USER ADDED MODULES GUIDE

WHAT IS A USER ADDED MODULE

This user's guide describes the procedures used to create User Added Modules in CHEMCAD version 5.4.5 and later. A user added module (UAM) is a user unit operation, K-value method, or enthalpy method created by the user or a third party. UAM's are programmed in Visual C++ and have access to a large number of internal CHEMCAD routines. For example, UAM's have been created to model membrane separators, fuel cells and other proprietary technology. UAM's also give users the ability to create new thermodynamic routines, or communicate data with other programs.

UAM's are functions compiled into the USRADD.DLL dynamic link library. A Microsoft Visual C++ 6.0 project for compiling this DLL is provided for you to customize with your own functions. We recommend using Microsoft Visual C++ version 6.0 with service pack 3 for coding your module. If you have user added unitops, a series of files defining the unitops dialog, array and reports must also be created.

`USRADD.DLL`-The dynamic link library, which contains your user added function

`ADD{n}.MY`-The dialog file for your user added unitop

`ADD{n}.MAP`-The definition file for the uspec (unitop) array

`ADD{n}.LAB`-The report definition file

In order to program UAM's you must first be proficient in C++ programming language. UAM's use many different features of C++ such as operator and function overloading. If you are not familiar with these concepts, I suggest you review C++.

C++ makes the UAM package very powerful, but somewhat challenging if you are new to programming. If you are new to programming, I suggest you do not use UAM to model your system, instead use our Excel unit operations. The Excel unit operations require only a basic knowledge of Visual Basic and fundamental concepts like arrays and functions.

This user's guide was written for the UAM developer with both CHEMCAD and Visual C++ experience. If you are a C++ developer new to CHEMCAD, I strongly suggest you review CHEMCAD until you become proficient in simulation before attempting to create a user added module.

This user's guide works together with the Screen Builder user's guide, which describes how to design and create dialog boxes for the user. Screen Builder documentation is required only for user-added unit operations, not K-value or enthalpy methods.

WHAT IS REQUIRED

Before we get started, let's talk about the UAM requirements. Programming a UAM requires:

A Pentium or faster PC with at least 32 megabytes of ram and at least 50 Megabytes of available hard drive space in addition to the following programs installed:

Microsoft Visual C++ version 6.0 service pack 3 or later

CHEMCAD version 5.4.5 or higher

CHEMCAD UAM version 5.4.5 or higher

Of course, a faster machine with more memory will yield faster, more reliable results. It is also important that you use the same version UAM as you do CHEMCAD itself. There is a different UAM package for each version of CHEMCAD.

INSTALLATION

To install the UAM development environment, select “Install accessories” from the CD menu, then select “User Added Modules” from the next menu. You may install the UAM anywhere, but I recommend using the default directory. If you use another directory, some project settings may not work properly. For information regarding how to install CHEMCAD or Microsoft Visual C++ version 5, please see their user’s guide.

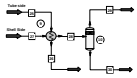
CHEMCAD SIMULATION STRUCTURE

Before you create a user-added module, you must first understand the structure of simulations in CHEMCAD. This section will familiarize you with how simulations are represented in CHEMCAD. This information is applicable to all simulations in CHEMCAD.

Simulation flowsheets are composed of two distinct objects, Streams and Unitops. Unitops are mathematical representations of process equipment. Streams connect Unitops to each other. Streams are flows of material at a given temperature and pressure. This network of Unitops connected by streams is called a flowsheet.

Unitops depend upon only two things regarding streams: what streams are connected and in what order. For example, a FLAS (Flash unitop) takes all the inlet streams, mixes them, brings them to equilibrium, then sends the vapor phase out the first outlet, and the liquid out the second.

Flowsheet may therefore be simplified into a topology. A topology is a listing of unitops, showing stream connections. The topology defines which streams connect which units. For example, this flowsheet:



May be described by the following topology:

Equipment	Stream Numbers
9 HTXR	27 25 -28 -26
10 FLAS	28 -29 –30

The feed and product arrows are merely placeholders for streams that don't originate or terminate with a unitop. The topology tells the simulator everything it needs to know: what streams are connected to which unitops and in what connection order.

UAM UNITOP ICONS

All unitops are represented on the flowsheet by icons. There is a stock set of icons for UAM's, but you are free to create your own user-added icons as well. Creation and editing of icons is done using the Symbol Builder program. Please refer to the CHEMCAD documentation for instructions on using the Symbol Builder program.

All unitops have some data associated with them, such as operating temperature, heat duty, number of stages, etc. This data is represented as a structured Array of no more than 250 elements (floating point or integer). If the data have engineering units (Degrees F, Degrees C, etc.) associated with them, the data is converted into the default internal unit set before being stored in the unit's array. Refer to the Dialog documentation for more details.

UAM LAYOUT

UAM's are simply functions in the `USRADD.DLL` dynamic link library. There are 12 different UAM's possible: 10 unitops, an ADDK function, and an ADDH function. Unitops are used within a flowsheet, usually to model a piece of equipment. ADDK () is used when you want to calculate K values via a new method. ADDH () is used when you wish to calculate stream enthalpy via a new method.

ADD1 ()-AD10 ()

All unitops (ADD1-AD10, as well as standard unitops) are accessed in the same general manner. CHEMCAD is a sequential simulator, it runs one unitop at a time, iterating around recycle loops for convergence. When the run sequence gets to a user unitop, such as ADD1, CHEMCAD runs the ADD1() function in USRADD.dll and then continues to the next unitop.

All unitops, including UAM's (e.g. ADD1 (), your function) must do the following:

- Read inlet stream data via SINP
- Read unitop parameters (if any). This data is passed into your function in the USPEC array
- Calculate the outlet streams. This is where your calculations come into effect.
- Put the data in the outlet streams via SOUT

User added unitops require extra files (.MY, .MAP files) to define the user dialog box.

An example unitop is given in the tutorial.

ADDK(FLOAT *Y, FLOAT *X, DOUBLE T, DOUBLE P, FLOAT *XKV)

If you wish to calculate your own K-values, you must define an ADDK function with the above format. Notice that x,y and xkv arrays are passed by reference. Your addk function must adjust them.

A simple example of this is the following function, which calculates K-values based upon ideal vapor pressure (vt(i,t) returns the vapor pressure of the ith component at a temperature of t(degrees Rankine)).

```
#include "stdafx.h"
#include "callccx.h"

EXTERN  short       nc,idcom[];
EXTERN  CP_RECORD   *cmp[];

void addk(float *y, float *x, double t, double p, float *xkv)
{
      // Sample ADDK using ideal vapor pressure for K values
      int         i;

      for( i = 0; i < nc; i++)
      {
            xkv[i] = vp(i,t) / p;
      }

    return;
}
```

ADDH(FLOAT *X, DOUBLE T, DOUBLE P, SHORT IPHASE, DOUBLE *HX)

If you wish to calculate your own enthalpy values, you must define an ADDH function with the above format. Notice hx is passed by reference. The example function below sets the stream enthalpy to 0.

```
      #include "stdafx.h"
#include "callccx.h"

EXTERN  short       nc,idcom[];
EXTERN  CP_RECORD   *cmp[];

void addh(float *x, double t, double p, short iphase, double *hx)
{
      *hx = 0.;
    return;
}
```

ADDPIPE1-10((DOUBLE WG, DOUBLE WL, DOUBLE RHOG, DOUBLE RHOL, DOUBLE VISG, DOUBLE VISL, DOUBLE SURF, DOUBLE D, DOUBLE RF, DOUBLE XL, DOUBLE Z,DOUBLE P, FLOAT *DP100F, FLOAT *DP100Z, FLOAT *DP100A, SHORT *IREGX))

The addpipe function allows you to enter a user added pipe pressure drop function into CHEMCAD's Pipe unitop. CHEMCAD will pass to your Addpipe1-10 function the following data in the specified units:

input:

wg	gas flow, lb.hr
wl	liquid flow , lb/hr
rhog	gas density, lb/ft3
rhol	liquid density, lb/ft3
visg	gas viscosity, cp
visl	liquid viscosity, cp
surf	liquid surface tension dyne/cm
d	pipe ID, inches
rf	pipe roughness, ft
xl	pipe length ft
z	elevation ft
p	inlet pressure pisa

The program expects your function to return to it the following data in the specified units (note data is passed by reference)

output:

dp100f	total pressure drop of two-phase friction psi/100ft
dp100z	total pressure drop of two-phase elevation psi/100ft
dp100a	total pressure drop of two-phase accelaration psi/100ft
iregx	flow region calculated by program

We give as an example the Beggs and Brill Calculation model in AddPIpe1, see next few pages for details.

```
        void  addpipe1(double wg, double wl, double rhog, double rhol, double visg,
                                                double visl, double surf, double d, double rf, double xl,
double z,
                                                double p,
                                                float *dp100f, float *dp100z, float *dp100a, short *iregx)
        /*
                This is an example which uses Beggs and Brill method for pipe pressure drop
calculation

                        input:
                        wg       gas flow, lb.hr
                        wl       liquid flow , lb/hr
                        rhog     gas density, lb/ft3
                        rhol     liquid density, lb/ft3
                        visg     gas viscosity, cp
                        visl     liquid viscosity, cp
                        surf     liquid surface tension dyne/cm
                        d        pipe ID, inches
                        rf       pipe roughness, ft
                        xl       pipe length ft
                        z        elevation ft
                        p        inlet pressure pisa

                    output:
                        dp100f    total pressure drop of two-phase friction psi/100ft
                        dp100z    total pressure drop of two-phase elevation psi/100ft
                        dp100a    total pressure drop of two-phase accelaration psi/100ft
                        iregx     flow region calculated by program

                        Reference: Two-Phase Flow in Pipes, Janmes P. Brill and H. Dale Beggs. Sixth
edition, Fourth Printing.  April, 1994

        */
        {
           double  A, dft, Vsg, Vsl, Vm, Nfr, lamda_L, Nlv;
           double  L1, L2, L3, L4;
           WORD    regime = 0;
           double  HL0, HL0_S, HL0_I, A_trans, B_trans, C, C_S, C_I, Rad;
           double  sin_fac, HL_S, HL_I, si, si_S, si_I, HL, rhos, dpdZ_el, rhon;
           double  visn, N_Ren, fn, y, lny, S, es, ftp, dpdZ_f;
           double  Ek, dpdZ, dpdZ_a;
```

```
dft = d / 12.;

//  Pipe area ft2
A = 3.14159 / 4. * dft * dft;

//  Superficial velocity  ft/s
Vsg = wg / 3600. / rhog / A;
Vsl = wl / 3600. / rhol / A;
Vm = Vsg + Vsl;

//  Nfr, pp 3-58
Nfr = Vm * Vm / (32.2 * dft);

// Nlv
Nlv = 1.938 * Vsl * pow((rhol/surf),0.25);

        // pp 3-58
lamda_L = Vsl / Vm;

L1 = 316. * pow(lamda_L,0.302);
L2 = 0.0009252 * pow(lamda_L, -2.4684);
L3 = 0.10 * pow(lamda_L,-1.4516);
L4 =0.5 * pow(lamda_L,-6.738);

//  Determine horizontal flow regime, pp3-58
if( (lamda_L < 0.01 && Nfr < L1 ) || ( lamda_L >= 0.01 && Nfr < L2 ))
{
   regime = SEGREGATED;
}
else if( lamda_L >= 0.01 && L2 <= Nfr && Nfr <= L3 )
{
   regime = TRANSITION;
}
else if( (0.01 <= lamda_L && lamda_L < 0.4 && L3 < Nfr && Nfr <= L1 ) ||
       (lamda_L >= 0.4 && L3 < Nfr && Nfr <= L4 ))
{
   regime = INTERMITTENT;
}
else if( (lamda_L < 0.4 && Nfr >= L1 ) || ( lamda_L >= 0.4 && Nfr > L4 ))
{
   regime = DISTRIBUTED;
}
```

```
else
{
   // Else, all other area is treated as distributed regime
   regime = DISTRIBUTED;
}

//  HL0 holdup which would exist at the same condition in a horizontal pipe, pp3-59
if( regime == SEGREGATED )
{
   HL0 = 0.98 * pow(lamda_L,0.4846) / pow(Nfr,0.0868);
}
else if( regime == INTERMITTENT )
{
   HL0 = 0.845 * pow(lamda_L,0.5351) / pow(Nfr,0.0173);
}
else if( regime == DISTRIBUTED )
{
   HL0 = 1.065 * pow(lamda_L,0.5824) / pow(Nfr,0.0609);
}
else
{
   //  Transition
   HL0_S = 0.98 * pow(lamda_L,0.4846) / pow(Nfr,0.0868);
   HL0_I = 0.845 * pow(lamda_L,0.5351) / pow(Nfr,0.0173);

   // dspmsg("HL0_S = %g HL0_I = %g\n",HL0_S, HL0_I);

   A_trans = (L3 - Nfr) / (L3 - L2);
   B_trans = 1. - A_trans;

   HL0 = A_trans * HL0_S + B_trans * HL0_I;
}


//  Constraint is HL0 > lamda_L
if( HL0 < lamda_L ) HL0 = lamda_L;


// Calc C, pp 3-60
if( z < 0. )
{
   //  Downhill
   C = (1. - lamda_L ) * log( 4.70 * pow(lamda_L,-0.3692)
```

```
                        * pow(Nlv,0.1244)
                        * pow(Nfr,-0.5056));
   C_S = C;
   C_I = C;
}
else
{
   // Uphill
   if( regime == SEGREGATED )
   {
      C = (1. - lamda_L ) * log( 0.011 * pow(lamda_L,-3.768)
                              * pow(Nlv,3.539)
                              * pow(Nfr,-1.614));
   }
   else if( regime == INTERMITTENT )
   {
      C = (1. - lamda_L ) * log( 2.96  * pow(lamda_L,0.305)
                              * pow(Nlv,-0.4473)
                              * pow(Nfr,0.0978));

   }
   else if( regime == DISTRIBUTED )
   {
      C = 0.;
   }
   else
   {
      //  Transition
      C_S = (1. - lamda_L ) * log( 0.011 * pow(lamda_L,-3.768)
                              * pow(Nlv,3.539)
                              * pow(Nfr,-1.614));
      C_I = (1. - lamda_L ) * log( 2.96  * pow(lamda_L,0.305)
                              * pow(Nlv,-0.4473)
                              * pow(Nfr,0.0978));
   }
}

//  Calc flow angle in radians
if ( z > xl ) z = xl;
Rad = asin(z/xl);

        // eqn 3.82
sin_fac = sin(1.8 * Rad) - 0.333 * pow(sin(1.8 * Rad),3.);
```

```
si = 1. + C * sin_fac;

if( regime == TRANSITION )
{
   si_S = 1. + C_S * sin_fac;
   HL_S = HL0_S * si_S;
   si_I = 1. + C_I * sin_fac;
   HL_I = HL0_I * si_I;

   HL = A_trans * HL_S + B_trans * HL_I;

}
else
{
   HL = HL0 * si;
}

if( HL > 1. ) HL = 1.;
if( HL < 0. ) HL = 0.;


// Calc rhos
rhos = rhol * HL + rhog * ( 1.- HL );


//  dpdZ_el in psf/ft
dpdZ_el = rhos;

// dp100z in psi/100 ft
*dp100z = dpdZ_el / 144. * 100.;

rhon = rhol * lamda_L + rhog * ( 1. - lamda_L);

visn = visl * lamda_L + visg * ( 1. - lamda_L);   // in cp

N_Ren = 1488. * rhon * Vm * dft / visn;


//  Calc non-slip friction factor, determined from the smooth pipe
//  curve on a Moody diagram or from eqn
        double Xre = N_Ren;
        if( Xre < 7. ) Xre = 7.;   // 7-30-02
```

```
    fn = 1. / pow( 2. * log10( Xre / (4.5223 * log10( Xre) - 3.8215)),2.);

    y = lamda_L / pow(HL,2.);

    if( 1 < y && y < 1.2 )
    {
       S = log(2.2 * y - 1.2);
    }
    else
    {
       lny = log(y);
       S = lny /
        ( -0.0523 + 3.182 * lny - 0.8725 * pow(lny,2.) + 0.01853 * pow(lny,4.));
    }
    es = exp(S);

    ftp = fn * es;

    //  dpdZ_f in psf/ft
    dpdZ_f = ftp * rhon * Vm * Vm / (2. * 32.2 * dft);

    // eqn 3.93 p * 144 to lbf/ft2
    Ek = rhos * Vm * Vsg / (32.2 * p * 144.);

    if( Ek > 0.8 )
    {
       Ek = 0.8;
    }

    dpdZ = (dpdZ_el * sin(Rad) + dpdZ_f) / (1. - Ek);

    dpdZ_a = dpdZ * Ek;

    *dp100f = dpdZ_f / 144. * 100.;
    *dp100a = dpdZ_a / 144. * 100.;

    *iregx = regime;

    return;
}
```

OVERVIEW OF PROJECT

The project to compile the USRADD.DLL file is included in the CC5UAM package. This project is a Microsoft Visual C++ version 5 project. It contains all the settings, header files, libraries and debug versions of our DLLs needed for you to compile your USRADD.DLL and test it in debug.

File listing of CC5UAM project:

CC5D (folder)-Debug versions of EXE and dlls available for troubleshooting your module in DEBUG

Debug (folder)-Used for Debug info on your dll

Release (folder)-Where the release version of your dll resides

Res-Resources folder

ADD1-AD10.CPP-Templates for your code

ADDH, ADDK.CPP-User added K values templates

ADDPIPE.CPP- User-added Pipe model template (Beggs and Brill)

CALLCCX.H-Header file describing various CHEMCAD functions

CC5.H-Header file describing various CHEMCAD functions

Microsoft generated files:

STDAFX.H, CPP

Resource.h

USRADD.*

User code goes in the provided templates for ADD1.cpp-AD10.cpp, ADDK.cpp, ADDH.cpp, and ADDPIPE.cpp. These files are yours to modify, but I recommend making copies of these files for your reference before you do so. We have structured this project so that code for each UAM function goes into a separate source file.

After completing your code, you will have to compile your USRADD.dll and move the compiled dll into your CC5 program directory, along with any additional files such as .MY, .MAP, or .LAB files for your unitops. The screen builder documentation describes these files and how to create them.

UNIT OPERATIONS: GETTING/SETTING STREAM AND UNIT DATA

All unitops, including UAM's (e.g. ADD1 (), your function) must do the following:

- Read inlet stream data via SINP
- Read unitop parameters (if any). This data is passed into your function in the USPEC array
- Calculate the outlet streams. This is where your calculations come into effect.

- Put the data in the outlet streams via SOUT

The uspec array is automatically passed to your function. In order to get stream data, you must use the SINP-> interface. For example, the following returns the flowrate of the jth component in the ith inlet stream in lbmol/hr:

Flowrate[i][j]=sinp->fc[i][j];

This next line sets a variable pout to be equal to the pressure of the first inlet stream of the unitop:

pout = sinp->fp[0];

Data is sent to outlet streams in a similar fashion. The next line sets the temperature of the second outlet equal to tout:

sout->ot[1] = tout;

It is the order of inlets and outlets (as connected in the topology) which determines how streams are accessed in your code. It is also important to note that the first inlet or outlet is listed as 0, the second as 1, the third as 2, etc.

GETTING STREAM PROPERTIES

Stream properties are defined here as properties dependent upon the composition, pressure and temperature of a stream. You cannot directly set these properties; T, P, composition, and the models being used determine them. You can however, read these properties by using the various Library functions listed in the appendix.

Flashing streams

Flashing a stream means to bring it to vapor-liquid equilibrium using the selected K and H model. It is important to flash your outlet streams to bring them to equilibrium before passing them through sout. There are several functions, depending on your needs:

- HSPFLASH: Given enthalpy and pressure, calculate temperature and vapor fraction
- TPFLASH: Given temperature and pressure calculate enthalpy and vapor fraction
- VPFLASH: Given vapor fraction and pressure, calculate temperature and enthalpy
- VTFLASH: Given vapor fraction and temperature, calculate pressure and enthalpy

All available functions are described in detail in the Function Library Appendix.

Examples of UAM's

Three UAM examples are included in the CC5UAM project. ADD1 (), ADDK (), and ADDH ().

DEBUGGING TIPS AND TECHNIQUES

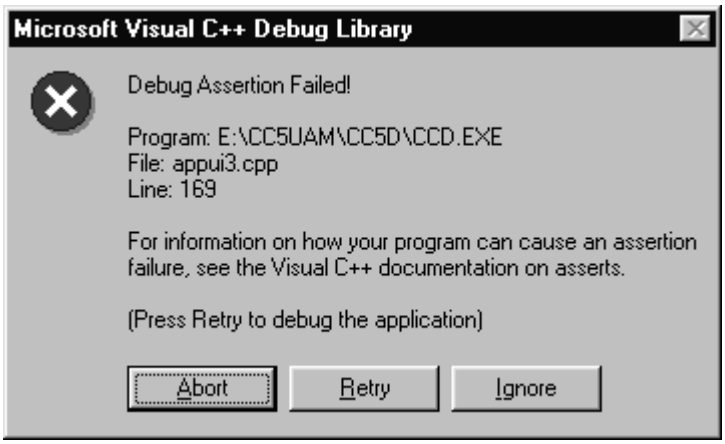
Debug mode in Visual Studio allows you to step through your code line by line examining variables and logic within your functions. It is an extremely useful tool to find hidden bugs and errors. The user should be familiar with the debugging tools of Visual C++ version 5.0. Microsoft has several excellent manuals on this subject, we will not repeat that information here except where necessary to explain a point unique to CHEMCAD UAM programming.

In order to use debug mode, special “debug” versions of your program must be created. Setting the active configuration, in the “build” menu to “USRADD Win32-debug” does this. Your debug version of the dll should always be used with the ccd.exe and debug versions of our dlls provided in the cc5d directory. The project is already set for this. While we provide debug versions of dlls, we do not provide enough information for you to debug into our program, so all code outside of that in usradd.dll will be inaccessible. Do not try to use a debug version usradd.dll with a release version of cc5.exe or vice versa. Trying to do so may result in an error.

The simplest way to use a debug session is to place a breakpoint in your user added function (such as ADD2 ()), run the debug mode, and create a simulation which uses that function. When CHEMCAD comes to your user added function, for example when you run your unit operation (make sure the inlet streams have some flow) Visual studio will detect your breakpoint and stop execution at that point. You can then step through your code, line by line, adding watches on important variables and using the mouse to check current values of variables.

When starting a debug session, you will get a message warning you “…CCd.exe does not contain any debugging information…” This is normal; we don’t give enough information to debug the executable, only your dll. Press **[OK]** to continue. If you wish, you may disable this notification by checking the checkbox on the dialog with the warning.

On certain systems (mostly Windows 9x operating systems) you will get a series of 4 debug assertion failures as CHEMCAD loads. These assertion failures happen in a Microsoft source file `appui3.cpp`. These are normal for this version and will cause no noticeable problems in the release version. Simply press the “Ignore” button to get past each message.



The first debug assertion failure

After these messages, CHEMCAD will start and appear just like the release version, albeit slightly slower due to the debug overhead. If you get a debug assertion failure in your code (i.e. other than the first four which normally show) you may use the “Retry” button to debug it.

APPENDIX: CHEMCAD LIBRARY OF FUNCTIONS

In order for your UAM to be successful, you will need access to physical properties and DATA from within CHEMCAD. This access is granted through a library of functions. This library consists of several parts:

- General Functions-Access to streams and equipment, printing functions
- Engineering Units-Unit conversion, units reporting information
- Physical Property Calculation-Calculation of stream properties
- Thermodynamic Calculations-Calculation of K-values, Enthalpy, and Entropy of streams

GENERAL FUNCTIONS

Message display

The following functions use the flowsheet window as a scrollable text area. The first call to Put_Scr_Line automatically opens this area. If you wish, you can clear the scrollable text area between calls to Put_Scr_Line with a call to Clear_Scr_Wndo. It is your responsibility to call Close_Scr_Wndo when you are finished using the scrollable text area. These functions are useful for displaying messages and calculations while your user-added functions are executing. They are used by several of the standard unit operation models. They are also useful for debugging purposes.

Prototypes:

```
void    Put_Scr_Line(char *scr_line);
void    Clear_Scr_Wndo(void);
void    Close_Scr_Wndo(void);
```

Reporting Functions

The following functions can be used to generate reports from your user-added modules. Typically, this is not done unless you are in the process of debugging your code. If you use these functions, the output is sent to the current report device, as specified by the Control/Reports menu option. This is the screen, a file, a printer, or a "NUL" device (meaning that no output is physically produced). It is your responsibility to call the Close_Report function when you are finished producing your report.

Starting a Report

You specify the name of a function, which is to be used to print page headings at the start of each page, and, if necessary, the address of a parameter block of values required by this page headings function.

```
typedef void    (*HEADER_FN)(void *parm_block);

void    Open_Report(HEADER_FN header_fn, void *hdr_parm_block);
```

The following code illustrates how the Open_Report function and a page headings function are specified.

```
/* this is the parameter block used to pass parameters to the */
/* function which prints the header lines at the top of each  */
/* page of the report */
typedef struct
{
```

```
  char    *title;
  char    *column_headings;
} HDR_PARMS;
/* function prototype for printing top-of-page headings */
static    void      Report_Header(void *arg1void);
/* declare our top-of-page function parameters area */
HDR_PARMS     header_parms;
/* initialize all values in the header parameter block */
header_parms.title = "Sample Report Headings";
header_parms.column_titles = "Stream ID    Flowrate   etc.";
/* open the report */
Open_Report(Report_Header, &header_parms);
/* if you don't want any headings at the top of each page, */
/* specify a NULL headings function and parameter block */
Open_Report(NULL, NULL);
/* this function prints the headers at the top of each page */
static    void      Report_Header(void *arg1void)
{
  HDR_PARMS   *hdr_parms = (HDR_PARMS *) arg1void;
  /* dump the page number */
  Put_Rpt_PageNo();
  /* dump the report heading */
  Put_Rpt_Line(hdr_parms->title);
  /* skip an intermediate line */
  Put_Rpt_CRLF();
  /* dump the column titles */
  Put_Rpt_Line(hdr->parms->column_titles);
  /* skip an intermediate line */
  Put_Rpt_CRLF();
  return;
}
```

Closing a Report

```
Void      Close_Report(void);
Close_Report();
```

It is your responsibility to call Close_Report when you are finished producing your report. When you call Close_Report, your output is sent to the current report output device, as specified by the Control/Reports menu option.

Sending Output to a Report

```
void      Put_Rpt_CRLF(void);
Put_Rpt_CRLF();
```

This function dumps a carriage-return line-feed to the report.

```
void      Put_Rpt_FormFeed(void);
Put_Rpt_FormFeed();
```

This function dumps a form-feed to the report.

```
void       Put_Rpt_Line(char *rpt_line);
Put_Rpt_Line("Example of a report line");
```

This function dumps a line to the report and appends this line with a carriage-return / line-feed.

```
void       Put_Rpt_String(char *rpt_line);
Put_Rpt_String("Waiting for the rest of the string ....");
```

This function dumps a string to the report but does NOT append a carriage-return / line-feed to the end of the line.

```
void       Put_Rpt_PageNo(void);
Put_Rpt_PageNo();
```

This function right-justifies the current page number on the page.

Determining the Current Report Status

```
int        Get_Rpt_Dest(void);
switch (Get_Rpt_Dest())
{
  case 'S':
  /* report is going to the screen */
  break;
  case 'P':
  /* report is going to the printer */
  break;
  case 'F':
  /* report is going to a file */
  break;
  case 'L':
  /* report is going to a Lotus file */
  break;
  case 'N':
  /* report is going nowhere (NUL output device) */
  break;
}
```

This function returns the current report destination, as specified in the Control/Reports menu option.

```
void       Set_Rpt_Dest(int dest);
Set_Rpt_Dest('S');     // send report output to the screen
Set_Rpt_Dest('P');     // send report output to the printer
Set_Rpt_Dest('F');     // send report output to a file
Set_Rpt_Dest('N');     // discard report output
```

This function sets the report destination, temporarily overriding the destination specified in the Control/Reports menu option.

```
int        Get_Rpt_Handle(void);
if (Get_Rpt_Handle()) // do some output
```

This function indicates whether or not the report output device is ready for output, i.e. whether or not you have already called Open_Report and have not yet called Close_Report.

int Get_Top_Margin(void);
top_rows = Get_Top_Margin();

This function returns the top margin as a number of rows.

int Get_Bottom_Margin(void);
bottom_rows = Get_Bottom_Margin();

This function returns the bottom margin as a number of rows.

int Get_Rpt_Rows(void);
page_rows = Get_Rpt_Rows();

This function returns the number of rows per page. This value includes the top margin and bottom margin. The number of useable rows per page can be calculated as follows:

use_rows = Get_Rpt_Rows() - Get_Top_Margin() - Get_Bottom_Margin();

int Get_Rpt_Cols(void);
char_cols = Get_Rpt_Cols();

This function returns the page width (in characters).

int Get_Rpt_LineNo(void);
cur_line = Get_Rpt_LineNo();

This function returns the current line number (on the current page).

int Get_Rpt_PageNo(void);
cur_page = Get_Rpt_PageNo();

This function returns the current report page number.

Function: Browse_File

Browse an ASCII file in read-only mode

Prototypes:

CALL Browse_File('TEST.CL'//nullchar, ' COMPONENT LIST '//nullchar)
int Browse_File(char *file_path, char *window_title);

Parameters:
Input: DOS Filespec, Browse Window Title (or NULL)
Return Value: 0 = file browsed OK, non-zero = file not found
Description:
This function allows you to browse any file from within the CHEMCAD graphics interface.

Function: Check_Stream_Rec
Check the validity of a stream (.STR) file record
Prototypes:

ierr = Check_Stream_Rec(ID)

int Check_Stream_Rec(STREAM_REC *stream_rec);

Return Value: 0 = record is good, non-zero otherwise

Description:
This function can be used subsequently to GETSTR or Get_Stream_Rec to check the validity of a record retrieved from the stream (.STR) file.

Function: Check_Unit_Rec
Check the validity of an equipment (.EQS) file record
Prototypes:

ierr = Check_Unit_Rec(ID)
int Check_Unit_Rec(UNIT_REC *unit_rec);

Return Value: 0 = record is good, non-zero otherwise

Description:
This function can be used subsequently to GETEQS or Get_Unit_Rec to check the validity of a record retrieved from the equipment (.EQS) file.

The following functions initialize a character pointer to a NULL-terminated string.

char *Get_Cur_Date_Str(void);
char *Get_Cur_Time_Str(void);

Function: Get_EQS_Count
Returns the number of valid equipment (.EQS) file records
Prototypes:

ieqs = Get_EQS_Count
int Get_EQS_Count(void);

Return Value: count of valid .EQS records

Description:
Use this function as a preliminary to Get_EQS_Ids
Function: Get_EQS_Ids
Returns a WORD array of valid equipment ID's in the .EQS file
Prototypes:

CALL Get_EQS_Ids(ieqsids)
void Get_EQS_Ids(WORD *id_array);

Parameters:
Output: array of valid .EQS ID's

Description:
This function can be used to determine the array of valid equipment ID's in the equipment (.EQS) file. The array size is given by
Get_EQS_Count.

Function: Get_STR_Count
Returns the number of valid equipment records in the stream (.STR) file

Prototypes:

istr = Get_STR_Count
int Get_STR_Count(void);

Return Value: count of valid .STR records

Description:
Use this function as a preliminary to Get_STR_Ids

Function: Get_STR_Ids
Returns a WORD array of valid stream ID's
Prototypes:

CALL Get_STR_Ids(istrids)
void Get_STR_Ids(WORD *id_array);

Parameters:
Input: None
Output: array of valid .STR ID's
Return Value: none

Description:
This function can be used to determine the array of valid stream ID's in the stream (.STR) file. The array size is given by
Get_STR_Count.

Function: Get_Stream_Rec
Retrieve a stream record from the stream (.STR) file.
Prototypes:

int Get_Stream_Rec(STREAM_REC *strm_buff, int strm_id);

Parameters:
Input: strm_id stream ID to be retrieved
Output: strm_buff strm_buff is initialized with all values retrieved from the stream file
Description:
Get_Stream_Rec is called by C functions to retrieve a record from the stream (.STR) file. The STREAM_REC structure is included in CC5.H. Use Check_Stream_Rec to test the validity of a stream record.

Function: Get_Unit_Rec
Retrieve an equipment record from the equipment (.EQS) file.
Prototypes:

Use GETEQS() if calling from FORTRAN
int Get_Unit_Rec(UNIT_REC *unit_buff, int unit_id);

Parameters:
Input: unit_id unit ID to be retrieved
Output: unit_buff unit_buff is initialized with all values retrieved from the equipment file

Description:
Get_Unit_Rec is called by C functions to retrieve a record from the equipment (.EQS) file. The UNIT_REC structure is included in CC5.H. Use Check_Unit_Rec to check the validity of an equipment record.

Function: Gprintf
Format and display program values on the bottom-of-screen message line and, if required, wait for a key to be pressed
Prototypes:

CALL Gprintf(0, 'Values are %d, %g'//nullchar, ival, fval)
void Gprintf(int message_type, char *format_string, ...);

example: Gprintf("Values are %d %g", ival, fval);

Parameters:
Input: message_type 0 = beep and wait for a key to be pressed
1 = don't beep and don't wait for a key
2 = beep but don't wait for a key
Output: string is displayed on the bottom of the screen
Return Value: none
Description:
Gprintf can be used as a substitute for printf to display program values. The bottom-of screen message area is used to display all output, one line at a time. Use message_type to indicate whether or not some action is required on the part of the user after a message is displayed.

This function can be used to display error messages on the bottom line of the screen. If a message_type of 0 is specified, the message will freeze the system until you press a key or a mouse button. Other values of message_type do not wait for a key to be pressed. This function is useful for debugging your subroutines.

Function: STRUNP
"Unpack" a string into a NULL-terminated string
Prototypes:

void strunp(char *target, char *source, unsigned int length);

Parameters:
Input: source address of non NULL-terminated string
length length of string to be unpacked
Output: target address to be used for saving the NULL-terminated string
Return Value: none

Description:
This function is useful for "unpacking" or "unloading" strings, which are not terminated with a NULL byte into a NULL-terminated string area. Examples of non NULL-terminated strings are unit labels in equipment records (UNIT_REC), stream labels in stream records (STREAM_REC), and component names in physical properties data records (CP_RECORD).

ENGINEERING UNITS FUNCTIONS

These functions display the current units set and allow for conversion to other sets. Keep in mind that all stream and unit data coming from unitop and stream records will always be in internal units.

Each of the following functions initializes a character pointer to a NULL-terminated character string representing the currently selected user units.

```
char    *Get_Act_Liq_Vol_Rate_Str(void);
char    *Get_Act_Vap_Vol_Rate_Str(void);
char    *Get_Area_Str(void);
char    *Get_Cake_R_Str(void);
char    *Get_Crude_Rate_Str(void);
char    *Get_Delta_P_Str(void);
char    *Get_Diameter_Str(void);
char    *Get-DM_Str(void);
char    *Get_Flow_Rate_Str(void);
char    *Get_Flow_Str_(void);
char    *Get_Heat_Rate_Str(void;
char    *Get_Heat_Str(void);
char    *Get_HTC_Str(void);
char    *Get_Inverse_Length_Str(void);
char    *Get_Length_Str(void);
char    *Get_Liq_Den_Str(void);
char    *Get_Liq_Vol_Str(void);
char    *Get_Mass_Cp_Str(void);
char    *Get_Mass_Heat_Str(void);
char    *Get_Mass_Rate_Str(void);
char    *Get_Mass_Str(void);
char    *Get_Mole_Heat_Str(void);
char    *Get_Mole_Rate_Str(void;
char    *Get_Mole_Str(void);
char    *Get_Packing_DP_Str(void);
char    *Get_Power_Str(void);
char    *Get_Pressure_Str(void);
char    *Get_Solub_Str(void);
char    *Get_Spec_Vol_Str(void);
char    *Get_Std_Vap_Vol_Rate_Str(void);
char    *Get_Std_Liq_Vol_Rate_Str(void);
char    *Get_SurfTen_Str(void);
char    *Get_TC_Str(void);
char    *Get_Temperature_Str(void);
char    *Get_Thickness_Str(void);
char    *Get_Time_Str(void);
char    *Get_Vap_Den_Str(void);
char    *Get_Vap_Vol_Str(void)
char    *Get_Velocity_Str(void);
char    *Get_Visc_Str(void);
char    *Get_Weight_Cp_Str(void)
```

```
char    *Get_Weight_Heat_Str(void);
char    *Get_Work_Str(void);
```

The following functions can be used to translate a value in internal units into one that reflects the currently selected user units. The appropriate function should be called prior to displaying a value or saving it in a report file.

```
REAL    Get_Area(double area);
REAL    Get_Cake_R(double cake_resistance);
REAL    Get_Crude_Rate(double crude_rate);
REAL    Get_Delta_P(double delta_p);
REAL    Get_Delta_T(double delta_t);
REAL    Get_Dia(double diameter);
REAL    Get_DM(double dipole_moment);
REAL    Get_Flow(double flow, double mw);
REAL    Get_Flow_Rate(double flow_rate, double mw);
REAL    Get_Heat(double heat);
REAL    Get_Heat_Rate(double heat_rate);
REAL    Get_HTC(double heat_tansfer_coeff);
REAL    Get_Length(double length);
REAL    Get_Liq_Den(double liq_den);
REAL    Get_Liq_Vol(double liq_vol);
REAL    Get_Liq_Vol_Rate(double liq_vol_rate);
REAL    Get_Mass(double mass);
REAL    Get_Mass_Cp(double mass_cp, double mw);
REAL    Get_Mass_Heat(double mass_heat, double mw);
REAL    Get_Mass_Rate(double mass_rate);
REAL    Get_Mole_Rate(double mole_rate);
REAL    Get_Packing_DP(double packing_dp);
REAL    Get_Power(double power);
REAL    Get_Pressure(double pressure);
REAL    Get_Solub(double solubility);
REAL    Get_SurfTen(double surften);
REAL    Get_TC(double tc);
REAL    Get_Temperature(double temperature);
REAL    Get_Thickness(double thickness);
REAL    Get_Vap_Den(double vap_den);
REAL    Get_Vap_Vol(double vap_vol);
REAL    Get_Vap_Vol_Rate(double vap_vol_rate);
REAL    Get_Velocity(double velocity);
REAL    Get_Visc(double viscosity);
REAL    Get_Work(double work);
```

PHYSICAL PROPERTY CALCULATIONS

Function: CP
Function to calculate the heat capacity of a mixture
Prototypes:

heatcap = CP(xmol, t, p, iphase)
REAL cp(REAL *xmol, double t, double p, WORD iphase);
Parameters:

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

Return Value:
REAL - cp in Btu/R/lbmol
Description:
Given composition, temperature and pressure, this function calculates the heat capacity of the stream.

Function: CV
Function to calculate the cv of a mixture
Prototypes:

cvret = CV(xmol, t, p, iphase)
REAL cv(REAL *xmol, double t, double p, WORD iphase);

Parameters:

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

Return Value:
REAL - cv in Btu/R/lbmol
Description:
Given composition, temperature and pressure, this function calculates the cv of the stream

Function: GVISCO
Function to calculate the vapor viscosity of a mixture
Prototypes:

CALL GVISCO(xmol, t, p, vis)
REAL gvisco(REAL *xmol, double t, double p);

Parameters:

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

Return Value (for C):
REAL - viscosity in CP
Description:

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

Function: LDENSE
Liquid density routine.
Prototypes:

CALL LDENSE(xliq, t, p, dens)
void ldense(REAL *xliq, double t, double p, REAL *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: none
Description:
For given component flow rate, temperature and pressure, this routine calculates the liquid density of the mixture.

Function: LTHC
Function to calculate the liquid thermal conductivity of a mixture
Prototypes:

CALL LTHC(xmol, t, p, thc)
REAL lthc(REAL *xmol, double t, double p);
Parameters:
Input: xmol[] mole flow rate in lbmol/hr
t temperature in R
p pressure in psia
Return Value (for C):
REAL - liquid thermal conductivity in Btu/hr-ft-F
Description:
Given composition, temperature and pressure, this function calculates the thermal conductivity of the liquid mixture.

Function: LVISCO
Function to calculate the liquid viscosity of a mixture
Prototypes:

CALL LVISCO(xmol, t, p, visc)
REAL lvisco(REAL *xmol, double t, double p);

Parameters:
Input: xmol[] mole flow rate in lbmol/hr
t temperature in R
p pressure in psia
Return Value (for C):
REAL - viscosity in CP
Description:
Given composition, temperature and pressure, this function calculates the viscosity of the liquid mixture.

Function: SURFTEN
Function to calculate the liquid surface tension of a mixture
Prototypes:

```
CALL SURFTEN(xmol, t, p, sft)
REAL surften(REAL *xmol, double t, double p);
```

Parameters:

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

Return Value (for C):
REAL - liquid surface tension in dyne/cm
Description:
Given composition, temperature and pressure, this function calculates the surface tension of the liquid mixture.

Function: VDENSE
Vapor density routine.
Prototypes:

```
CALL VDENSE(xvap, t, p, dens)
void vdense(REAL *xvap, double t, double p, REAL *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: none
Description:
For given component flow rate, temperature and pressure, this routine calculates the vapor density of the mixture.

Function: VP
Function to calculate the pure component vapor pressure
Prototypes:

```
vpres = VP(i, t)
REAL    vp(WORD i, double t);
```

Parameters:

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

Return Value:
REAL - vapor pressure in psia
Description:
Given component i and temperature, this routine calculates the vapor pressure.

Function: VTHC
Function to calculate the vapor thermal conductivity of a mixture
Prototypes:

CALL VTHC(xmol, t, p, thc)
REAL vthc(REAL *xmol, double t, double p);

Parameters:

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

Return Value (for C):
REAL - vapor thermal conductivity in Btu/hr-ft-F
Description:
Given composition, temperature and pressure, this function calculates the thermal conductivity of the vapor mixture.

Function: ZFACTOR
Z factor calculation routine.
Prototypes:
CALL ZFACTOR(xmol, t, p, iphase, z, ierr)
void zfactor(REAL *xmol, double t, double p, WORD iphase, REAL *z, WORD *ierr);

Parameters:

Input:	xmol[]	component mole flow rate(lbmol/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: none
Description:
For given component flow rate, temperature and pressure, this routine calculates Z factor of the stream.

THERMODYNAMIC CALCULATIONS

Function: ENTHALPY
Enthalpy calculation routine.
Prototypes:

CALL Enthalpy(xmol, t, p, iphase, hx)
void enthalpy(REAL *xmol, double t, double p, WORD iphase,REAL *hx);

Parameters:

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

Return Value: none
Description:
Note: Do not call this function from your ADDH function

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: Entropy
Entropy calculation routine.
Prototypes:

CALL Entropy(XMOL, T, P, IPHASE, SX)
void Entropy(REAL *xmol, double t, double p, WORD iphase, REAL *sx);

Parameters:

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

Return Value: none
Description:
For given component flow rate, temperature and pressure, this routine calculates the entropy of the stream.

Function: Hxstream
Enthalpy calculation routine called from ADDH.
Prototypes:

CALL Hxstream(xmol, t, p, iphase, hx)
void Hxstream(REAL *xmol, double t, double p, WORD iphase, REAL *hx);

Parameters:

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

Return Value: none
Description:
This enthalpy routine can be called from routine ADDH, where enthalpy models such as SRK can be used as the default 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	modeh	H model
1	Polynomial H	7	Latent Heat

2	Redlich-Kwong	8	Amine
3	SRK	9	No Enthalpy
4	API SRK	10	Enthalpy Table
5	Peng-Robinson	11	ADDH
6	Lee-Kesler	12	Mixed Model

Function: Hspflash
Adiabatic/Isentropic flash calculation at specified pressure
Prototypes:

```
CALL Hspflash(feed, p, hsin, mode, testi, vapor, xliq, totalv,
    &         totall, tout, hvap, hliq, htotal, xk, v);

WORD Hspflash(REAL *feed, double p, double hsin, WORD mode,
              double testi, REAL *vapor, REAL *xliq, REAL *totalv,
              REAL *totall, REAL *tout, REAL *hvap, REAL *hliq,
              REAL *htotal, REAL *xk, REAL *v, REAL *f_ions);
```

Parameters:

Input:	feed[]	lbmoles/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 (for C):

WORD	ret_val	= 0	Normal return
		= 1	Diverge

Description:
For adiabatic flash (constant enthalpy) 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: Keq
K value calculation routine
Prototypes:

```
CALL Keq(yv, xl, t, p, xkv)
void Keq(REAL *yv, REAL *xl, double t, double p, REAL *xkv,
         REAL *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: none
Description:
Note: Do not call this function from your ADDK function

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: Tpflash
Isothermal flash calculation at specified temperature and pressure
Prototypes:

```
CALL Tpflash (feed, hin, tr, psia, vapor, xliq, totalv, totall,
    &         hvap, hliq, htotal, vout, xk, delq)
WORD Tpflash (REAL *feed, double hin, double tr, double psia,
              REAL *vapor, REAL *xliq, REAL *totalv,
              REAL *totall, REAL *hvap, REAL *hliq, REAL *htotal,
              REAL *vout, REAL *xk, REAL *delq, REAL *f_ions);
```

Parameters:

Input:	feed[]	lbmol/hr
	hin	btu/hr
	tr	flash T in degree R
	psia	flash P in psia
Output:	vapor[]	vapor out lbmol/hr
	xliq[]	liquid out lbmol/hr
	totalv	total vapor flow lbmol/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 (for C):

WORD	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: Vpflash
Flash calculation at specified mole vapor fraction and pressure
Prototypes:

```
CALL Vpflash(feed, v, p, hin, testi, irelod, vapor, xliq, totalv,
    &       totall, tout, hvap, hliq, htotal, xk, delq)
WORD Vpflash(REAL *feed, double v, double p, double hin, double testi,
             WORD irelod, REAL *vapor, REAL *xliq, REAL *totalv,
             REAL *totall, REAL *tout, REAL *hvap, REAL *hliq,
             REAL *htotal, REAL *xk, REAL *delq, REAL *f_ions);
```

Parameters:

Input:	feed[]	lbmol/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 lbmol/hr
	xliq[]	liquid product lbmol/hr
	totalv	total vapor rate lbmol/hr
	totall	total liquid rate lbmol/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:

WORD	ret_val = 0	Normal return
	= 1	Diverge

Description:
This function 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: Vtflash
Flash calculation at specified mole vapor fraction and temperature
Prototypes:

```
CALL Vtflash(feed, vfrac, t, hin, pesti, irelod, vapor, xliq,
    &        totalv, totall, pout, hvap, hliq, htotal, xk, delq)
WORD Vtflash(REAL *feed, double vfrac, double t, double hin,
             double pesti, WORD irelod, REAL *vapor, REAL *xliq,
             REAL *totalv, REAL *totall, REAL *pout, REAL *hvap,
             REAL *hliq, REAL *htotal, REAL *xk, REAL *delq,
             REAL *f_ions);
```

Parameters:

Input:	feed[]	lbmol/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 lbmol/hr
	xliq[]	liquid product lbmol/hr
	totalv	total vapor rate lbmol/hr
	totall	total liquid rate lbmol/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 (for C):

WORD	ret_val = 0	Normal return
	= 1	Diverge

Description:

This function 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: Kxeq

K value calculation routine called from user added K value routine (ADDK)

Prototypes:

```
CALL Kxeq(yv, xl, t, p, xkv)
void Kxeq(REAL *yv, REAL *xl, double t, double p, REAL *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: none
Description:
This K value routine can be called from routine ADDK, where K models such as SRK can be used as the default model. 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	modek	K model
1	Polynomial K	17	Van Laar
2	Grayson Streed	18	ADDK
3	SRK	19	Henry's law
4	API SRK	20	Flory Huggins
5	UNIFAC	21	UNIFAC Polymers
6	K table	22	MSRK
7	Wilson	23	PPAQ
8	Ideal Vapor Pressure	24	TSRK
9	Peng-Robinson	25	TEG Dehydration
10	NRTL	26	ACTX
11	ESSO	27	T-K Wilson
12	Amine	28	HRNM Modified Wilson
13	Sour Water	29	PSRK
14	UNIQUAC	30	GMAC (Chien-Null)
15	Margules	31	UNIQUAC/UNIFAC
16	Regular Solution	32	UNIFAC LLE