CHEMCAD CALCULATOR/PARSER MODULE

GENERAL INFORMATION

The Calculator/Parser module in CHEMCAD is a powerful utility module, which offers the power and flexibility of the high level C-like language without having to use a compiler. The module allows the user to do the following things:

- Create your own unit operation within a flowsheet.
- Retrieve stream/equipment/component information in a flowsheet.
- Perform mathematical calculation. Most mathematical functions are built into the standard library.
- The "for statement" for loop calculation, the "if else" statement for information control and the "goto" statement are allowed in the module.
- All flash functions, physical property routines and many other useful functions can be called to perform necessary calculations.
- Print the calculated results by the "printf" statement.
- Put the calculated results back to the flowsheet.

Typical applications of the calculator/parser module include:

- Calculate stream compositions or property for use by the controller.
- Calculate size/cost of process equipment.
- Calculate the flowrate of the makeup streams for better convergence.
- Simulate a special unit operation as required.

The usefulness of the module is limited only by the imagination of the user.

COSTING CALCULATIONS

All of the cost calculating routines in CHEMCAD are done using the Parser language described here. This is to enable users to modify the costing routines for their own use. The source code for these calculations is stored in the CHEMCAD program directory (typically C:\CC5) and have the following names:

$$$aclr.sf	Air cooled heat exchanger costing
$$$cfug.sf	Centrifuge costing
$$$comp.sf	Compressor costing
$$$crsh.sf	Crusher costing
$$$crys.sf	Crystallizer costing
$$$cycl.sf	Cyclone costing
$$$dpip.sf	Double pipe heat exchanger costing
$$$evap.sf	Evaporator heat exchanger costing
$$$expn.sf	Expander costing

$$$fire.sf	Fired heater costing
$$$flas.sf	Flash (vessel) costing
$$$fltr.sf	Filter costing
$$$htxr.sf	Shell and tube heat exchanger costing
$$$pump.sf	Pump costing
$$$rfrg.sf	Refrigeration costing
$$$scds.sf	SCDS distillation column costing
$$$scre.sf	Screen costing
$$$towr.sf	TOWR distillation costing
$$$tpls.sf	TPLS distillation costing
$$$vesl.sf	Vessel costing

DIALOG BOXES FOR PARSER UNITS

If your CALC unitop needs to use a dialog screen, it must be built according to the Screen Builder manual and documentation. Please see the screen builder documentation for more details.

For a Parser unit op, the screen files will be named in the same format as the program file, as "CALC#.MY" and "CALC#.MAP" where # is the unit ID number. For example, if your unit id number is 5, then the files would be named "calc5.my" and "calc5.map". Please see the screen builder section of this manual for more details on these files.

PARSER LANGUAGE SYNTAX AND STRUCTURE

The Calculator/Parser module uses a C-like language to perform all calculations. It is recommended that the user be familiar with some of the basic C language syntax before writing the code for this module. Several examples at the end of this chapter should be studied before you proceed.

Data types:

int	integer
float	floating point real numbers
double	double precision real numbers

Examples:

```
int      i, j, k;
double   x, y, z;
float    a, b, c;
```

Both one dimensional array and two dimensional arrays are supported.

Examples:

```
int      ix[100];
double   x[50];
float    uspec[300];
float    matirx[10][10];
```

Intrinsic functions:

+	add
-	subtract
*	multiply
/	divide
exp	exponential
log	natural logarithm (Ln)
log10	common logaruthm
pow(x,y)	x to the power of y
fabs	absolute value of a real number
abs	absolute value of an integer
sqrt	square root of a real number
sin	Sine(radians)
cos	Cosine(radians)
tan	Tangent(radians)
asin	Arc Sine(radians)
acos	Arc cosine(radians)
atan	Arc tangent(radians)
==	equal
>	greater than
>=	greater or equal
<	less than
<=	less or equal
&&	and
||	or

Loop calculations:

The "for statement" can be used for loop calculation.

Example:

```
int       i, n;

n = 30;
for( i = 0; i < n; i++)
{
     /*  Program code */

     ….

}
```

Up to two nested loops are allowed.

Example:

```
int       i, j;
for( i = 0; i < 10; i++)
{
     for( j = 0; j < 20; j++)
```

```
        {
           ….
        }
    }
```

Information control:

- if (expression) expression;

  Example:

  ```
  a = 0.;
  if( x > 20. ) a = 1.;
  ```

- if(expression)

  ```
  {
      expressions;
  }
  ```

  Example:

  ```
  a = 0.;
  b = 0.;
  if( x > 20. )
  {
      a = 1.;
      b = 2.;
  }
  ```

- if(expression)

  ```
  {
      expressions;
  }
  else if( expression )
  {
      expressions;
  }
  else if( expression )
  {
      expressions;
  }
  else
  {
      expressions;
  }
  ```

  Example:

  ```
  a = 0.;
  b = 0.;
  if( x > 10. )
  {
  ```

```
    a = 1.;
    b = 1.;
}
else if( x > 20. )
{
    a = 2.;
    b = 2.;
}
else if( x > 30. )
{
    a = 3.;
    b = 3.;
}
else
{
    a = 4.;
    b = 4.;
}
```

Note: *Nested "if else" up to two levels are allowed.*

Example:

```
if( a > (b * 23.))
{
    if( c > 25. )
    {
        …..
    }
}
else
{
    ….
}
```

- goto LABEL;

Example:

```
    x = 23 * y;
    if( x > 100. ) goto THERE;
    ….
    …
THERE:
    …
```

Note: *The goto statement can only be used to jump to the statement below to goto line. It cannot be used to jump back to the previous statement.*

printf statement

printf statement can be used similar to the C language.
Up to 5 arguments can be included in one printf statement.

For debugging purpose, it is recommended that the user use the "Generate Run History" checkbox on the "Convergence" screen of the "Run" menu so that the printf and any error messages can be seen on the screen when you run the calculator module.

Example:

```
printf("Calculate area = %g, heat duty = %g\n",area,duty);
printf("Stream %d, Temperature = %13.2f\n",str_ID,temp);
```

Return Values of the Arguments:

In the standard C syntax, the return value of an argument in a function call should be passed by the address. This is omitted in the calculator module as illustrated by the following example.

Standard C:

```
ProgramX()
{
    float   input_data;
    float   output_data;

    /*  Notice the address sign (&) for the output data */

    functionX(input_data,&output_data);

}
```

CHEMCAD Calculator:

```
CHEMCAD_Link Program_Name
{
    float   input_data;
    float   output_data;

    /*  Notice the address sign (&) is omitted here */

    functionX(input_data,output_data);
}
```

PROGRAM STRUCTURE

A program file has to be created in the job subdirectory with the file name CALC#.TXT where # is the equipment ID number of the calculator. For example, if the calculator number is 5, you need to write a program CALC5.TXT and put this file in your job subdirectory. The program file has to have the following structure:

```
CHEMCAD_Link Program_Name
{
    /*  Section 1 : Variable declarations */
    ..... ;
```

```
        ..... ;
        /*   Sections 2: Perform all calculations */
        ..... ;
        ..... ;
    }
```

Creating a dialog screen for a Parser (Calculator) unit op

If your unit has any settings, such as output pressure for a pump or # of stages for a distillation column, you will need to generate a dialog box. The construction of dialogs is covered in the Screen Builder section of this manual.

For a Parser unit op, the screen files will be named in the same format as the program file, as “CALC#.MY” and “CALC#.MAP” where # is the unit ID number. For example, if your unit id number is 5, then the files would be named “calc5.my” and “calc5.map”. Please see the screen builder section of this manual for more details on these files.

CHEMCAD LIBRARY FUNCTIONS

The following functions may be accessed from a Parser unit. They fall into the general categories of:

Flowsheet Interface: Functions to move data from the flowsheet into the calculator and back

Thermodynamics/Engineering Calcs Functions used to calculate properties of mixtures

FLOWSHEET INTERFACE FUNCTIONS:

These functions are used to pass information into the calculator function and back.

CC_Get_Equip_Parameters-returns the unitop array for a specified unit

CC_Put_Equip_Parameters-overwrites the unitop array for a specified unit

CC_Get_Process_Stream-returns a flowsheet stream data

CC_Put_Process_Stream-overwrites a flowsheet stream with new data

CC_Get_Input_Stream-returns stream data for a stream connected to the Calculator unit's inlet

CC_Put_Inlet_Stream-overwrites the stream data for a stream connected to the Calculator unit's inlet

CC_Get_Outlet_Stream-returns stream data for a stream connected as an outlet to the Calculator unit.

CC_Put_Outlet_Stream-overwrites the stream data for a stream connected to the Calculator unit's outlet

CC_No_Of_Components-returns the number of components

CC_Int_Array_Function-returns an array of flowsheet information

CC_Get_Stream_Property-returns a property of a given stream

CC_Get_Value-generic interface to pure component, stream and unit data

Function: CC_Get_Equip_Parameters

Get equipment parameters for a specified equipment ID number.

Prototype: int CC_Get_Equip_Parameters(int id, float uspec[])

Parameters:

input   id   equipment ID
if id = 0, the current (active) equipment parameters in the memory will be retrieved.
output uspec[]   an array stores all parameters for the specified equipment

Return Value:

= 0 normal return
= 1 error

Description:

A float array of uspec[300] should be declared in the file before this function is called. After calling CC_Get_Equip_Parameters the array uspec[300] should store all parameters in internal units for this equipment.

Example:

```
/*   Get heat exchanger area and LMTD for equipment 12 */

CHEMCAD_Link Parser
{
     float     uspec[300];
     float     area_ft2, LMTD;

     /*   Get parameters for equipment 12 which is an exchanger */
     CC_Get_Equip_Parameters(12,uspec);

/*   Area (var. no = 29) in ft2 */
area_ft2 = uspec[29];

/*   LMTD (var. no = 22) in deg R */
LMTD = uspec[22];

......

}
```

Function: CC_Put_Equip_Parameters

Put equipment parameters for a specified equipment ID number into the flowsheet

Prototype: int CC_Put_Equip_Parameters(int id, float uspec[])

Parameters:

input id equipment ID
if id = 0, the current (active) equipment parameters in the memory will be updated
uspec[] an array stores all parameters for the specified equipment

Return Value:

=0 normal return
= 1 error

Description:

A float array of uspec[300] should be declared in the file before this function is called. The array uspec[300] should store all parameters in internal units.

Example:

```
/*   Calculate basic heat exchanger cost for equipment 12 */

CHEMCAD_Link Parser
{
    float    uspec[300];
    float    area_ft2, log_area, bcost;

/*   Get parameters for equipment 12 which is an exchanger */
CC_Get_Equip_Parameters(12,uspec);

/*   Area (var. no = 29) in ft2 */
area_ft2 = uspec[29];

/*   Log area */
log_area = log(area_ft2);

/*   Calc base cost of exchanger */
bcost   =    exp(8.821 - 0.30863 * log_area +

          0.0681 * log_area * log_area);

/*   Put cost(var. no 48) back to htxr */
uspec[48] = bcost;
CC_Put_Equip_Parameters(12,uspec);

}
```

Function: CC_Get_Process_Stream

Get component flowrates, temperature, pressure, vapor fraction and enthalpy of a specified stream.

Prototype: int CC_Get_Process_Stream(int id, float xmol[], float t, float p, float vf, float h)

Parameters:

input	id	stream ID
output	xmol[]	An array stores component flowrates in lbmol/hr
	t	temperature in R
	p	pressure in psia
	vf	mole vapor fraction
	h	enthalpy in btu/hr

Return Value:

= 0 normal return
= 1 error

Description:

A valid stream ID > 0 must be specified. Xmol[] must have a dimension equal to or greater than the number of components in the system.

Example:

```
/*  Get stream 20 information */

CHEMCAD_Link Parser
{
    float    xmol[50];
    float    t, p, vf, h;

    /*  Get stream 1 information */
    CC_Get_Process_Stream(20,xmol, t, p, vf, h);

.....
}
```

Function: CC_Put_Process_Stream

Put component flowrates, temperature, pressure, vapor fraction and enthalpy of a specified stream back to the flowsheet.

Prototype: int   CC_Put_Process_Stream(int id, float xmol[], float t, float p, float vf, float h)

Parameters:

```
input   id   stream ID
        xmol[]  An array stores component flowrates
             in lbmol/hr
        t    temperature in R
        p    pressure  in psia
        vf   mole vapor fraction
        h    enthalpy in btu/hr
```

Return Value:

= 0	normal return
= 1	error

Description:

A valid stream ID > 0 must be specified. Xmol[] must have a dimension equal to or greater than the number of components in the system.

Example:

```
/*  Double the flowrate and enthalpy of stream 20 */

CHEMCAD_Link Parser
{
    float    xmol[50];
    float    t, p, vf, h;
    int  nc, i;

    /*  Get stream 1 information */
```

```
        CC_Get_Process_Stream(20, xmol, t, p, vf, h);

        /*  Get the no of components in the system */
        nc = CC_No_Of_Components();

        /*  Double the component flowrate */
        for( i = 0; i < nc; i++)
        {
        xmol[i] = 2. * xmol[i];
        }

        /*  Double the enthalpy */
        h = 2. * h;

        /*  Put the calculated information back to stream 20 */
        CC_Put_Process_Stream(20, xmol, t, p, vf, h);

    }
```

Function: CC_Get_Input_Stream

Get component flowrates, temperature, pressure, vapor fraction and enthalpy of a input stream for the calculator module.

Prototype: int   CC_Get_Input_Stream(int ith_input, float xmol[], float t, float p, float vf, float h)

Parameters:

input	ith_input	The ith input stream of the calculator base 1.
output	xmol[]	An array stores component flowrates in lbmol/hr
	t	temperature in R
	p	pressure in psia
	vf	mole vapor fraction
	h	enthalpy in btu/hr

Return Value:

= 0	normal return
= 1	error

Description:

Xmol[] must have a dimension equal to or greater than the number of components in the system.

Example:

```
    /*  Get the first input stream information */

    CHEMCAD_Link Parser
    {
        float     xmol[50];
        float    t, p, vf, h;

        /*  Get stream 1 information */
        CC_Get_Input_Stream(1,xmol, t, p, vf, h);

        .....

    }
```

Function: CC_Get_Output_Stream

Get component flowrates, temperature, pressure, vapor fraction and enthalpy of a output stream for the calculator module.

Prototype: int   CC_Get_Output_Stream(int ith_output, float xmol[], float t, float p, float vf, float h)

Parameters:

input		ith_output The ith output stream of the calculator base 1.
output		xmol[] An array stores component flowrates in lbmol/hr
	t	temperature in R
	p	pressure in psia
	vf	mole vapor fraction
	h	enthalpy in btu/hr

Return Value:

= 0	normal return
= 1	error

Description:

Xmol[] must have a dimension equal to or greater than the number of components in the system.

Example:

```
    /*  Get the first output stream information */

    CHEMCAD_Link Parser
    {
        float    xmol[50];
        float    t, p, vf, h;

        /*  Get stream 1 information */
        CC_Get_Output_Stream(1,xmol, t, p, vf, h);

        .....

    }
```

Function: CC_Put_Input_Stream

Put component flowrates, temperature, pressure, vapor fraction and enthalpy of a input stream back to the flowsheet.

Prototype: int CC_Put_Input_Stream(int ith_input, float xmol[], float t, float p, float vf, float h)

Parameters:

input ith_input The ith input stream of the calculator base 1.

xmol[]	An array stores component flowrates in lbmol/hr
t	temperature in R
p	pressure in psia
vf	mole vapor fraction
h	enthalpy in btu/hr

Return Value:

= 0	normal return
= 1	error

Description:

Xmol[] must have a dimension equal to or greater than the number of components in the system.

Function: CC_Put_Output_Stream

Put component flowrates, temperature, pressure, vapor fraction and enthalpy of a output stream back to the flowsheet.

Prototype: int CC_Put_Output_Stream(int ith_output, float xmol[], float t, float p, float vf, float h)

Parameters:

input ith_output The ith output stream of the calculator base 1.

xmol[]	An array stores component flowrates in lbmol/hr
t	temperature in R
p	pressure in psia
vf	mole vapor fraction
h	enthalpy in btu/hr

Return Value:

= 0	normal return
= 1	error

Description:

Xmol[] must have a dimension equal to or greater than the number of components in the system.

Example:

/* Simulate a 50/50 split divider */

CHEMCAD_Link Parser

```
{
    float    xmol[50];
    float    t, p, vf, h;
    int  nc, i;

    /*  Get stream 1 information */
    CC_Get_Input_Stream(1,xmol, t, p, vf, h);

    /*  Get the no of components in the system */
    nc = CC_No_Of_Components();

    /*  Double the component flowrate */
    for( i = 0; i < nc; i++)
    {
        xmol[i] = 0.5 * xmol[i];
    }

    /*  Double the enthalpy */
    h = 0.5 * h;

    /*  Put the calculated information to output stream 1 */
    CC_Put_Output_Stream(1, xmol, t, p, vf, h);

    /*  Put the calculated information to output stream 2 */
    CC_Put_Output_Stream(2, xmol, t, p, vf, h);

}
```

Function: CC_No_Of_Components

Get the number of components in the system.

Prototype: int   CC_No_Of_Components()

Parameters:

input   None

Return Value:

No of components in the system

Description:

This function returns the number of components in the system.

Example:

```
CHEMCAD_Link Parser
{
    int  nc;

    /*  Get the no of components in the system */
    nc = CC_No_Of_Components();
```

```
        ......
    }
```

Function: CC_Get_Stream_Property

Get the property of a stream.

Prototype: float Get_Stream_Property(float xmol[], double t, double p, double vf, double h, int iprop)

Parameters:

input	xmol[]	An array stores component flowrates in lbmol/hr
	t	temperature in R
	p	pressure in psia
	vf	mole vapor fraction
	h	enthalpy in btu/hr
	iprop	property flag

1	= Temperature
2	= Pressure
3	= Mole vap frac
4	= Enthalpy
5	= Tot. mole rate
6	= Tot. mass rate
7	= Total std liq.
8	= Total std vap.
9	= Total act. vol
10	= Tot. act. dens
11	= Total Mw
12	= Gross H value
13	= Net H value
14	= Reid vapor P
15	= UOPK
16	= VABP
17	= MeABP
18	= Flash point
19	= Pour point
20	= Total entropy
21	= Mass vap frac
22	= PH value
23	= Total resv2
24	= Total resv3
25	= Total resv4
26	= Vap mole rate
27	= Vap mass rate
28	= Vapor enthalpy
29	= Vapor entropy

30	= Vapor Mw
31	= Vap act. dens
32	= Vap act. vol.
33	= Vap std. liq.
34	= Vap std. vap.
35	= Vapor cp
36	= Vapor Z factor
37	= Vap viscosity
38	= Vapor th. cond
39	= Vapor resv1
40	= Vapor resv2
41	= Liq. mole rate
42	= Liq. mass rate
43	= Liq. enthalpy
44	= Liq. entropy
45	= Liq. Mw
46	= Liq. act. dens
47	= Liq. act. vol.
48	= Liq. std. liq.
49	= Liq. std. vap.
50	= Liq. cp
51	= Liq. Z factor
52	= Liq. viscosity
53	= Liq. th. cond.
54	= Liq. surftens

= -i , flow rate of ith component
(lbmol/hr)
= -(i+200) ith comp mass flow rate(lb/hr)
= -(i+400) ith comp std vol rate(ft3/hr)
= -(i+600) ith comp mole frac
= -(i+800) ith comp mass frac
= -(i+1000) ith comp vol. frac

Return Value:

The desired property in internal unit.

temperature	=	R
pressure	=	psia
mole flow rate	=	lbmol/hr
mass flow rate	=	lb/hr
enthalpy	=	btu/hr
work	=	btu/hr
entropy	=	btu/R/hr
vol flow rate	=	ft3/hr
density	=	lb/ft3
viscosity	=	cp
surface tens	=	dyne/cm

therm. Cond	=	Btu/hr ft2 F / ft
area	=	ft2
length	=	ft

Description:

This function returns the property of a stream with given component flowrate, temperature, pressure, vapor fraction and enthalpy. The return value is always in internal unit.

Example:

```
/*  Get the liquid viscosity of the first input stream in the calculator module */

CHEMCAD_Link Parser
{
    float    xmol[50];
    float    t, p, vf, h;
    float    liq_vis;

    /*  Get stream 1 information */
    CC_Get_Input_Stream(1,xmol, t, p, vf, h);

    /*  Get liquid viscosity */
    liq_vis = CC_Get_Stream_Property(xmol,t,p,vf,h,52);

    .....

}
```

Function: CC_Get_Value

Get a value of the system.

Prototype: double CC_Get_Value(int id1, int id2, int id3, int id4)

Parameters:

input:

```
    if id1 = 1   Pure comp data
            id2 = 1   Mw
            id2 = 2   Tc
            id2 = 3   Pc
            id2 = 4   accentric factor

            id3 = comp position, base 1
    if id1 = 2   stream property data
            id2 = stream ID
            id3 = property flag, See CC_Get_Stream_Property
    if id1 = 3   Get equipment input/output stream ID
            id2 = equip ID
            id3 > 0  (id3)th input stream ID
            id3 < 0 -(id3)th output stream ID
```

```
if id1 = 4   Get cost index
        id2 = equip type, see $index.sf in cc3
        id3 = 0   Base cost index
        = 1   Current cost index

if id1=7     Get dynamic time in minutes
```

Return Value:

The desired value from the system

Description:

This is a generalized function that lets you get a single value from the process including pure component properties, stream properties, and unit operation input/output stream numbers.

Example:

```
CHEMCAD_Link Parser
{
    int  nc, i;
    float     Mw[50];
    float     total_vol;

    /*  Get the no of components in the system */
    nc = CC_No_Of_Components();

    /*  Get the molecular weight of each component */
    for( i = 0; i < nc; i++)
    {
        /*  Note: i+1 below is the component position base 1 */
        Mw[i] = CC_Get_Value(1,1,i+1,0);
    }

    /*  Get the actual total volumetric flowrate of stream 3 */
    total_vol = CC_Get_Value(2,3,9,0);

    ...

}
```

Function: CC_Int_Array_Function

General integer array function

Prototype: int CC_Int_Array_Function(int id1, int id2, int id3, int intArray[])

Parameter:

id1 = 1 Get all streams in flowsheet
id2, id3 not used
return intArray[] contains all stream ID's
return value is no of element in intArray

id1 = 2 Get equipment IDs in flowsheet
id2, id3 not used
return intArray[] contains all equipment ID's
return value is no of element in intArray
id1 = 3 Not used
id1 = 4 Get component IDs
id2, id3 not used
return intArray[] contains all component ID's
return value is no of element in intArray

Return Value: Return value is no of element in intArray.

Description:

This is a generalized function that lets you get an array of integers from the flowsheet. Depends on the id1 value, the function returns stream numbers, equipment numbers or component ID numbers in intArray[].

THERMODYNAMICS/ENGINEERING FUNCTIONS:

Used to calculate the properties of mixtures

CC_tpflash-constant temperature and pressure equilibrium

CC_vpflash-constant vapor fraction and pressure equilibrium

CC_vtflash-constant vapor fraction and temperature equilibrium

CC_hspflash-adiabatic/isentropic equilibrium

CC_keq-returns K values (Yfraction/X fraction) for a mixture

CC_kxeq-returns K-values using ADDK for a mixture

CC-??????

CC_enthalpy-returns enthalpy for a mixture at conditions

CC_hxstream-returns enthalpy cased on ADDH?

CC_entropy-returns entropy calculation for a given mixture

CC_zfactor-returns compressability (z) factor for a mixture

CC_ldense-returns liquid density

CC_vdense-returns vapor density

CC_vp-returns pure component vapor pressure

CC_cp-returns heat capacity of a mixture

Cc_cv-returns cv of a mixture

CC_lvisco-returns liquid viscosity of a mixture

CC_gvisco-returns vapor viscosity of a mixture

CC_lthc-returns liquid thermal conductivity of a mixture

CC_vthc-returns vapor thermal conductivity of a mixture

CC_surften-returns the surface tension of a mixture

Function: CC_tpflash

Isothermal flash calculation at specified temperature and pressure

Prototype: int CC_tpflash(float feed[], double hin, double tr, double psia,
float vapor[], float xliq[], float totalv,
float totall, float hvap, float hliq, float htotal,
float vout, float xk[], float delq, float f_ions[])

Parameters:

Input:	feed[]	lbmole/hr
	hin	btu/hr
	tr	flash T in degree R
	psia	flash P in psia

Output:	vapor[]	vapor out lbmole/hr
	xliq[]	liquid out lbmole/hr
	totalv	total vapor flow lbmole/hr
	totall	total liquid flow lbmol/hr
	hvap	vapor enthalpy btu/hr
	hliq	liquid enthalpy btu/hr
	htotal	total H in output streams btu/hr
	vout	vapor fraction
	xk[]	K values
	delq	heat duty = htotal - hin
	f_ions[]	ions flow rate lbmol/hr

Return Value int ret_val = 0 Normal return
= 1 Diverge

Description:

This program calculates isothermal flash at given T(R) and P(psia). The input enthalpy (hin) should be non-zero if enthalpy calculation is needed. If the electrolyte package is chosen, the routine also returns the flow rate of ions in the liquid phase.

Function: CC_vpflash
Flash calculation at specified mole vapor fraction and pressure

```
Prototype:  int  CC_vpflash(float feed[], double v, double p, double hin, double testi,
            int  irelod, float vapor[], float xliq[], float totalv,
            float totall, float tout, float hvap, float hliq,
            float htotal, float xk[], float delq, float f_ions[])
```

Parameters:

```
    Input : feed[]   lbmole/hr
        v    vapor fraction
        p    pressure psia
        hin feed enthalpy Btu/hr
        if = 0, no heat duty will be calculated
        testi    estimated flash temperature (R)
        irelod   = 0 Start from scratch.
                 = 1 Reload K values from xk[],

    Output: vapor[]     vapor product lbmole/hr
        xliq[]      liquid product lbmole/hr
        totalv      total vapor rate lbmole/hr
        totall      total liquid rate lbmole/hr
        tout        flash output temperature (R)
        hvap        vapor enthalpy  Btu/hr
        hliq        liquid enthalpy Btu/hr
        htotal      total output enthalpy Btu/hr
        xk[]        flash K values
        delq        heat duty Btu/hr
        f_ions[]    ions flow rate lbmol/hr

Return Value  int  ret_val = 0 Normal retur
                   = 1     Diverge
```

Description:

This program performs flash calculation at given vapor fraction and pressure.
The input enthalpy (hin) should be non-zero if enthalpy calculation is needed.
If input vapor fraction = 0, the routine calculates the bubble point temperature.
If input vapor fraction = 1, the routine calculates the dew point temperature.

Estimated output temperature must be given.

If the electrolyte package is chosen, the routine also returns the flow rate of ions in the liquid phase.

Function: CC_vtflash
Flash calculation at specified mole vapor fraction and temperature

```
Prototype: int  CC_vtflash(float feed[], double vfrac, double t, double hin,
        double pesti, int  irelod, float vapor[], float xliq[],
        float totalv, float totall, float pout, float hvap,
        float hliq, float htotal, float xk[], float delq,
        float f_ions[])
```

Parameters:

Input :	feed[]	lbmole/hr
	vfrac	vapor fraction
	t	flash temperature (R)
	hin	feed enthalpy Btu/hr
	pesti	estimated flash pressure (psia)
	irelod	= 0, Start from scratch.
		= 1, Reload K values.
Output:	vapor[]	vapor product lbmole/hr
	xliq[]	liquid product lbmole/hr
	totalv	total vapor rate lbmole/hr
	totall	total liquid rate lbmole/hr
	pout	flash output pressure (psia)
	hvap	vapor enthalpy Btu/hr
	hliq	liquid enthalpy Btu/hr
	htotal	total output enthalpy Btu/hr
	xk[]	flash K values
	delq	heat duty Btu/hr
	f_ions[]	ions flow rate lbmol/hr

Return Value int ret_val = 0 Normal return
= 1 Diverge

Description:

This program performs flash calculation at given vapor fraction and temperature.
The input enthalpy (hin) should be non-zero if enthalpy calculation is needed.
If input vapor fraction = 0, the routine calculates the bubble point pressure.
If input vapor fraction = 1, the routine calculates the dew point pressure.
Estimated output pressure must be given.
If the electrolyte package is chosen, the routine also returns the flow rate of ions in the liquid phase.

Function: CC_hspflash
Adiabatic/Isentropic flash calculation at specified pressure

Prototype: int CC_hspflash(float feed[], double p, double hsin, int mode,

```
    double testi, float vapor[], float xliq[], float totalv,
    float totall, float tout, float hvap, float hliq,
    float htotal, float xk[], float v, float f_ions[])
```

Parameters:

Input : feed[] lbmole/hr

p	pressure psia
hsin,mode	H (mode=0)Btu/hr or S(mode=1)Btu/R/hr
testi	estimated flash temperature (R)

Output:

vapor[]	vapor product lbmole/hr
xliq[]	liquid product lbmole/hr
totalv	total vapor rate lbmole/hr
totall	total liquid rate lbmole/hr
tout	flash output temperature (R)
hvap	vapor enthalpy Btu/hr
hliq	liquid enthalpy Btu/hr
htotal	total output enthalpy Btu/hr
xk[]	flash K values
v	output vapor fraction
f_ions[]	ions flow rate lbmol/hr

Return Value int ret_val = 0 Normal return
= 1 Diverge

Description:

For adiabatic flash (Constant enthaly) calculation, set mode = 0.
For isentropic flash (Constant entropy) calculation, set mode = 1.

Parameter hsin is feed enthalpy for adiabatic calculation.
Parameter hsin is feed entropy for adiabatic calculation.

Estimated output temperature must be given.

If the electrolyte package is chosen, the routine also returns the flow rate of ions in the liquid phase.

Function: CC_keq
K value calculation routine

Prototype: void CC_keq(float yv[], float xl[], double t, double p,
float xkv[], float f_ions[])

Parameters:

Input:

yv[]	vapor mole flow rate (lbmol/hr)
xl[]	liquid mole flow rate(lbmol/hr)
t	stream temperature in degree R
p	stream pressure in psia

Output:

xkv[]	K values
f_ions[]	ions flow rate in liquid phase, lbmol/hr

Return Value: void

Description:

For given vapor composition, liquid composition, temperature and pressure, this routine calculates the equilibrium K value for each component according to the K model selected by the user. If electrolyte package is chosen, the routine also returns the flow rate of ions in the liquid phase.

Function: CC_kxeq
K value calculation routine called from user added K value routine (ADDK)

Prototype: void CC_kxeq(float yv[], float xl[], double t, double p, float xkv[])

Parameters:

Input:	yv[]	vapor mole flow rate (lbmol/hr)
	xl[]	liquid mole flow rate(lbmol/hr)
	t	stream temperature in degree R
	p	stream pressure in psia
Output:	xkv[]	K values

Return Value: void

Description:

Calculates K values for any thermo method. For given vapor composition, liquid composition, temperature and pressure, this routine calculates the equilibrium K value for each component according to the external variable modek.

The following table shows the value of modek and corresponding model:

modek	K model
1	Polynomial K
2	Grayson Streed
3	SRK
4	API SRK
5	UNIFAC
6	K table
7	Wilson
8	Ideal Vapor Pressure
9	Peng-Robinson
10	NRTL
11	ESSO
12	Amine
13	Sour Water
14	UNIQUAC
15	Margules
16	Regular Solution
17	Van Laar
18	ADDK
19	Henry's law

20	Flory Huggins
21	UNIFAC Polymers
22	MSRK
23	PPAQ
24	TSRK
25	TEG Dehydration
26	ACTX
27	T-K Wilson
28	HRNM modified Wilson
29	PSRK
30	GMAC (Chien-Null)
31	UNQC (Uniquac with Unifac R and Q)
32	UNIFAC LLE

Function: CC_enthalpy
Enthalpy calculation routine.

Prototype: double CC_enthalpy(float xmol[], double t, double p, int iphase, float hx)

Parameters:

Input:	xmol[]	component mole flow rate(lbmole/hr)
	t	temperature in degree R
	p	pressure in psia
	iphase	0 = Liquid ,1 = Vapor

Output hx Btu/hr

Return Value: function return value is enthalpy in double precision.

Description:

For given component flow rate, temperature and pressure, this routine calculates the enthalpy of the stream according to the enthalpy model selected by the user. This routine only calculates the enthalpy of a single phase stream. For a two-phase stream, the flash routine such as tpflash should be used to determine the overall stream enthalpy.

Function: CC_hxstream
Enthalpy calculation routine called from ADDH.

Prototype: void CC_hxstream(float xmol[], double t, double p, int iphase, float hx)

Parameters:

Input:	xmol[]	component mole flow rate(lbmole/hr)
	t	temperature in degree R
	p	pressure in psia
	iphase	0 = Liquid ,1 = Vapor

Output hx Btu/hr

Return Value: void

Description:

Calculates enthalpy based on any enthalpy model. For given component flow rate, temperature and pressure, this routine calculates the enthalpy of the stream according to the external variable modeh.

The following table shows the value of modeh and corresponding model:

modeh	H model
1	Polynomial H
2	Redlich-Kwong
3	SRK
4	API SRK
5	Peng-Robinson
6	Lee-Kesler
7	Latent Heat
8	Amine
9	No Enthalpy
10	Enthalpy Table
11	ADDH
12	Mixed Model

Function: CC_entropy
Entropy calculation routine.

Prototype: void CC_entropy(float xmol[], double t, double p, int iphase, float sx)

Parameters:

Input:	xmol[]	component mole flow rate(lbmole/hr)
	t	temperature in degree R
	p	pressure in psia
	iphase	0 = Liquid ,1 = Vapor
Output	sx	Btu/R/hr

Return Value: void

Description:

For given component flow rate, temperature and pressure, this routine calculates the entropy of the stream.

Function: CC_zfactor
Z factor calculation routine.

Prototype: void CC_zfactor(float xmol[], double t, double p, int iphase, float z, int ierr)

Parameters:

Input:	xmol[]	component mole flow rate(lbmole/hr)
	t	temperature in degree R
	p	pressure in psia
	iphase	0 = Liquid ,1 = Vapor

Output z   Z factor
Ierr   = 0 normal return
> 0 error

Return Value: void

Description:

For given component flow rate, temperature and pressure, this routine calculates Z factor of the stream.

Function: CC_ldense
Liquid density routine.

Prototype: void CC_ldense(float xliq[], double t, double p, float dens)

Parameters:

Input :	xliq[]	component mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Output: dens   liquid density in lb/ft3

Return Value: void

Description:

For given component flow rate, temperature and pressure, this routine calculates the liquid density of the mixture.

Function: CC_vdense
Vapor density routine.

Prototype: void CC_vdense(float xvap[], double t, double p, float dens)

Parameters:

Input :	xvap[]	component mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Output: dens   Vapor density in lb/ft3

Return Value: void

Description:

For given component flow rate, temperature and pressure, this routine calculates the vapor density of the mixture.

Function: CC_vp
Function to calculate the pure component vapor pressure

Prototype: float CC_vp(int i, double t)

Parameters:

Input:	i	component POSITION. (base 0 = first component)
	t	temperature in degree R

Return Value: float, vapor pressure in psia

Description:

Given component i and temperature, this routine calculates the vapor pressure.

Function: CC_cp
Function to calculate the heat capacity of a mixture

Prototype: float CC_cp(float xmol[], double t, double p, int iphase)

Parameters:

Input:

xmol[]	mole flow lbmol/hr
t	temperature R
p	pressure psia
iphase	0 = liquid phase
	1 = vapor phase

Return Value: float cp in Btu/R/lbmole

Description:

Given composition, temperature and pressure, this function calculates the heat capacity of the stream.

Function: CC_cv
Function to calculate the cv of a mixture

Prototype: float CC_cv(float xmol[], double t, double p, int iphase)

Parameters:

Input:

xmol[]	mole flow lbmol/hr

t	temperature R
p	pressure psia
iphase	0 = liquid phase
	1 = vapor phase

Return Value: float cv in Btu/R/lbmole

Description:

Given composition, temperature and pressure, this function calculates the cv of the stream.

Function: CC_lvisco
Function to calculate the liquid viscosity of a mixture

Prototype: float CC_lvisco(float xmol[], double t, double p)

Parameters:

input:	xmol[]	mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Return Value: float viscosity in CP

Description:

Given composition, temperature and pressure, this function calculates the viscosity of the liquid mixture.

Function: CC_gvisco
Function to calculate the vapor viscosity of a mixture

Prototype: float CC_gvisco(float xmol[], double t, double p)

Parameters:

input:	xmol[]	mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Return Value: float viscosity in CP

Description:

Given composition, temperature and pressure, this function calculates the viscosity of the vapor mixture.

Function: CC_lthc
Function to calculate the liquid thermal conductivity of a mixture

Prototype: float CC_lthc(float xmol[], double t, double p)

Parameters:

input:	xmol[]	mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Return Value: float liquid thermal conductivity in Btu/hr ft2 F / ft

Description:

Given composition, temperature and pressure, this function calculates the thermal conductivity of the liquid mixture.

Function: CC_vthc
Function to calculate the vapor thermal conductivity of a mixture

Prototype: float CC_vthc(float xmol[], double t, double p)

Parameters:

input:	xmol[]	mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Return Value: float vapor thermal conductivity in Btu/hr ft2 F / ft

Description:

Given composition, temperature and pressure, this function calculates the thermal conductivity of the vapor mixture.

Function: CC_surften
Function to calculate the liquid surface tension of a mixture

Prototype: float CC_surften(float xmol[], double t, double p)

Parameters:

input:	xmol[]	mole flow rate in lbmol/hr
	t	temperature in R
	p	pressure in psia

Return Value: float liquid surface tension in dyne/cm

Description:

Given composition, temperature and pressure, this function calculates the surface tension of the liquid mixture.

SAMPLE FILES:

```
/*
    Cost estimation for heat exchangers

        Type
            = 0  Fixed head
            = 1  Kettle reboiler
            = 2  U-tube
        Material
            = 0  Carbon steel
            = 1  Stainless steel 316
            = 2  Stainless steel 304
            = 3  Stainless steel 347
            = 4  Nickel 200
            = 5  Monel 400
            = 6  Inconel 600
            = 7  Incoloy 825
            = 8  Titanium
            = 9  Hastelloy

*/

CHEMCAD_Link Parser
{
    int  ID;
    int  type, material;
    float    p_design, fd, g1, g2, fm, p1, p2, fp, C;
    float    area_ft2, log_area, bcost, uspec[300];
    int  in_str1, in_str2;

    type = 0;
    material = 0;

    /*  Get current equip parameters, 0 arg */
    CC_Get_Equip_Parameters(0,uspec);

    bcost = 0.;
    ID = uspec[1];
    printf("Preliminary Shell and Tube Heat Exchanger Cost Estimation\n\n");
    printf("Exchanger Cost for Equip. %d\n",ID);

    /*  Area in ft2 */
    area_ft2 = uspec[29];

    /*  Check range and give warnings */
    if( area_ft2 < 150. )
    {
        printf("Area %g < Amin (150 ft2)\n",area_ft2);
    }
    if( area_ft2 > 12000. )
```

```
{
    printf("Area %g > Amax (12000 ft2)\n",area_ft2);
}

if( area_ft2 > 0. )
{
    /*  Log area */
    log_area = log(area_ft2);

    /*  Calc base cost of exchanger */
    bcost = exp(8.821 - 0.30863 * log_area + 0.0681 * log_area * log_area);

    printf("Area(ft2) = %g\n",area_ft2);
    printf("Base Cost = $%g\n",bcost);

    /*  Calc type correction fd */
    if( type == 2 )
    {
        printf("U-tube\n");
        fd = exp(-0.9816 + 0.0830 * log_area);
    }
    else if( type == 1 )
    {
        printf("Kettle reboiler\n");
        fd = 1.35;
    }
    else
    {
    printf("Fixed head\n");
    fd = exp(-1.1156 + 0.0906 * log_area);
      }

    /*  Calc material correction factor */
    if( material == 1 )
    {
        printf("Material = Stainless steel 316\n");
        g1 = 0.8603;
        g2 = 0.23296;
    }
    else if( material == 2 )
    {
        printf("Material = Stainless steel 304\n");
        g1 = 0.8193;
        g2 = 0.15984;
    }
    else if( material == 3 )
    {
        printf("Material = Stainless steel 347\n");
        g1 = 0.6116;
        g2 = 0.22186;
}
else if( material == 4 )
{
     printf("Material = Nickel 200\n");
     g1 = 1.5092;
     g2 = 0.60859;
}
else if( material == 5 )
{
     printf("Material = Monel 400 \n");
     g1 = 1.2989;
     g2 = 0.43377;
}
else if( material == 6 )
{
     printf("Material = Inconel 600\n");
     g1 = 1.204;
     g2 = 0.50764;
}
else if( material == 7 )
{
     printf("Material = Incoloy 825\n");
     g1 = 1.1854;
     g2 = 0.49706;
}
else if( material == 8 )
{
     printf("Material = Titanium\n");
     g1 = 1.5420;
     g2 = 0.42913;
}
else if( material == 9 )
{
     printf("Material = Hastelloy\n");
     g1 = 0.1549;
     g2 = 0.51774;
}
else
{
     printf("Material = Carbon steel\n");
     g2 = 0.;
     g1 = 1.;
}
fm = g1 + g2 * log_area;

/*  Get 1st inlet stream ID */
in_str1 = CC_Get_Value(3,ID,1,0);
```

```
        /*  Get 2nd inlet stream ID */
        in_str2 = CC_Get_Value(3,ID,2,0);

        /*  Inlet pressure in psig */
        p1 = CC_Get_Value(2,in_str1,2,0) - 14.696;
        p2 = CC_Get_Value(2,in_str2,2,0) - 14.696;

        p_design = p1;
        if( p2 > p1 ) p_design = p2;

        printf("Design pressure (psig) = %g\n",p_design);

        if( p_design <= 300. )
        {
            fp = 0.7771 + 0.04981 * log_area;
        }
        else if( p_design >= 600. )
        {
            fp = 1.1400 + 0.12088 * log_area;
        }
        else
        {
            fp = 1.0305 + 0.07140 * log_area;
        }

        C = fd * fm * fp * bcost;

        printf("Exchanger cost = $%g\n",C);

        /*  Put cost back to htxr */
        uspec[48] = C;

        CC_Put_Equip_Parameters(0,uspec);

        }
        else
        {
            printf("No area data available.\n");
        }
    }
/*  Sample calculator program

    This program simulate a component separator

    The flow rate, temperature, and pressure of the second output stream are specified and fixed.  The
    program put the difference of the input stream and second output stream into the first output stream
    and perform an adiabatic flash calculation.

*/

CHEMCAD_Link CALC1

{
    int   nc, i, j;

    float in1[100];
    float tin;
    float pin;
    float vin;
    float hin;

    float out1[100], out2[100];
    float tv, tl, hv, hl;
    float t1out, t2out, p1out, p2out, h1out, h2out, v1out, v2out;

    float vapor[100], xliq[100], xk[100], f_ions[100];

    //  Get input stream
    CC_Get_Input_Stream(1, in1, tin, pin, vin, hin);

    //  Get 2nd output stream
    CC_Get_Output_Stream(2, out2, t2out, p2out, v2out, h2out);

    //  Calc the difference and put in first output stream
    nc = CC_No_Of_Components();

    for( i = 0; i < nc; i++)
    {
        out1[i] = in1[i] - out2[i];
        if( out1[i] < 0.)
        {
            printf("Error comp %d, output > input\n",i);
            out1[i] = 0.;
        }
    }

    t1out = tin;
    p1out = pin;

    //  Assume adiabatic
    h1out = hin - h2out;

    //  Call adiabatic flash calculation to get the output temperature
    //  and vapor fraction
    CC_hspflash(out1, p1out, h1out, 0, tin, vapor,xliq, tv, tl, t1out, hv,
        hl, h1out, xk, v1out,f_ions);

    //  Put the results to output 1
    CC_Put_Output_Stream(1, out1,t1out, p1out, v1out, h1out);

    }

/*  Sample calculator program
```

```
    This program simulates the water entrainment into the oil phase.  The calculator block has two input
    and two output streams.  The two input streams come from the liquid outputs of a three phase flash
    calculation.

    The first input is the oil stream that contains small amount of water dissolved in the oil phase.  The
    second input stream contains all the free water.  This program allows the user to specify the
    volumetric fraction of water to be carried into the oil phase and calculate the new compositions for
    the two output streams.

*/

CHEMCAD_Link CALC
{
    float factor;
    float t, p, vf, h_oil;
    float uspec[300];
    float oil_mol[100], wtr_mol[100];
    float entr_mol[100];
    float oilvol, entrain_wtrvol, h_wtr, wtrvol;

    int  oilstr;
    int  wtrstr;
    int  nc;
    int  i;

    /*  Get the equipment parameters of the calculator */
    CC_Get_Equip_Parameters(0,uspec);

    factor = uspec[2];
    oilstr = 1;
    wtrstr = 2;

    /*  Get oil stream */
    CC_Get_Input_Stream(oilstr, oil_mol, t, p, vf, h_oil);

    /*  Calc oil volume rate */
    oilvol = CC_Get_Stream_Property(oil_mol,t,p,0.,h_oil,47);
    printf("oilvol = %g\n",oilvol);

    /* Calc entrained water volume */
    entrain_wtrvol = oilvol * factor;
    printf("factor = %g\n",factor);
    printf("entrain_wtrvol = %g\n",entrain_wtrvol);

    /*  Get water stream */
    CC_Get_Input_Stream(wtrstr, wtr_mol, t, p, vf, h_wtr);

    /*  Calc water volume rate */
    wtrvol = CC_Get_Stream_Property(wtr_mol,t,p,0.,h_wtr,47);
    printf("wtrvol = %g\n",wtrvol);
    if( wtrvol <= 0. )
    {
        wtrvol = 0.00001;
    }

    /*  Calc total wtr moles */
    nc = CC_No_Of_Components();
    for( i = 0; i < nc; i++)
    {
        entr_mol[i] = wtr_mol[i] / wtrvol * entrain_wtrvol;

        /*  Entrained flow rate can not be larger than the water flow rate */
        if( entr_mol[i] > wtr_mol[i] ) entr_mol[i] = wtr_mol[i];
    }

    /*  New oil stream */
    for( i = 0; i < nc; i++)
    {
        oil_mol[i] = oil_mol[i] + entr_mol[i];
    }

    /*  New oil enthalpy */
    CC_enthalpy(oil_mol,t,p,0,h_oil);

    /*  New water stream */
    for( i = 0; i < nc; i++)
    {
        wtr_mol[i] = wtr_mol[i] - entr_mol[i];
    }

    /*  New water enthalpy */
    CC_enthalpy(wtr_mol,t,p,0,h_wtr);

    /*  Put oil back to process */
    CC_Put_Output_Stream(oilstr, oil_mol, t, p, 0., h_oil);

    /*  Put water back to process */
    CC_Put_Output_Stream(wtrstr, wtr_mol, t, p, 0., h_wtr);

}
```