SecCool API

MORTEN JUEL SKOVRUP

Document version 1.11
Date 2005-10-03
Contact mjs@ipu.dk

Contents

1 Introduction

SecCool is a program for calculating, comparing and plotting thermophysical properties of secondary refrigerants.

SecCool is a package of programs consisting of three parts:

1. **SecCool Properties**. The main program, described "SecCool Properties Users Manual"
2. **SecCool Datafit**. Program for adding new secondary refrigerants to the package. SecCool Datafit is documented separately in "SecCool Datafit Users Manual".
3. **SecCool API**. Two dll's which can be used in your own programs. The API is documented in this document.

The SecCool package is developed as part of the project DESIK (*Energirigtig design og regulering af sekundærsiden på indirekte køleanlæg med naturlige kølemidler*) financed by ELFOR R&D (#334-001). Project period from 2003-04-01 to 2005-12-31.

1.1 Installation

The package comes with two dll's, located in the **`SecCool API`** subdirectory of the installation directory:

1. A general purpose dll called **`SecCoolEqns.dll`**, which can be used from several programming languages, Excel, LabView, Matlab etc.
2. A dll called **`SecCoolEES.dll`** which can be used from EES.

Installing the general purpose dll

1) Copy the dll you want to use from the **`SecCool API\SecCoolEqns`** directory to a directory of your choice.
2) Copy the **`Fluids`** directory to the same directory (i.e. the **`Fluids`** directory should be a subdirectory to the directory where the dll is located).

Normally the destination directory will be same directory where the program using the dll is installed, but you could also copy the contents to your **`windows\system32`** directory (or **`windows\system`** on win95 and win98 machines). As default **`SecCoolEqns.dll`** expects the **`Fluids`** directory to a subdirectory to the directory where **`SecCoolEqns.dll`** is installed, but you can change the placement of the **`Fluids`** directory by calling the **`SetFluidsDir`** function (see chapter 2).

Installing the EES dll

1) Copy the contents of the **`SecCool API\SecCoolEES`** directory (including the **`Fluids`** subdirectory) to your **`EES32\Userlib`** directory.
2) Copy the **`Fluids`** directory to the same directory (i.e. the **`Fluids`** directory should be a subdirectory to the **`EES32\Userlib`** directory).

When you start EES **`SecCoolEES.dll`** will automatically create a file called **`AllFluids.txt`** in the **`EES32\Userlib\Fluids`** directory, which contains the fluid numbers of the installed fluids (you have to use these numbers in calls to the dll from EES).

In the description of the functions in chapter 3 to 5 (where the functions you can call from EES are described), you should note that the call from EES only differ in that a parameter giving the fluid number is the first parameter in the function heading.

1.2 Fluids

The following fluids are (so far) included in this package (note that the numbers might change in future versions):

Number	Trade name
0	Aspen Temper -10
1	Aspen Temper -20
2	Aspen Temper -30
3	Aspen Temper -40
4	Aspen Temper -55
5	HYCOOL 20
6	HYCOOL 30
7	HYCOOL 40
8	HYCOOL 45
9	HYCOOL 50
10	Freezium
11	ASHRAE, Carbon dioxide
12	Ice slurry, Propylene Glycol
13	Ice slurry, Ethanol
14	Ice slurry, NaCl
15	Antifrogen KF
16	Antifrogen L
17	Antifrogen N
18	ASHRAE, Ethylene Glycol
19	ASHRAE, Propylene Glycol
20	Dowtherm J
21	Dowtherm Q
22	Glykosol N
23	HFE-7100
24	Melinder, Ammonia
25	Melinder, Calcium Cloride
26	Melinder, Ethanol
27	Melinder, Ethylene glycol
28	Melinder, Glycerol
29	Melinder, Magnesium Chloride
30	Melinder, Methanol
31	Melinder, Potassium Acetate
32	Melinder, Potassium Carbonate
33	Melinder, Propylene Glycol
34	Melinder, Sodium Chloride
35	Pekasol 2000
36	Pekasol L
37	Syltherm XLT
38	VDI, Calcium Cloride
39	VDI, Magnesium Chloride
40	VDI, Methanol
41	VDI, Potassium Carbonate
42	VDI, Sodium Chloride
43	Water

1.3 Fluid kind, types and trade names

Each fluid in the package has three attributes:

1. Fluid kind. This can be one of:
    a. Pure fluid – i.e. not including water solutions, but including pre-mixed water solutions only sold in specific concentrations (properties depends only on temperature)
    b. Water solution (properties depends on temperature and concentration)
    c. Two-phase fluid (properties depends on temperature and quality)
    d. Ice-slurry (properties depends on temperature and ice concentration)
2. Fluid type. Each fluid has a type – for example propylene glycol – but for each type there may be several trade names or equation sources.
3. Trade name. Each fluid has a unique trade name. This name can be a trade name directly (for example Pekasol 2000) or it can be a name indicating the source for the equations used in this package (for example Melinder, Propylene glycol)

1.4 Conventions

The dll uses `stdcall` calling convention for all functions.

The unit system can be changed, but as default the following units are used:

Property	Units
Temperature	$^{\circ}C$
Density	kg/m^3
Specific heat	$J/(kg \cdot K)$
Thermal conductivity	$W/(m \cdot K)$
Dynamic viscosity	cP
Kinematic viscosity	cSt
Diameter	m
Length	m
Roughness	m
Heat-transfer coefficient	$W/(m^2 \cdot K)$
Pressure	bar
Pressure drop	$pascal$
Velocity	m/s
Enthalpy	J/kg

The following standard types are used in the package:

Object Pascal	C/C++	Meaning
`Double`	`double`	8-byte floating point number
`Integer`	`int`	signed 32-bit integer
`WordBool`	`bool`	2-byte boolean value False if 0 True if different from zero
`PChar`	`char*`	Pointer to zero-terminated character array.

Most functions exposed by the dll take temperature and concentration as parameter.

Concentration can have different meaning depending of the kind of fluid that is used:

- If the fluid is a water solution the concentration is the concentration of the secondary coolant, either by mass or by volume.
- If the fluid is a two-phase fluid (CO_2) then concentration equals quality (i.e. can be between 0 (liquid) and 1 (gas))
- If the fluid is ice-slurry the concentration is the ice-concentration.

NOTE: Most functions have a counterpart which is called only with temperature as a parameter. These functions can be called for pure fluids

1.5 Calculation details

For all fluids reference for equations/data can be inquired by calling `GetReference` (see 2). Besides from that the following correlations for pressure drop and heat-transfer coefficients have been used:

- Pure fluids and water solutions
  - Pressure drop: Colebrook's formula [2] with smoothening function in transition area
  - Heat-transfer coefficient: Gnielinski [3] with smoothening function in transition area
- Two-phase fluids (CO_2)
  - Pressure drop: Correlation by Müller-Steinhagen & Heck [4]
  - Heat-transfer coefficient: Cannot be calculated in current version
- Ice slurry
  - Pressure drop: Correlation by Danish Technological Institute [5]
  - Heat-transfer coefficient: Cannot be calculated in current version

Note that thermodynamic properties (density, specific heat, conductivity and viscosity) for two-phase fluids are calculated using the leaver-rule (i.e. simple weighting by quality) – so use these properties with caution.

2 Information functions

The functions in this chapter cannot be called from EES!

```
function SetFluidsDir(ADir : PChar) : WordBool; stdcall;
```

Sets the directory where the fluid-files (***.secfit**) can be found. As default a subdirectory called **Fluids** located beneath the directory of the dll is assumed to contain the fluid-files. Returns **True** if successful, **False** otherwise.

Example:

```
SetFluidsDir('c:\SecCoolEqns\Fluids');
```

```
function GetFluidTypes(Types : PChar) : Integer; stdcall;
```

Returns the types of fluids in the package in **Types**. A fluid-type is for example "Calcium Cloride". If **Types** equals **nil**, the function only returns the necessary length of **Types**. **Types** is a comma-separated list of the fluid types (if the type-name contains spaces, it is enclosed in double quotes).

Example:

```
var
  Len : Integer;
  FluidTypesStr : PChar;
begin
  Len := GetFluidTypes(PChar(nil));
  GetMem(FluidTypesStr,Len+1);         //Allocate memory
  GetFluidTypes(FluidTypesStr);        //Now get the string
  //FluidTypesStr is now equal to "Calcium cloride",Glycerol,...
  //Use FluidTypesStr...
  FreeMem(FluidTypesStr,Len+1);        //Free FluidTypesStr after it's
                                         been used
end;
```

```
function GetFluidsByType(FluidType,Fluids : PChar) : Integer; stdcall;
```

Returns all fluids of type **FluidType** in **Fluids**. If **Fluids** equals **nil**, the function returns the necessary length of **Fluids**. **Fluids** is a comma-separated list of the fluids (if the fluid-name contains spaces, it is enclosed in double quotes).

Example:

```
var
  Len       : Integer;
  FluidsStr : PChar;
begin
  Len := GetFluidsByType('Calcium cloride',PChar(nil));
  GetMem(FluidsStr,Len+1);                      //Allocate memory
  GetFluidsByType('Calcium cloride',FluidsStr);  //Now get the string
  //FluidsStr is now equal to "Melinder, Calcium Cloride",...
  //Use FluidsStr...
  FreeMem(FluidsStr,Len+1); //Free FluidsStr after it's been used
end;
```

```
function GetTradeNames(TradeNames : PChar) : Integer; stdcall;
```

Returns the trade names of all the fluids installed in **TradeNames** (a trade name is for example "Pekasol 2000"). If **TradeNames** equals **nil**, the function returns the necessary length of **TradeNames**. **TradeNames** is a comma-separated list of the trade names (if the trade name contains spaces, it is enclosed in double quotes).

See example for **GetFluidTypes**.

```
procedure SetFluidNumber(ANumber : Integer); stdcall;
```

Selects the fluid with number **ANumber**. The fluids gets numbered as they are loaded from the fluids directory, so you cannot be sure this number is the same for the fluids in future versions of this package. You can however be sure that the list you get when you call **GetTradeNames** reflects the number of the fluid (i.e. the first fluid in the list returned by **GetTradeNames** has number 0, the second number 1 etc.)

```
function GetFluidNumber(TradeName : PChar) : Integer; stdcall;
```

Returns the number of the fluid with trade name equal to **TradeName**.

```
function GetFluidType(AType : PChar) : Integer; stdcall;
```

Returns the fluid type of the currently selected fluid. If **AType** equals **nil**, the function returns the necessary length of **AType**.

See example for **GetFluidTypes**.

```
function GetFluidKind : Integer; stdcall;
```

Returns the fluid-kind of the currently selected fluid, can be:

0: Pure fluids (i.e. not water solution but including pre-mixed water solution only sold in specific concentrations)
1: Water solution
2: Two-phase fluid (for example CO_2)
3: Ice-slurry

```
function GetCasNr(ANr : PChar) : Integer; stdcall;
```

Returns the CAS-number for the currently selected fluid (not implemented for all fluids yet). If **ANr** equals **nil**, the function returns the necessary length of **ANr**.

See example for **GetFluidTypes**.

```
function GetTradeName(ATradeName : PChar) : Integer; stdcall;
```

Returns the trade name for the currently selected fluid. If **ATradeName** equals **nil**, the function returns the necessary length of **ATradeName**.

See example for **GetFluidTypes**.

```
function GetReference(AReference : PChar) : Integer; stdcall;
```

Returns the reference for the currently selected fluid. If **AReference** equals **nil**, the function returns the necessary length of **AReference**.

See example for **GetFluidTypes**.

```
function GetConcBase : Integer; stdcall;
```

Returns the concentration unit for the currently selected fluid (works only for water solutions). The concentration can be one of:

0: None (fluid is not a water soluition)
1: Mass %
2: Volume %

```
procedure SetConcBase(ABase : Integer); stdcall;
```

Set the concentration unit for the currently selected fluid (works only for water solutions). Call this function only if **ConcBaseChange** returns **True**.
ABase can be one of:

0: None
1: Mass %
2: Volume %

```
function GetEqConcBase : Integer; stdcall;
```

Get the default concentration unit for the currently selected fluid (see **GetConcBase** for possible return values).

```
function ConcBaseChange : WordBool; stdcall;
```

If return value is true, then you can switch between Mass % and Volume % by calling **SetConcBase**.

```
function GetValidate : WordBool; stdcall;
```

Returns whether validation is turned on (it is on by default).

```
procedure SetValidate(AValidate : WordBool); stdcall;
```

Sets whether validation should be on. If validation is Off then no checks are performed on the input values to the function calls. If validation is On, then calling a function with values outside the valid range will make the function return -1.7e308.

```
function GetFormula(AFormula : PChar) : Integer; stdcall;
```

Returns the chemical formula for the currently selected fluid (not implemented for all fluids yet). If **AFormula** equals **nil**, the function returns the necessary length of **AFormula**.

See example for **GetFluidTypes**.

3 Thermophysical properties

Note: functions which take only temperature as a parameter can only be used with pure fluids. Functions which take both temperature and concentration can be used will all fluid kinds (for pure fluids the value of concentration is ignored).

```
function RhoTX(T,X : Double): Double; stdcall;
```

Returns density as function of temperature and concentration

```
function CpTX(T,X : Double): Double; stdcall;
```

Returns specific heat as function of temperature and concentration

```
function CondTX(T,X : Double): Double; stdcall;
```

Returns thermal conductivity as function of temperature and concentration

```
function MuTX(T,X : Double): Double; stdcall;
```

Returns dynamic viscosity as function of temperature and concentration

```
function NuTX(T,X : Double): Double; stdcall;
```

Returns kinematic viscosity as function of temperature and concentration

```
function RhoT(T : Double): Double; stdcall;
```

Returns density as function of temperature (works only for pure fluids)

```
function CpT(T : Double): Double; stdcall;
```

Returns specific heat as function of temperature (works only for pure fluids)

```
function CondT(T : Double): Double; stdcall;
```

Returns conductivity as function of temperature (works only for pure fluids)

```
function MuT(T : Double): Double; stdcall;
```

Returns dynamic viscosity as function of temperature (works only for pure fluids)

```
function NuT(T : Double): Double; stdcall;
```

Returns kinematic viscosity as function of temperature (works only for pure fluids)

4 Concentration and freezing point functions

```
function  MassToVol(X : Double) : Double; stdcall;
```

Converts concentration in Mass% to Vol% for selected fluid. See **ConcBaseChange**

```
function  VolToMass(X : Double) : Double; stdcall;
```

Converts concentration in Vol% to Mass% for selected fluid. See **ConcBaseChange**

```
function TFreeze : Double; stdcall;
```

Returns freezing point temperature for pure fluids.

```
function  TFreezeX(X : Double) : Double; stdcall;
```

Returns freezing point temperature as a function of concentration for water solutions.

```
function  XTFreeze(T : Double) : Double; stdcall;
```

Returns concentration as function of freezing point temperature for water solutions.

```
procedure TminMaxX(X : Double; var Tmin,Tmax : Double); stdcall;
```

Returns valid temperature area as function of concentration for water solutions.

```
procedure XMinMaxT(T : Double; var XMin,XMax : Double); stdcall;
```

Returns valid concentration area as function of temperature for water solutions.

```
procedure TMinMax(var TMin,TMax : Double); stdcall;
```

Returns valid temperature area for the selected fluid.

```
procedure XMinMax(var XMin,XMax : Double); stdcall;
```

Returns valid concentration area for the selected fluid.

```
procedure TFreezeMinMax(var TMin,TMax : Double); stdcall;
```

Returns min and max freezing point temperature for the selected fluid.

5 Additional functions

Note: functions which take only temperature as a parameter can only be used with pure fluids. Functions which take both temperature and concentration can be used will all fluid kinds (for pure fluids the value of concentration is ignored) except otherwise noted.

```
function HEvapTX(T,X : Double) : Double; stdcall;
```

Returns heat of evaporation as function of temperature and concentration. Works only for:

- Ice slurry
- Two-phase fluids (in which case the concentration parameter is ignored)

```
function DHTX(T,X1,X2 : Double) : Double; stdcall;
```

Return enthalpy difference between two concentrations. Works only for:

- Ice slurry
- Two-phase fluids (in which case the concentration parameter is quality)

```
function HTX(T,X : Double) : Double; stdcall;
```

Returns enthalpy as function of temperature and concentration. Works only for:

- Ice slurry
- Two-phase fluids (in which case the concentration parameter is quality)

```
function PBubTX(T,X : Double) : Double; stdcall;
```

Returns bubble point pressure as function of temperature for the selected fluid. So far only two-phase fluids are supported.

```
function Prandtl(Cp,Cond,Mu : Double) : Double; stdcall;
```

Returns the Prandtl number as function of specific heat, conductivity and dynamic viscosity.

```
function PrandtlT(T : Double) : Double; stdcall;
```

Returns the Prandtl number as function of temperature.

```
function PrandtlTX(T,X : Double) : Double; stdcall;
```

Returns the Prandtl number as function of temperature and concentration.

```
function Reynolds(Rho,Mu,u,D : Double) : Double; stdcall;
```

Returns the Reynolds number as function of density, dynamic viscosity, velocity and hydraulic diameter.

```
function ReynoldsT(T,u,D : Double) : Double; stdcall;
```

Returns the Reynolds number as function of temperature, velocity and hydraulic diameter.

```
function ReynoldsTX(T,X,u,D : Double) : Double; stdcall;
```

Returns the Reynolds number as function of temperature, concentration, velocity and hydraulic diameter.

```
function PressureDrop(Rho,Mu,u,L,D,Eps : Double) : Double; stdcall;
```

Returns pressure drop as function of density, dynamic viscosity, velocity, length, diameter and roughness. This procedure works for pure fluids, water solutions and ice slurry but not for two-phase fluids.

```
function PressureDropT(T,u,L,D,Eps : Double) : Double; stdcall;
```

Returns pressure drop as function of temperature, velocity, length, diameter and roughness. This procedure works for pure fluids.

```
function PressureDropTX(T,X,u,L,D,Eps : Double) : Double; stdcall;
```

Returns pressure drop as function of temperature, concentration, velocity, length, diameter and roughness. This procedure works for all fluids.

```
function PressureDropTXInOut(T,X_in,X_out,u,L,D,Eps : Double) : Double; stdcall;
```

Returns pressure drop as function of temperature, inlet quality, outlet quality, velocity, length, diameter and roughness. This procedure works for two-phase fluids.

```
function HeatTransferCoef(Rho,Cp,Cond,Mu,u,L,D,Eps : Double) : Double; stdcall;
```

Returns heat-transfer coefficient as function of density, specific heat, conductivity, dynamic viscosity, velocity, length, diameter and roughness. This procedure works for pure fluids and water solutions (and for two-phase fluids in either gas- or liquid-phase).

```
function HeatTransferCoefT(T,u,L,D,Eps : Double) : Double; stdcall;
```

Returns heat-transfer coefficient as function of temperature, velocity, length, diameter and roughness. This procedure works for pure fluids.

```
function HeatTransferCoefTX(T,X,u,L,D,Eps : Double) : Double; stdcall;
```

Returns heat-transfer coefficient as function of temperature, concentration, velocity, length, diameter and roughness. This procedure works for pure fluids and water solutions (and for two-phase fluids in either gas- or liquid-phase).

```
function HTEFTX(T,X : Double) : Double; stdcall;
```

Returns the relative Heat-Transfer Efficiency Factor as function of temperature and concentration (compared to water at 10°C). Works for pure fluids and water solutions (and for two-phase fluids in either gas- or liquid-phase). See [1].

```
function HTEFT(T : Double) : Double; stdcall;
```

Returns the relative Heat-Transfer Efficiency Factor as function of temperature (compared to water at 10°C). Works for pure fluids. See [1].

6 Unit functions

The functions in this chapter cannot be called from EES!

```
procedure SetDensityUnit(AUnit : Integer); stdcall;
```

Select the unit for density. **AUnit** can be one of:

0 $\left[kg/m^3\right]$

1 $\left[kg/L\right]$

2 $\left[g/L\right]$

3 $\left[lb/\ ft^3\right]$

4 $\left[lb/in^3\right]$

```
procedure SetSpecificHeatUnit(AUnit : Integer); stdcall;
```

Select the unit for specific heat. **AUnit** can be one of:

0 $\left[kJ/\left(kg\cdot K\right)\right]$

1 $\left[J/\left(kg\cdot K\right)\right]$

2 $\left[BTU/\left(lb\cdot {}^\circ F\right)\right]$

3 $\left[BTU/\left(lb\cdot {}^\circ R\right)\right]$

```
procedure SetConductivityUnit(AUnit : Integer); stdcall;
```

Select the unit for thermal conductivity. **AUnit** can be one of:

0 $\left[W/\left(m\cdot K\right)\right]$

1 $\left[BTU/\left(h\cdot ft\cdot {}^\circ F\right)\right]$

2 $\left[BTU\cdot in/\left(h\cdot ft^2\cdot {}^\circ F\right)\right]$

3 $\left[BTU\cdot in/\left(s\cdot ft^2\cdot {}^\circ F\right)\right]$

```
procedure SetDynamicViscosityUnit(AUnit : Integer); stdcall;
```

Select the unit for dynamic viscosity. **AUnit** can be one of:

0 $\left[cP\right]$

1 $\left[P\right]$

2 $\left[Pa\cdot s\right]$

3 $\left[lb/\left(\ ft\cdot s\right)\right]$

4 $\left[lb/\left(\ ft\cdot h\right)\right]$

procedure SetKinematicViscosityUnit(AUnit : Integer); **stdcall**;

Select the unit for kinematic viscosity. **AUnit** can be one of:

0 $[cSt]$

1 $[St]$

2 $\left[m^2/s\right]$

3 $\left[ft^2/s\right]$

4 $\left[ft^2/h\right]$

procedure SetEnthalpyUnit(AUnit : Integer); **stdcall**;

Select the unit for enthalpy. **AUnit** can be one of:

0 $[kJ/kg]$

1 $[J/kg]$

2 $[BTU/lb]$

procedure SetTemperatureUnit(AUnit : Integer); **stdcall**;

Select the unit for temperature. **AUnit** can be one of:

0 $\left[^{\circ}C\right]$

1 $[K]$

2 $\left[^{\circ}F\right]$

3 $\left[^{\circ}R\right]$

procedure SetPressureUnit(AUnit : Integer); **stdcall**;

Select the unit for pressure (as returned by the **PBubTX** function). **AUnit** can be one of:

0 $[bar]$

1 $[kPa]$

2 $[Pa]$

3 $[psi]$

4 $[psf]$

procedure SetPressureDropUnit(AUnit : Integer); **stdcall**;

Select the unit for pressure drop (as returned by the pressure drop functions). **AUnit** can be one of:

0 $[bar]$

1 $[kPa]$

2 $[Pa]$

3 $[psi]$

4 $[psf]$

```
procedure SetHeatTransferCoeffUnit(AUnit : Integer); stdcall;
```

Select the unit for heat-transfer coefficient. **AUnit** can be one of:

0 $\left[W/\left(m^2 \cdot K\right)\right]$

1 $\left[kW/\left(m^2 \cdot K\right)\right]$

2 $\left[BTU/\left(h \cdot ft^2 \cdot {}^{\circ}F\right)\right]$

3` $\left[BTU/\left(s \cdot ft^2 \cdot {}^{\circ}F\right)\right]$

```
procedure SetDiameterUnit(AUnit : Integer); stdcall;
```

Select the unit for diameter (used in calls to pressure drop and heat-transfer coefficient functions). **AUnit** can be one of:

0 $[mm]$

1 $[cm]$

2 $[m]$

3 $[in]$

4 $[ft]$

```
procedure SetLengthUnit(AUnit : Integer); stdcall;
```

Select the unit for length (used in calls to pressure drop and heat-transfer coefficient functions). **AUnit** can be one of:

0 $[m]$

1 $[cm]$

2 $[ft]$

3 $[in]$

4 $[yd]$

```
procedure SetRoughnessUnit(AUnit : Integer); stdcall;
```

Select the unit for pipe roughness (used in calls to pressure drop and heat-transfer coefficient functions). **AUnit** can be one of:

0 $[mm]$

1 $[cm]$

2 $[m]$

3 $[in]$

4 $[ft]$

```
procedure SetVelocityUnit(AUnit : Integer); stdcall;
```

Select the unit for velocity (used in calls to pressure drop and heat-transfer coefficient functions). **AUnit** can be one of:

0 $[m/s]$

1 $[ft/s]$

2 $[ft/min]$

```
function GetDensityUnit : Integer; stdcall;
```

Returns density unit. See **SetDensityUnit** for a list of possible return values.

```
function GetSpecificHeatUnit : Integer; stdcall;
```

Returns specific heat unit. See **SetSpecificHeatUnit** for a list of possible return values.

```
function GetConductivityUnit : Integer; stdcall;
```

Returns conductivity unit. See **SetConductivityUnit** for a list of possible return values.

```
function GetDynamicViscosityUnit : Integer; stdcall;
```

Returns dynamic viscosity unit. See **SetDynamicViscosityUnit** for a list of possible return values.

```
function GetKinematicViscosityUnit : Integer; stdcall;
```

Returns kinematic viscosity unit. See **SetKinematicViscosityUnit** for a list of possible return values.

```
function GetEnthalpyUnit : Integer; stdcall;
```

Returns enthalpy unit. See **SetEnthalpyUnit** for a list of possible return values.

```
function GetTemperatureUnit : Integer; stdcall;
```

Returns temperature unit. See **SetTemperatureUnit** for a list of possible return values.

```
function GetPressureUnit : Integer; stdcall;
```

Returns pressure unit. See **SetPressureUnit** for a list of possible return values.

```
function GetPressureDropUnit : Integer; stdcall;
```

Returns pressure drop unit. See **SetPressureDropUnit** for a list of possible return values.

```
function GetHeatTransferCoeffUnit : Integer; stdcall;
```

Returns heat-transfer coefficient unit. See **SetHeatTransferCoeffUnit** for a list of possible return values.

```
function GetDiameterUnit : Integer; stdcall;
```

Returns diameter unit. See **SetDiameterUnit** for a list of possible return values.

```
function GetLengthUnit : Integer; stdcall;
```

Returns length unit. See **SetLengthUnit** for a list of possible return values.

```
function GetRoughnessUnit : Integer; stdcall;
```

Returns roughness unit. See **SetRoughnessUnit** for a list of possible return values.

```
function GetVelocityUnit : Integer; stdcall;
```

Returns velocity unit. See **SetVelocityUnit** for a list of possible return values.

7 References

[1] Skovrup, Morten J – *Heat-Transfer Efficiency Factor*. DESIK 2004-06-01.

[2] Fox,R.W.; McDonald,A.T. – *Introduction to Fluid Mechanics*; Third Edition; John Wiley & Sons, Inc. 1985

[3] Incropera, Frank P.; DeWitt, David P. –*Introduction to Heat Transfer*. Wiley 2002, fourth edition.

[4] Müller-Steinhagen, H.; Heck, K. – *A Simple Pressure Drop Correlation for Two-Phase Flow in Pipes*; Chem. Eng. Process, 20, 1986. Page 297-308

[5] Hansen, Torben et.al.. –*Pressure Drop Correlation for Ice-slurry*; Internal communication

[6] Melinder, Åke – *Thermophysical properties of liquid secondary refrigerants*; IIR, France, 1997