
• Dry mass without solid matters
• Dry mol without solid matters

Here solid matters are the following substances: C, H, O, N, S, CL, ASH, LIME, CA, H2OB, ASHG, MG, CACO3, CAO, CASO4, MGCO3, and MGO.

The solid matters are specified / displayed as partial mass balance and the remaining substances are scaled (without $H_2O$ if applicable) to 100 percent as a mixture (as mass composition and molar composition respectively).

Of course, these new options can also be used when representing compositions in value crosses and in calls in EbsScript and EbsOpen.

# 2. User Interface

## 2.1. New Data Types for EbsMatrix

Previously, in objects of the type EbsMatrix (specification and result matrices respectively), only floating point numerical values (data type “double”) could be saved.

User-defined matrices in macro objects can now additionally have one of the following data types as quantity for the value:

| | |
|---|---|
| • “Integral” | (integral values) |
| • “Text” | (Text) |
| • “Variant” | (independently of the others, each cell can contain a floating point numerical value, an integer value or a text) |

## 2.2. Inner Margin for Text Fields

For text fields, it is now possible to specify a distance to the outer margin for all four directions (top, bottom, right, and left). This allows to e.g. control the distance of the text to the frame for a framed text field.

## 2.3. Manipulating Simulation Notifications

Errors, comments, and warnings can be manipulated in the Model Options under “Simulation” -> “Notification Settings”, i.e. they can be assigned to another level (error / warning / comment) or also entirely hidden.

To do so, a selection can be made in the left four columns in the list in each case (error number, object kind, minimal and maximal error level), and a new error level can be assigned in the last column.

PLEASE NOTE: For each message, the list is searched top-down and the first row that fulfills the message defines the new error level.

If “Callback” is selected as new error level, the EbsScript expression shown below will be called with the current message and its result will be used as new error level. The callback expression must be of the type

```
function (const error: CALCULATIONERROR) : NOTIFICATIONSETTING;
```

The types CALCULATIONERROR and NOTIFICATIONSETTING are both defined in the standard unit @System.

For example, the following EbsScript-expression

```
function (const error: CALCULATIONERROR) : NOTIFICATIONSETTING
begin
        if error.errorLevelOriginal = errorLevelWarning then
        begin
                result.errorLevel := errorLevelComment;
                result.apply := true;
        end;
end
```

turns all warnings into comments. (Please note: The expression is a so-called **unnamed function** or **lambda-function**. Those functions do not have a name, and there is no colon-sign between the return-type and the following begin.)

Alternatively, the notification settings can also be edited component-specifically in the component properties on the page “Notification Settings”. When searching, the component-specific notification settings are always treated first, before the ones specified at model level.

PLEASE NOTE: In the error bar under “Extras”, it is possible to switch off possible modifications and have the original notifications displayed by means of the option “Disable user-modifications (show original notifications)”.

## 2.4. Simulation Dialog

In the General Options under “Calculation” -> “Show message-box during/after calculation” there is now a new option: “Show simulation progress” (this is now the default setting for new models).

With this option, a dialog appears during the simulation which displays the convergence behavior and also enables the premature abortion of a simulation. Here the maximum number of iterations can be retroactively adjusted upward or downward as well.

## 2.5. User-Defined Toolbar / Menu Commands

In Release 16, menu items and toolbar buttons can be equipped with user-defined commands in the user interface.

### 2.5.1. Inserting User-Defined Commands into the Toolbar / Menu

To insert a user-defined command please select “Customize” for the menus/toolbars (Extras -> Customize...). In the “Customize” dialog under “Commands/Categories” please select “New Command”, which you can drag onto a toolbar or into a menu by means of drag&drop.

Now please activate the context menu in the newly inserted button / menu item and select “Button design...”. Besides the designation and image, there is now a “button command” as well. Here the command to be executed is specified JSON-coded. The following syntax / key word is used here:

```
{
“command”:
            {
                        “id”:“command-Id”,
                        “data”:Daten
            }
}
```

Possible commands:

| **command-id** | **data** | **Effect** |
|---|---|---|
| none | | No effect |
| simulate | | Execute simulation |
| validate | | Execute validation |
| ebsscript | {“argument”:*ID*, “param_int”:*Integer-value*, “param_string”:“*String-value*”} | Execute EbsScript |
| shiftview | {“argument”:*ID*} | Launch control room view |
| provismt | {“argument”:*ID*} | Provis multitrend |
| provisxy | {“argument”:*ID*} | Provis xy trend |
| provismultitimetrend | {“argument”:*ID*} | Provis multitime trend |
| proffirstchild | | Activate first child profile |
| proflastchild | | Activate last child profile |
| profparent | | Activate parent profile |
| profnextsibl | | Activate next sibling profile |
| profprevsibl | | Activate previous sibling profile |
| profnext | | Activate next profile |
| profprev | | Activate previous profile |
| profid | {“argument”:*ID*} | Activate profile with given ID |
| profname | {“argument”:“*name*”} | Activate profile with given name |
| provistrend | {“argument”:*ID*} | Provis trend |
| ebsscriptextern | {“argument”:“*path*”, “param_int”:*Integer-value*, “param_string”:“*String-value*”} | Execute external EbsScript |
| insert_partial_model | {“tag”:“*Tag-Name*”, “name”:“*macro-name*”} | Insert partial model/macro |
| ebsscript_direct | {“code”:“*EbsScript-Code*”} | Execute specified EbsScript code |
| open_model | {“path”:“*name/path*”} | Open model with given name / path |
| open_context | {“context”:“*name*”} | Activates the specified context in the active model |

| | | |
|---|---|---|
| multi_command | {"data":[<br>{command},...<br>]}<br>where each {command} has to be structured as follows:<br>{<br>"command":<br>{<br>"id":"command-ID",<br>"data":data<br>},<br>"on_error_continue":boolean,<br>"on_success_continue":boolean,<br>"negate_command_result":boolean<br>} | Executes several commands.<br><br>The fields<br><br>"on_error_continue",<br>"on_success_continue", and<br><br>"negate_command_result" control the execution of several commands and are all optional.<br><br>The field "on_success_continue" has "true" as default value. The fields "on_error_continue" and "negate_command_result" have "false" as default value. |

Example: Command for inserting the Ebsilon macro "Diagnosis_Coloring"

```
{
"command":
            {
                        "id":"insert_partial_model",
                        "data":{"tag":"Ebsilon","name":"Diagnosis_Coloring"}
            }
}
```

### 2.5.2. Commands for Inserting Partial Models (Macros)

When "customizing" the menus / toolbars (Extras -> Customize...), commands for inserting macros can be selected in the "Customize" dialog under "Commands/Categories".

## 2.6. Diagrams

### 2.6.1. Drawing the Background Iso Curves: Automatic Adjustment of the Search Range to the Corresponding Material Property Table

When drawing the background Iso curves in the diagrams of the type "h,s", "t,s", "log p,h", previously X- and Y-values were searched for in a statistically defined range. This range was quite suitable for the water-steam table, but less for e.g. hydrogen. Therefore the search has been refined from Release 16 on. At least for the fluids where this is possible, the attempt is made to determine at runtime the minimum and maximum values for the search directly in the respective thermodynamic function.

These limit values for the search **must not be confused** with the user-defined axis scalings / zoom settings.

### 2.6.2. Possibility to Select the Axis Unit

From Release 16 on, it is possible select a non-standard EBSILON unit for the X-axis or the Y-axis of the diagram. For this purpose, there is a combo box “Axis unit” for the respective axis (X and Y) in the diagram dialog under the menu “Options – Properties”.

### 2.6.3. Crosshair Cursor

In diagrams, you can now use a crosshair cursor displaying the current position and information via lines as well. The crosshair cursor can be switched on and off via the context menu.

## 2.7. Taking Over Line Values into a Selected Component

For component 1 (boundary value) and 33 (start value) there is the possibility to attach them to (1) or on (33) a line already filled with data and then, in the context menu (right mouse button), execute the “Take over stream results to selected component” function.

The switch FNCVSRC, which indicates whether the calorific value is calculated from the composition or specified manually, however, cannot be taken from the line, as only the calorific value itself is stored in the line, but not the information about the source of the calorific value.

For this reason, the FNCVSRC switch was left unchanged up to Release 15. However, this was a potential source of error, since the components 1 and 33 are often copied from somewhere else in the model, and then FNCVSRC kept the value that had it in the other place and therefore possibly the calculation was performed with an incorrect calorific value without notice.

In Release 16, a message box appears now in which the user must specify which setting the switch FNCVSRC should get.

Also the FFLSET switch is not available on the line. However, there is potential for danger not so high, because an incorrect setting of FFLSET leads to an error message.

# 3. Additional Modules

## 3.1. EbsScript

### 3.1.1. Digraph Parsing

Digraphs are two characters in each case that are processed as a connected token. Examples of these are “:=” (assignment) and “<=” (less or equal). In earlier versions of Ebsilon, two matching characters were also parsed as digraph if they were separated by a so-called whitespace (blanks, tabs, line breaks) or even comments.

In Standard Pascal and also Delphi, this is not allowed; therefore the goal is to detect digraphs in a future version of Ebsilon only if they follow each other immediately.

To facilitate the transition, digraphs can be separated by whitespace at most (no comments allowed in between) from Release 16 on. Moreover, the digraph “::” MUST be written connected so it is possible to tell apart the strings “::␣:” and “:␣::”.

Furthermore, in the EbsScript Editor under “Extras->Options...->Syntax” you can use the setting “relaxed digraph-parsing” to change the behavior of the parser: if “relaxed digraph-parsing” is activated, whitespace may occur between the characters, otherwise not. The behavior can also be changed via the pragma ${RDP(+)} and ${RDP(-)} respectively in the source code.

### 3.1.2. Predefined Compile Time Macros

With this Release, ***compile time macros*** are implemented. These are replaced by the respective **constants while compiling**, i.e. all values are defined at the time of the compilation and will not change later. The access syntax is:

$(Macroname)

The following predefined macros are available:

| **Macro** | **Type** | **Description** |
|---|---|---|
| $(file) | string | Name of the compiled file |
| $(line) | integer | Current line number |
| $(date) | string | Current date (at compilation!) |
| $(time) | string | Current time (at compilation!) |
| $(program) | boolean | Is the compiled code a main program? |
| $(unit) | boolean | Is the compiled code a unit? |
| $(subroutinename) | string | Name of the compiled function / procedure |
| $(subroutinefullname) | string | Name of the compiled function / procedure with type description (“function” / “procedure”, name, argument list and return type) |
| $(version) | string | Ebsilon version (e.g. 16.1.0.30567) |

| | | |
|---|---|---|
| $(versionmajor) | integer | Ebsilon Release number (e.g. 16) |
| $(versionpatch) | integer | Ebsilon Patch number (e.g. 1) |
| $(versionhotfix) | integer | Ebsilon Hotfix number (e.g. 0) |
| $(versionrevision) | integer | Ebsilon Revision number (e.g. 30567) |
| $(counter) | integer | Counter: starts at 0 and increases by 1 with each call. Starts at 0 for each individual translation unit (program, unit) |

### 3.1.3. New Primitive Type "Byte"

There is a new primitive type named "Byte". Variables and constants of this type correspond to an 8-bit-wide unsigned integer, i.e. they assume values of 0 – 255. All arithmetic calculations are executed modulo 256.

Please note that in contrast to this, the type "Char" corresponds to a 16-bit-wide integer and a UCS-2 (UTF-16) character (EbsScript strings are UCS-2-coded).

### 3.1.4. Inline Definition von Record Instances

Instances of record types can now be defined as constants inline in the code. The syntax for this is:

Type name( field1:value1; field2:value2; ... )

e.g.

```
type person_type=record
		last_name:string;
		middle_name:string;
		first_name:string;
end;

var person:person_type;
begin
		//...
		person:=person_type(last_name:"Miller"; first_name:"Arthur");
		//...
end;
```

Like with the definition of record constants, in the initialization the fields must occur in the same sequence as in the definition of the record; however, fields may be omitted (these are default-initialized).

PLEASE NOTE: This expansion only applies to record types and NOT to class types.

### 3.1.5. “case ... of” Expansions

The type for “case .. of“ can now be strings too; here the individual anchor points must be string constants:

```
procedure foo(s:string);
begin
        case s of
        “Hello” :
        begin
                // ...
        end;

        “world” :
        begin
                // ...
        end

        otherwise
        begin
                // ...
        end
        end;
end;
```

The comparison is ALWAYS executed in a case-sensitive way.

Furthermore, entire ranges can now be specified as anchor points.

- 5 .. 10 : // all values greater or equal 5 and less or equal 10
- .. 2 : // all values less or equal 2
- 15 .. : // all values greater or equal 15

```
procedure bar(value:integer);
begin
        case value of
        ..-10 :
        begin
                println(“<= -10”);
        end;
        20.. :
        begin
                println(“=> 20”);
        end;
        5..8 :
        begin
                println(“5 <= .. <= 8”);
        end;
```

```
            4 :
            begin
                        println(“== 4”);
            end;
            0 :
            begin
                        println(“== 0”);
            end
            otherwise
            begin
                        println(“something else”);
            end
            end;
end;
```

For strings, the lexicographic order is used here.

### 3.1.6. Multiple Implementation of Interfaces

An EbsScript class (ONLY “class”, NO “record”!) can now implement several interfaces at the same time. Here, for instance, is the definition of an amphibious vehicle that can swim and roll:

```
type ICanSwim = interface
            procedure swim;
end;

type IHasWheels = interface
            procedure roll;
end;

type AmphibiousVehicle = class(ICanSwim, IHasWheels)
            procedure swim;override;
            procedure roll;override;
end;
```

If name conflicts occur during the multiple implementation of interfaces, the standard allocation can be overridden by means of the method allocation clause:

```
type ICanSwim = interface
            procedure swim;
            procedure honk;
end;

type IHasWheels = interface
            procedure roll;
            procedure honk;
end;
```

```
type AmphibiousVehicle = class(ICanSwim, IHasWheels)
        procedure swim;override;
        procedure roll;override;

        procedure blow_foghorn; virtual;      // honking when swimming
        procedure trot;     virtual; // honking on wheels

        procedure ICanSwim.honk = blow_foghorn;
        procedure IHasWheels.honk = trot;

end;
```

### 3.1.7. Key Words “is” and “as”

The key words “is” and “as” are used in the context of EbsScript classes (ONLY “class”, NO “record”!) and interfaces.

The key word “is” allows to test whether the runtime type of a class or interface instance is of the type of a certain class or implements a certain interface. (All combinations of classes and interfaces are admissible here.)

The following program shows possible uses of “is” and “as”:

```
type
        IAnimal = interface
                function make_noise          : string;
        end;

type Dog = Class (IAnimal)
                function make_noise          : string; override;
                procedure tell(command:string);
        end;

function Dog.make_noise     : string;
begin
        result:=“woof”;
end;

procedure Dog.tell(command:string);
begin
        println(“Executing command: ”, command);
end;

type Cat = Class (IAnimal)
                function make_noise          : string; override;
                procedure relax();
        end;
```

```
function Cat.make_noise        : string;
begin
        result:=“purr”;
end;

procedure Cat.relax;
begin
        println(“relaxing...”);
end;

procedure interact(animal:IAnimal);
begin
        println(animal.make_noise());

        if animal is Dog then
        begin
                (animal as Dog).tell(“Sitz!”);
        end;

        if animal is Cat then
        begin
                (animal as Cat).relax();
        end;

end;

var
        a_cat:Cat;
        a_dog:Dog;
begin
        a_cat:= Cat.Create();
        interact(a_cat);
        a_dog:= Dog.Create();
        interact(a_dog);
end;
```

Output of the program:

```
purr
relaxing...
woof
Executing command: Sitz!
```

### 3.1.8. Overloading Class Operators

The overloading of implicit and explicit operators has been completely revised and previously missing operators have been added. Now the following operators for classes can be overloaded:

| Operator | Category | Declaration signature | Operator Symbol |
|---|---|---|---|
| Implicit | Conversion | Implicit(a : type): result type; | Implicit type conversion |
| Explicit | Conversion | Explicit(a: type): result type; | Explicit type conversion |
| Negative | Unary | Negative(a: type): result type; | - |
| Positive | Unary | Positive(a: type): result type; | + |
| Inc | Unary | Inc(a: type): result type; | Inc |
| Dec | Unary | Dec(a: type): result type | Dec |
| LogicalNot | Unary | LogicalNot(a: type): result type; | not |
| Trunc | Unary | Trunc(a: type): result type; | Trunc |
| Round | Unary | Round(a: type): result type; | Round |
| Equal | Comparison | Equal(a: type; b: type): Boolean; | = |
| NotEqual | Comparison | NotEqual(a: type; b: type): Boolean; | <> |
| GreaterThan | Comparison | GreaterThan(a: type; b: type) Boolean; | > |
| GreaterThan-OrEqual | Comparison | GreaterThanOrEqual(a: type; b: type): Boolean; | >= |
| LessThan | Comparison | LessThan(a: type; b: type): Boolean; | < |
| LessThan-OrEqual | Comparison | LessThanOrEqual(a: type; b: type): Boolean; | <= |
| Add | Binary | Add(a: type; b: type): result type; | + |
| Subtract | Binary | Subtract(a: type; b: type): result type; | - |
| Multiply | Binary | Multiply(a: type; b: type): result type; | * |
| Divide | Binary | Divide(a: type; b: type): result type; | / |
| IntDivide | Binary | IntDivide(a: type; b: type): result type; | div |
| Modulus | Binary | Modulus(a: type; b: type): result type; | mod |
| LeftShift | Binary | LeftShift(a: type; b: type): result type; | shl |
| RightShift | Binary | RightShift(a: type; b: type): result type; | shr |
| LogicalAnd | Binary | LogicalAnd(a: type; b: type): result type; | and |
| LogicalOr | Binary | LogicalOr(a: type; b: type): result type; | or |
| LogicalXor | Binary | LogicalXor(a: type; b: type): result type; | xor |
| BitwiseAnd | Binary | BitwiseAnd(a: type; b: type): result type; | bitwiseand |
| BitwiseOr | Binary | BitwiseOr(a: type; b: type): result type; | bitwiseor |
| BitwiseXor | Binary | BitwiseXor(a: type; b: type): result type; | bitwisexor |

### 3.1.9. New and Extended Functions

- Function “getFluxDirectionAtLink” added: returns the flow direction to a component pin.

- Function “getLineTypeAtLink” made applicable to components that dock onto lines (types 33, 45, 46, 105, and 147): if the component docks onto a line and is specified for “linkNo” of the value 1, the type of the line will be returned.

- Functions
  “getProductVersionReleaseNumber”
  “getProductVersionPatchNumber”
  “getProductVersionHotfixNumber”
  “getProductVersionRevisionNumber”
  These functions return the individual positions of the version number (e.g. 15.4.0.29840) as integer values.

- Function “runEbsScript” for macro objects added. For the explicit execution of the macro EbsScripts “to run before calculation”, “to run after calculation” and “to take over nominal values”.

- Function “getCalculationErrors” added: returns the errors/warnings/comments of the latest calculation on the current profile.

- Method “getobjects(ebsobjecttype:string, includeChildContexts:boolean)” added to the type “ebsmacrobase”: allows access to objects within a macro (if applicable, filtered according to type or whether objects from contained macros are to be contained too).

- Property “shape:integer” added to the type “ebscomp”: allows to read and set the current display format.

- Property “shapeCount:integer” (read only) added to the type “ebscomp”: allows to read the number of different display formats.

- Procedure “ksSetComponentCalcState (component:ebscomp; enable_calculation:Boolean; recursive:Boolean = False)” added in the standard unit @KernelScripting: This function allows to temporarily deactivate the calculation of a component during the simulation. In doing so, temporarily deactivated components retain the equations from the previous iteration step. If “component” is a macro object, all objects within the macro can be temporarily activated and deactivated respectively by means of “recursive=True”.

- Please note that this is a highly advanced feature which may lead to calculation errors that are not easily comprehensible if applied incorrectly.

- Mathematical functions

| | |
|---|---|
| “trunc(arg:Real):Real” | : returns the integer part of a real number, with the same algebraic sign as that of “arg”. |
| “frac(arg:Real):Real” | : returns the part after the decimal point of a real number, with the same algebraic sign as that of “arg”. |

### 3.1.10. New EbsScript Standard Unit @KernelInterface
The new EbsScript standard unit @KernelInterface pools functions from the standard units @KernelScripting and @KernelExpression that are available independent of the current calculation domain (e.g. KernelScript or KernelExpression). These functions can be called by any EbsScripts during a simulation or validation. By means of the function "kiGetMode", you can determine in which iteration mode the application currently is and if at all.

### 3.1.11. Key Word and Object Tree
At the top of both tree diagrams, there is now a text field by means of which the display of the tree can be filtered. Furthermore, to the right of it there is now a dropdown button that displays filtering options.

As long as no uppercase letters are entered, the search runs independently of capitalization. As soon as at least one uppercase letter is entered, the search is case-sensitive. In case the search-text starts with double-quotes (") then the text must start with the search-text. In case the search-text ends with double-quotes then the text must end with the search-text.

Regarding the functions, all functions from all internal units and the ones directly implemented into the language are now displayed.

Furthermore, in the object-tree a dot (.) is used to search in the different sub-levels of the tree: with every dot the search moves to the next level. A search text can thus be created for each level (which can also contain double-quotes, for example). The option "Ignore namespaces" ignores macro namespaces in the search, i.e. all objects (including those in macros) are searched for in the top level.

## 3.2. EbsOpen

In the header file "EbsUser.h", there is now a new version of the calculation interface.

Version 10 (in the namespace "v10", only available for C++) contains a CallBack function for calculating material properties. This makes linking to wdt.lib and mixture.lib obsolete.

### 3.2.1. Values of Result Values / Fields / Families, and Matrices
Now the values of result values/fields/families and matrices always come from the active profile, i.e. no further search takes place in the parent profiles (of course, values bound to particular profiles by means of "Fix..." are not affected by this).

### 3.2.2. Interface IFluidAnalysis
The interface IFluidAnalysis has two new methods "GetAllFractions" and "SetAllFractions", by means of which all substances can be read and written respectively at the same time.

### 3.2.3. Interface IComp
The property “Shape:long” and the read-only property “ShapeCount:long” for reading and setting the current display format and for reading the number of different display formats respectively have been added.

### 3.2.4. CoClasses ApplicationBuilder and DllBuilder
The CoClasses ApplicationBuilder and DllBuilder allow to specify an EbsOpen instance (Dll-Inproc or Exe-OutofProcess) incrementally and to create it by calling “Finalize”. The option to additionally specify license options is new here.

### 3.2.5. .net Class EbsOpen.Factory
Just like the new CoClasses in (3.2.3), the method “EbsOpen.Factory .Create(InitParams init_params)” supports the new license-options. For this purpose, please use the field “builder_params” (of the type “BuilderParams”) in the structure “InitParams”.

### 3.2.6. IApplication / IModel RunExtendedCommand
By means of the method “RunExtendedCommand”, a user-defined toolbar command (cf. 2.5) can be executed at application and model level respectively.

## 3.3. Excel AddIn
In the case of specification and result values in the Excel AddIn (“spec” and “result” types), EbsOpen properties of the objects can now be called as well. For instance, a specification/result value “HTX_1.Description” sets / reads the text of the “description” in the component named “HTX_1”.

PLEASE NOTE: The code after the object name is evaluated as C# expression, which needs to be considered in the specification of further arguments and the further linking of properties respectively.

Furthermore, besides the EbsScript evaluation type “expr” for results, there is now a new type “assign” (or also “assignment”) that allows to set values via EbsScript syntax before the calculation. For this purpose, the EbsScript assignment operator “:=” is added to the expression in the column “Name” and the value is added in the profile column. If the unit “String” is present in the unit column, the text is set in quotation marks and possibly occurring control signs are protected by means of backslashes.

For instance:

| Type | Name | Description | Unit | FirstProfile |
|---|---|---|---|---|
| assign | '@calcoptions.sim.waterSteamTable | | | 1 |
| assign | P4.MEASM | | String | P4.MEASM + 0.01 |

(Please note the single-quote ‘ at the start of upper expression. It is necessary to tell Excel that the content of the cell is a text.) In this example, the EbsScript expression

@calcoptions.sim.waterSteamTable := 1

is executed for the first line and the EbsScript expression

```
P4.MEASM := "P4.MEASM + 0.01"
```

is executed for the second line (i.e. in the second line, in Component 46 named “P4” the expression “P4.MEASM + 0.01” is set for the specification value “MEASM”).

Please note that the most EbsOpen-properties resp. EbsScript-assignments are **not profile-dependent** (like “Description”). If such a value is changed within one profile it is changed in all other profiles also.

## 3.4. Xui.dll and Programmable.dll

There is a new version of the calculation interface in the header file “EbsUser.h”.

Version 10 (in namespace “v10”, only available for C++) contains a CallBack function for calculating physical properties. This makes linking to wdt.lib and mixture.lib obsolete.

## 3.5. SRxWebAPI

The SRxWebAPI can now also handle HTTPS (TSL/SSL) connections. Possibly required certificates must be added via the operating system in the Windows Certificate Manager!

# 4. Changes in the Results

## 4.1. Three Way Valve (Component 49)

By mistake, up to Release 15 it was accepted for this component to have the pressure specified on the outlet line. According to the valve position, it was then transferred onto one of the two inlet lines. However, this type of specification could only work for one of the two valve positions. In the event of a switchover of the valve, the pressure would be transferred onto the other inlet line, where it would then be defined twice, while the pressure specification would be missing on the other line. Unfortunately, no error message was output in this case either. The problem has been fixed by no longer accepting a specification of the pressure on the outlet line. The affected models (which only worked correctly with one valve position anyway) have to be modified accordingly.

Another thing that has been changed is the behavior when all mass flows are almost zero. Previously, simply a mean value of the two inlets was formed. For the lines themselves with zero mass flow, the pressure does not matter, and therefore, this simplification is not a problem. However, it happened that this pressure was transferred to other lines and caused wrong results. Therefore, this simplification has been removed.

## 4.2. Stratified Storage (Component 145)

Up to Release 15, the calculation of the fluid pressure was not plausible. In Release 16, the pressure calculation has been corrected. Therefore the result values change, mainly the calculated fluid pressure values at the lines connected to the component as well as the result value DPGEOD in Component 145.

Thus it may be necessary to adjust existing models to the modification of Release 16 in order to eliminate the new error messages like e.g. regarding steam fractions in the water.

The impact of the pressure calculation on the temperature fields is very low, so that no significant change of the results is to be expected here.

## 4.3. k-Values for Heat Exchangers

### 4.3.1. Consideration of Both Heat Transfer Coefficients

When calculating the k-value, both heat transfer coefficients are now considered as a rule:

1/k = 1/al12 + 1/al34

Previously, the al12 term was often neglected concerning the economizer and the evaporator as in the case of al12 >> al34 it only has a small impact. However, also at al12 = 5000 W/m$^2$K and al34 = 50 W/m$^2$K there was an error of approximately 1%, which no longer occurs from Release 16 on.

However, this approximation did not have any great effect before either as the thermodynamics only depends on the product k*A, which was calculated precisely before as well. Thus in the design case, an inaccuracy in A only had an effect on the result value A (area) but not on the thermodynamic data, and not on the line results in particular.

Regarding applications that require the area or use working fluids with other alpha numbers, the more precise calculation certainly makes sense.

If the more precise calculation is not desired, you can basically cause the AL12 term to be neglected and restore the old results by specifying very high numeral values for AL12.

In off-design, however, slight changes in the line results due to the consideration of both heat transfer coefficients may occur in components where the off-design behavior is determined by exponents. This is caused by the non-linearity that enters the calculation due to these exponents as the exponents are not defined for k*A as a whole but for AL12 and AL34.

### 4.3.2. Special Treatment: Default Value AL12N = 500 W/m²K

However, the consideration of both heat transfer coefficients will only lead to results that are better, i.e. closer to reality, if the heat transfer coefficients are correct too. If, however, the default value AL12N = 500 W/m²K is used for the economizer or the evaporator, the results will become unrealistic. Unfortunately, up to Release 14 these 500 W/m²K were also the default value for Component 70 (Evaporator with steam drum).

To prevent the consideration of an AL12N of 500 W/m²K from leading to unrealistic results, a special treatment has been implemented for this value: a warning is output and the calculation is carried out with AL12N = 6000 W/m²K (economizer) and AL12N = 10000 W/m²K (evaporator) respectively.

## 4.4. Result Values in Components 128, 129 at FFU = 0

In Components 128 (Coal mill (hard coal)) and 129 (Coal mill (lignite)), the result values at FFU = 0 (Mill OFF) assume more plausible values from Release 16 on. The irrelevant result values are set to void.

## 4.5. Component 131 at FDELAY > 0

In Component 131 (Transient separator), an error has been fixed that had occurred in combination with the flag FDELAY > 0.

## 4.6. Component 118 at M1=M2

In Component 118 (Direct storage), an error has been fixed that had occurred in case M1 = M2 (equal mass flow values at inlet and at outlet).

# 5. Known Errors

## 5.1. Documentation

Unfortunately, it has not been possible to include all of the new features of Release 16 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“.

**STEAG Energy Services GmbH**
Wetzbach 35
64673 Zwingenberg
Phone: +49 6251 1059-0
info@ebsilon.com
www.ebsilon.com