REFPROP Documentation

Release 10.0

Eric W. Lemmon, Ian H. Bell, Marcia L. Huber, Mark O. McLinden

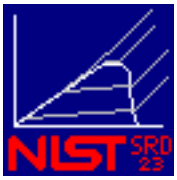
Jun 16, 2023

Contents

1 REFPROP Graphical User Interface **3**
1.1 General Information . . . 3
1.2 Menu Commands . . . 7
1.3 DLLs . . . 26

2 REFPROP DLL documentation **29**
2.1 High-Level API . . . 29
2.2 Legacy API . . . 57

Index **135**



REFPROP is an acronym for REFerence fluid PROPerties. This program, developed by the National Institute of Standards and Technology (NIST), calculates the thermodynamic and transport properties of industrially important fluids and their mixtures. These properties can be displayed in *Tables* and *Plots* through the graphical user interface; they are also accessible through spreadsheets or user-written applications accessing the *REFPROP dll*.

REFPROP is based on the most accurate pure fluid and mixture models currently available. It implements three models for the thermodynamic properties of pure fluids: equations of state explicit in Helmholtz energy, the modified Benedict-Webb-Rubin equation of state, and an extended corresponding states (ECS) model. Mixture calculations employ a model that applies mixing rules to the Helmholtz energy of the mixture components; it uses a departure function to account for the departure from ideal mixing. Viscosity and thermal conductivity are modeled with either fluid-specific correlations, an ECS method, or in some cases the friction theory method.

CHAPTER 1

REFPROP Graphical User Interface

1.1 General Information

1.1.1 About REFPROP

REFPROP is an acronym for REFerence fluid PROPerties. This program, developed by the National Institute of Standards and Technology (NIST), calculates the thermodynamic and transport properties of industrially important fluids and their mixtures. These properties can be displayed in *Tables* and *Plots* through the graphical user interface; they are also accessible through spreadsheets or user-written applications accessing the *REFPROP dll*.

REFPROP is based on the most accurate pure fluid and mixture models currently available. It implements three models for the thermodynamic properties of pure fluids: equations of state explicit in Helmholtz energy, the modified Benedict-Webb-Rubin equation of state, and an extended corresponding states (ECS) model. Mixture calculations employ a model that applies mixing rules to the Helmholtz energy of the mixture components; it uses a departure function to account for the departure from ideal mixing. Viscosity and thermal conductivity are modeled with either fluid-specific correlations, an ECS method, or in some cases the friction theory method.

The property formulations and fluid data files were programmed by:

Eric W. Lemmon, Ian H. Bell, Marcia L. Huber, and Mark O. McLinden
Applied Chemicals and Materials Division
National Institute of Standards and Technology
Boulder, CO 80305

Eric.Lemmon@nist.gov
Ian.Bell@nist.gov
Marcia.Huber@nist.gov

REFPROP 10.0 is the culmination of several years of revisions and updates. Work never stops on the development of thermophysical properties and equations, but the last two years have been especially intense and fully dedicated

to this release. Although there are only four authors of this work, we are very grateful to the many contributions of our NIST colleagues, including Gary Hardin, Allan Harvey, Chris Muzny, Vladimir Diky, Ala Bazyleva, and Janiel Reed who have provided support over the last several versions of REFPROP. Also of NIST are Adam Morey, Cindy McKneely, and Sherena Johnson who distribute the product for us to industry. A number of individuals from industry have contributed continuously over the last several years; we are indebted to them for their help, and we thank Tobias Loew, Nik Felbab, Nicolas James, Dan Williams, Jim Pollard, and Stuart Lawson.

We acknowledge our many colleagues whose property models we have taken from the literature, and without which this database would be much reduced in scope. In particular, the Ruhr University in Bochum, Germany, has for many decades worked alongside us in the development of equations of state. The contributions of Wolfgang Wagner, Roland Span, and Monika Thol can easily be seen by browsing through the fluid information. We thank Marc Assael of Aristotle University of Thessaloniki (Greece) for his many contributions to the development of transport property formulations, and Ryo Akasaka of Kyushu Sangyo University for his contributions to the refrigerant equations of state. We also thank our colleagues within our division whose efforts have made possible the NIST/TRC SOURCE and TDE Databases, of which we have made extensive use for our data needs required to develop thermophysical property equations.

Although REFPROP is a program built on equations of state, its entire existence is built on a foundation of experimental data, some of which dates back to the late 1800s. Through experimental measurements, especially the highly accurate values of Wolfgang Wagner, Reiner Kleinrahm, Martin Trusler, Mark McLinden, Markus Richter, their students and colleagues, and many others, equations are built that are then used throughout industry world-wide. Many things that touch our lives have been influenced in one way or another by these measurements. Power generation alone affects all, and the properties from these equations influence the efficiency and design of that infrastructure. Likewise, heating, cooling, and transportation have all been influenced by the measurements and subsequent property equations. We are greatly indebted to the enormous work of so many scientists and engineers that continues unseen by most.

The development of this software package was supported by the NIST Applied Chemicals and Materials Division and the NIST Standard Reference Data Program. The development of the models and the measurement of the data on which REFPROP is based have been supported over a period of many years by numerous sponsors.

IMPORTANT: Please visit the REFPROP FAQ web site as your first resource when you encounter difficulties or have questions. Most email enquiries are answered by pointing to the FAQ. Using the FAQ will save valuable NIST resources that can be used to further develop REFPROP.

*Certain trade names and other commercial designations are used in this work for the purpose of clarity. In no case does such identification imply endorsement by the National Institute of Standards and Technology, nor does it imply that the products or services so identified are necessarily the best available for the purpose.

1.1.2 Cautions

Users of the REFPROP program should be aware of several potential pitfalls:

If you experience large differences in your expected values of enthalpy or entropy as compared to those calculated by the program, see information on *Reference State*.

Changing the units in the Options/Units menu does not change the units on the tables already created, but only for new tables and plots.

The equation parameters for mixtures composed of natural gas fluids come from the 2008 GERG model (see *Preferences* for the reference). The default pure fluid equations of state in REFPROP are not the same as those used in the GERG model, rather they are more complex with lower uncertainties. The GERG equations for the pure fluids are shorter, less complex, and faster, but slightly less accurate. To use the GERG model, as published, choose the corresponding option under Options/Preferences. The preference screen also has an option to use the AGA8 model for natural gas calculations.

The NIST REFPROP program is designed to provide the most accurate thermophysical properties currently available for pure fluids and their mixtures. The present version is limited to vapor-liquid equilibrium (VLE) only and does not address liquid-liquid equilibrium (LLE), vapor-liquid-liquid equilibrium (VLLE) or other complex forms of phase

equilibrium. The program does not know the location of the freezing line for mixtures. Certain mixtures can potentially enter into these areas without giving warnings to the user.

Some mixtures have components with a wide range of volatilities (i.e., large differences in boiling points), as indicated by a critical temperature ratio greater than 2. Certain calculations, especially saturation calculations, may fail without generating warnings. Plotting the calculation results may reveal such cases–looking for discontinuities in density is a good check. Such mixtures, including many with hydrogen, helium, or water, may not have Type I critical behavior, that is they do not have a continuous critical line from one pure component to the other. The estimated critical parameters specified in the Substance/Fluid Information screen for these types of mixtures will not be displayed.

There are cases where an input state point can result in two separate valid states. The most common is temperature-enthalpy inputs. Viewing a T-H diagram will help show how there can be two valid state points for a given input. For example, nitrogen at 140 K and 1000 J/mol can exist at 6.85 MPa and at 60.87 MPa. When this situation occurs, REFPROP returns the state with the higher density. See the *Specified State Points* section for information on calculating the upper and lower roots.

There are certain properties pertaining only to the saturation line, such as dp/dT. For most cases, displayed properties at saturation states are those for the single phase on the saturation boundary. Thus, derivative properties at saturation as well as saturation properties that are given along constant property paths, such as Cv, Cp, or Csat, pertain to their state in the single phase. Those properties label with a '[sat]' indicate a path along the saturation line.

For pure fluids, when the 'Show 2-phase' option in the plot menu is selected, the generated lines for pressure and temperature represent metastable fluid states and the calculated lines between them. These are calculations from the equation of state disregarding any saturation states and generally have no physical significance.

Two equations of state are available for hydrogen to account for the different quantum spin states of the molecule. Normal hydrogen should be used in applications where it was created and stored at 250 K or above, or when it was cooled to below 250 K and stored without a catalyst for less than a day. The parahydrogen equation should be used where hydrogen was catalyzed or stored for several days at the normal boiling point (NBP) and used at any temperature within 1 day of storage at the NBP. Since the rate of conversion between quantum states is dependent on temperature, pressure, and the storage container, these values are only estimates. For more information, see the Leachman et al. literature reference in the Fluid Information window for hydrogen.

1.1.3 Copyright and Disclaimer

REFPROP is a product of the **National Institute of Standards and Technology (NIST)**.

Developed and maintained by:

Applied Chemicals and Materials Division
Boulder, CO 80305, USA

Distributed by

Standard Reference Data
Gaithersburg, MD 20899, USA

Copyright

Standard Reference Databases are copyrighted by the U.S. Secretary of Commerce on behalf of the United States of America. All rights reserved. No part of the database may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior permission.

Disclaimer

The National Institute of Standards and Technology (NIST) uses its best efforts to deliver a high-quality copy of the database and to verify that the data contained therein have been selected on the basis of sound scientific judgement. However, NIST makes no warranties to that effect, and NIST shall not be liable for any damage that may result from errors or omissions in the database.

1.1.4 REFPROP FAQ

The REFPROP FAQ web site (located at https://pages.nist.gov/REFPROP-docs/) provides detailed information on many aspects of using REFPROP. It is always the most current source of information. Providing answers to Frequently Asked Questions (FAQ), it is an important resource for new and experienced users alike. Please visit the FAQ web site as your first resource when you encounter difficulties or have questions. Most email enquiries are answered by pointing to the FAQ. Using the FAQ will save valuable NIST resources that can be used to further develop REFPROP.

1.1.5 First Time Users

The following information describes a procedure for quickly getting the REFPROP program set up and running. We encourage you to explore the many menu options and the Help file so you can obtain the most from the program.

Upon starting the program, a 'Credits' dialog box appears; click 'Continue'. Other information or warning boxes may appear, as appropriate. Once these have been dispatched, a blank screen appears within the program boundaries with a row of commands at the top. You should first select the fluid you wish to use by clicking on the *Substance* command and then on one of the options labeled *Pure Fluid*, *Predefined Mixture*, or *Define New Mixture*. If you select Pure Fluid, a dialog appears that lists compounds available in REFPROP. The number of fluids displayed can be substantial, making it more tedious to find what you are looking for. To decrease the number of fluids shown, select one of the predefined sets in the Substance/Specify Fluid Set menu, or set up your own combination using the 'Add New Set' option in that same menu. To permanently save your selection, select '*Save Current Options* ' from the *Options* menu and save the options as defaults.prf.

At this point you may calculate properties (using default settings) or customize the units system and properties displayed. To calculate, click on the Calculate command and then on the Specified State Points option. A blank grid appears on which you can enter properties. Once two inputs have been entered, the other properties in the active row will be calculated if it is a valid state point. The Saturation Points option is similar to the Specified State Points option except that only one input is needed and the properties will always be at saturation. Calculating tables is accomplished by clicking on either the Saturation Tables or Isoproperty Tables options.

To select which properties you would like displayed, click on the Options command, then on Properties. Under the 'Thermodynamic' tab, you can select properties like temperature, pressure, and enthalpy. Under the 'Transport, Misc.' tab, you can select properties such as viscosity, thermal conductivity, and surface tension.

If you would like to change the units you will use for the calculations, select the Options command, and then click on Units. The Units screen allows you to change each type individually, or use a set of predefined units. If you are using English units, you would click on the English button under the 'Reset Units' label. This dialog also allows you to select whether you will use mass or molar properties, and whether you will use a mass or molar basis to enter the compositions for mixtures.

If you desire to make plots, many predefined plots have been set up for you. For example, click on Plot then on P-h Diagram. You can then make changes to the dialog that appears and click OK. A plot will be generated based on your inputs.

All of these commands, plus others not described here are available in this Help file. The *Index* will help you find other information.

1.1.6 Status Line

The Status Line is displayed at the bottom of the screen. It displays the fluid name and reference state. For mixtures it also displays the components and composition. If the fluid (or mixture) applicable to the active window is different than the current fluid (or mixture), information on both is shown in the status line. The reference state for enthalpy and entropy values (selected with the *Reference State* command) is shown last. Clicking the mouse button on the Status Line brings up the *Fluid Information* window.

1.2 Menu Commands

1.2.1 Calculate

The Calculate menu holds commands for initiating the calculations of thermodynamic and transport properties of the *Substance*. Several modes of calculations are provided with the following commands.

Format, Column Position, and Automatic Data Entry

The format of the data in each column is set to a default style. You can change the format of the data in the column by right clicking the mouse in the column header. This action will bring up a window in which the format of the data in the column and the column position can be changed.

To change the format of the values in the column, click on the arrow to the right of the 'Style' field. A list of 'Default', 'Fixed digits', 'Fixed decimal', and 'Scientific notation' is shown. The 'Fixed digits' option allows a fixed number of total digits, including those before and after the decimal point. The 'Fixed decimal' option allows a fixed number of digits to be specified to the right of the decimal point. The 'Scientific notation' option allows the number to be displayed in that format. An example illustrating the display format appears to the right of the word 'Format'. The number of digits to be displayed can be changed globally with the *Preferences* command.

To change the column position, click the up or down arrows until the desired column position is shown. The positions of all other columns shift accordingly. To change the column position permanently, use the option in the *Substance* menu called *Property Order*. The units of the column can be changed using the 'Convert units' option. For mixtures, unit conversions requiring the molar mass of the fluid are not available. Note that you may not convert units when using specified state tables; however, the option to clear an entire column is available.

To automaticall fill a table with a range of numbers, enter a combination of values into the cells under the 'Specify values' section. The value to be placed in the first row of the selected row must be specified in the 'First value' field. Beneath this field are additional input boxes in which the last value in the range, the increment (or a multiplier), or last row can be specified. If 'Increment' is selected, the value of successive rows in the table is determined by adding the value in the increment field to the previous value. Similarly, if 'Multiply' is selected, successive values are determined by multiplying the previous value by the value indicated in the field. If 'Final value' is selected, the variable is varied in uniform increments from the specified initial value to the final value. In the special case where you wish to fill a certain number of rows with a constant, enter the starting row, the ending row, and the constant in the 'First value' field. The independent values entered must be in the unit system selected with the *Units* command. You can also specify that the values are saturated liquid or vapor states. This simply adds an 'l' or 'v' to the number as it places it in the field, as described in the *Specified State Points* dialog.

Isoproperty Tables

Two properties must be specified to fix the state of a fluid. The Isoproperty command allows one of the properties to be held constant while the other is varied in the specified manner. After the two input properties are specified and the OK button is selected, another window appears in which the value of the constant property and the range and increment of the varying property are specified.

After the second OK button is selected, calculations are initiated and a table displaying the results appears. The columns are initially shown in a preselected order and with a default format. The format and position of each column can be changed by right-clicking in the column header to bring up the *Format, Column Position, and Automatic Data Entry* dialog.

Select the 'Leave active' option to keep this window on top so that multiple isoproperty lines can be generated. Select the 'Add to current table' option to add the calculated values for each new isoproperty to the bottom of the current table.

Multiple tables (from different windows) can be plotted together by selecting all of the desired tables (hold down the Shift or Ctrl key) in the *New Plot* menu.

Saturation Points (bubble and dew points at same composition)

The blank grid allows you to select either the saturated temperature, pressure, or density to fix the state of the fluid. The remaining thermodynamic and transport properties corresponding to this saturation state will then be calculated. This menu item is only active for mixtures. The properties of the two phases will NOT be in equilibrium (see *Saturation Tables*). Use the *Saturation Points (at equilibrium)* table to calculate properties of phases in equilibrium.

The values of the independent properties can be entered in three ways (in addition to *Read Data from File*), as listed below.

1. Values can be simply typed into the cells of the property table.
2. Values can be automatically entered in a systematic manner by right-clicking on the header at the top of the column. This action brings up the *Automatic Data Entry* window.
3. Values can be pasted into the independent variable cells from another application, such as a spreadsheet. Copy the values to the clipboard using the other application's Copy command. Then, click the mouse in the first cell in the REFPROP table and select the *Paste* command.

Using the Saturation Points (bubble and dew points at same composition) Table

Whether the independent properties are read from a file or entered manually, as soon as a valid input property has been entered, the table will calculate the remaining properties and display them in the appropriate columns. Note that you can then change the input or select a new input (which causes the calculated values in that row to be erased) and repeat the calculation process.

The composition of the mixture can be entered on a line, but this must be done BEFORE the input property is entered. This can allow you to vary the composition and view the effect while holding temperature or pressure constant. The composition fields can be moved permanently to the beginning of the table by selecting the *Property Order* to make it easier to enter the values.

The position and format of any column can be changed, if desired, by right-clicking on the *column header*.

Saturation Points (at equilibrium)

The blank grid allows you to select either the saturated temperature, pressure, or density to fix the state of the fluid. The remaining thermodynamic and transport properties corresponding to this saturation state will then be calculated. For mixtures, the properties of the corresponding phase will be in equilibrium with the specified phase. The 'Composition' option should be selected (see *Properties*) to show the composition of the equilibrium phase.

Use the *Saturation Points (bubble and dew points at same composition)* table to calculate vapor and liquid saturated properties where the phases are at the same temperature, pressure, and specified composition. The liquid and vapor states in this table are not in equilibrium with each other.

The values of the independent properties can be entered in three ways (in addition to *Read Data from File*), as listed below.

1. Values can be simply typed into the cells of the property table.
2. Values can be automatically entered in a systematic manner by right-clicking on the header at the top of the column. This action brings up the *Automatic Data Entry* window.
3. Values can be pasted into the independent variable cells from another application, such as a spreadsheet. Copy the values to the clipboard using the other application's Copy command. Then, click the mouse in the first cell in the REFPROP table and select the *Paste* command.

Using the Saturation Points (at equilibrium) Table Whether the independent properties are read from a file or entered manually, as soon as a valid input property has been entered, the table will calculate the remaining properties and display them in the appropriate columns. Note that you can then change the input or select a new input (which causes the calculated values in that row to be erased) and repeat the calculation process.

For mixtures, the composition of the mixture can be entered on a line, but this must be done BEFORE the input property is entered. This can allow you to vary the composition and view the effect while holding temperature or pressure constant. The composition fields can be moved permanently to the beginning of the table by selecting the *Property Order* to make it easier to enter the values.

When entering temperatures or pressures, it is important that the value is placed in the appropriate column. Placing the temperature in the 'Liquid phase temperature' column will calculate a bubble point with its corresponding vapor state at some equilibrium condition. Entering the temperature in the 'Vapor phase temperature' column will calculate a dew point with its equilibrium liquid condition. These two lines will be completely different, and can be confusing if the compositions are not displayed.

The position and format of any column can be changed, if desired, by right-clicking on the *column header*.

Saturation Tables

There are several different types of saturation tables. For a pure fluid, when the Saturation Tables command is selected, the type of calculations to be made is specified by selecting one item from each list displayed. After the OK button is selected, another window appears in which the range and increment of the property to be varied are specified. After the second OK button is selected, calculations are initiated and a table displaying the results appears. The columns are initially shown in a preselected order and with a default format. The format and column position of each column can be changed by right-clicking in the column header to bring up the *Format, Column Position, and Automatic Data Entry* dialog.

For mixtures, the window contains additional items that allow you a variety of methods for calculating saturation conditions. The first option in the 'Composition basis' menu ('Liquid and vapor at the same composition') calculates state points that are NOT in equilibrium with each other. The composition of the liquid and vapor state points are the same as that of the specified mixture, and at the specified temperature or pressure. The second two options calculate properties that are in equilibrium, but that will be at different compositions (except in the case of azeotropes, or azeotrope-like mixtures). The composition should be displayed (see *Properties*) to view the full state of the system. Different properties can be selected to vary in the table. The first four items that can be selected under the 'Vary' label generate points at the composition of the mixture; the last two items calculate properties at different compositions and have no connection to the composition specified when the system was set up. The red dots in the example picture indicate which points will be calculated. The gray lines connect points that will be displayed on the same row of the table.

If the composition of the mixture is to be varied, an additional window will appear. The temperature is entered at the upper left. The composition will be linearly varied from the condition indicated in the 'Initial' column to that in the 'Final' column using the number of points indicated at the upper right. With 11 points, the composition of this binary system is varied from pure component 1 to pure component 2 with increments of 0.1 in mole fraction.

For solid-fluid saturation calculations (sublimation or melting), the properties displayed are the equilibrium fluid phase for either the liquid or the vapor. Properties for the coexisting solid are not available.

Specified State Points

The blank grid allows you to select two independent properties to fix the state of the fluid. The remaining thermodynamic and transport properties corresponding to this fixed state will then be calculated.

The values of the independent properties can be entered in three ways (in addition to *Read Data from File*), as listed below.

1. Values can be simply typed into the cells of the property table.
2. Values can be automatically entered in a systematic manner by right-clicking on the header at the top of the column. This action brings up the *Automatic Data Entry* window.
3. Values can be pasted into the independent variable cells from another application, such as a spreadsheet. Copy the values to the clipboard using the other application's Copy command. Then, click the mouse in the first cell in the REFPROP table and select the *Paste* command.

Using the Specified State Points Table Whether the independent properties are read from a file or entered manually, as soon as two valid input properties have been entered, the table will calculate the remaining properties and display them in the appropriate columns. Note that you can then change the inputs or select new inputs (which causes the calculated values in that row to be erased) and repeat the calculation process.

Saturation or melting-line states can be calculated by adding the letters 'l', 'v', or 'm', respectively, before or after the value of temperature, pressure, density, enthalpy, or entropy. Note that 'm' returns liquid properties at the solid-liquid boundary. REFPROP does not calculate solid properties. In the case where 'l' or 'v' is entered, and there is only one valid liquid or vapor state point, that point is returned regardless of whether or not it is in the liquid or vapor state. If a liquid density is entered with a 'v', the liquid state is calculated internally; however, the vapor state is returned. The same is true for a vapor state entered with 'l'. The properties at the critical point can be calculated easily by entering 'cr', and the properties at the triple point can be calculated by entering 'tr' (liquid properties can be obtained with 'trl' and vapor properties with 'trv').

For state points that are double valued (see the *Cautions* menu for more information), the character '>' or '<' can be added to the number (before or after) to find the lower or upper root, respectively. These same characters can also be used to find metastable fluid states for temperature and pressure inputs. For example, the saturation pressure for nitrogen at 100 K is 0.778 MPa. Inputs of 100 K and 0.777 MPa will return a vapor state, but inputs of '100' for temperature and '0.777>' for pressure will return a metastable liquid state.

For mixtures, the composition of the mixture can be entered on a line, but this must be done BEFORE the two valid state points are entered. For convenience, the Specified State Points (varying composition) command can be chosen. This command places the composition fields at the beginning of the table (and activates them if they were not selected in the *Properties* menu). The composition fields can be moved permanently by selecting the *Property Order*.

The position and format of any column can be changed, if desired, by right-clicking on the *column header*.

Tables

Tables can be generated from a variety of options in the REFPROP program, including *Saturation Tables*, *Isoproperty Tables*, *Specified State Points*, *Saturation Points (at equilibrium)*, and *Saturation Points (bubble and dew points at same composition)*. The number of digits displayed, the number format, and the units can be changed by right-clicking on the column header. This will display the *Format, Column Position, and Automatic Data Entry* window. Also accessible from this window is the ability to generate a sequence of numbers, either in steps or exponentially.

The height and width of the rows and columns can be changed by placing the mouse over the line separating two rows or columns, and dragging the line to a new position. The column widths are also enlarged in some operations when the width of the numbers in the column exceeds the column width.

Rows can be deleted by pressing Shift-Del or by using the appropriate menu item (see *Inserting and Deleting Rows*). This action permanently removes the row, and a *Preference* can be set to warn before a row is deleted.

Selecting the contents of the entire table can be done either through *Select All* or by clicking the uppermost left cell and dragging the mouse. Clicking the row number or column heading selects the entire row or column. The *Copy All or Selected Table Data* command will copy the text to the clipboard. Text can be pasted into some tables with the *Paste* command.

An averaged value of a selected set of cells can be obtained by clicking the right mouse button once the cells have been selected.

1.2.2 Edit Menu

The Edit menu contains commands to copy and paste information between applications. Click on one of the following command names for details.

Copy Plot

The Copy Plot command copies a selected plot to the clipboard, which can then be pasted into other applications. The quality of the plot depends on the screen resolution; the copy is simply a bit-mapped image of that displayed on the screen. For higher quality outputs, see the *Print Setup and Dialog* command.

Copy All or Selected Table Data

The Copy All Table Data command copies all of the information in a table to the clipboard. The descriptive column headings will be included with the numerical data if this option has been selected in the *Preferences*.

The Copy Selected Table Data command copies the selected section of a table to the clipboard. If none of the cells have been selected, only the value in the current cell will be copied to the clipboard. To copy part of the data in a table, select the cells to be copied by clicking in the upper left cell followed by a click in the lower right cell with the Shift key held down (or simply click and drag the mouse). Selected cells are displayed in inverted color. Alternatively, use the *Select All* command to select all of the cells in the table. Once the desired cells are selected, use the *Copy All or Selected Table Data* command to copy them to the clipboard.

Tab characters are used as separators between the values copied to the clipboard. A return character is placed at the end of each row. With this format, the selected data can be pasted into most spreadsheet programs and word processors.

Inserting and Deleting Rows

The Insert Row command inserts a blank row into the current table where the cursor is located. The Delete Row command allows a specified row to be permanently removed from the table. In both cases, the rows are not renumbered. These commands are accessible only if a window displaying a table is active.

Paste

The Paste command is used to enter values contained in the clipboard into open tables in the program. This command is typically used in conjunction with a Copy command from a spreadsheet application. Paste can be used only in the *Specified State Points* or *Saturation Tables* tables. Values in the same row must be separated by tabs. A return character must be used to separate data in successive rows.

Read Data from File

Values can be read from an ASCII file and placed in an open table. An Open file dialog appears in which you may specify the file containing the independent property values. The box at the top of the window displays the first 20 lines of the selected file so that you can see the format of the values in the file. Although only 20 lines are shown, all of the lines are read in when you press the OK button. Lines in the file that begin with a nonnumeric character, such as *, are treated as comments and are ignored.

Data are read from the file and inserted into the table depending on the inputs under the 'Read column' and 'Insert data into field' options. The column specified under 'Read column' will be entered into the field specified under 'Insert data into field'. It is not necessary to read all of the input properties from the file. For example, you can read the temperature but not the pressure by selecting the blank line (at the top of the drop down box) in the second (and subsequent) rows.

Extra data from the file can be read and placed in a comment column in the *Specified State Points* table. The capability to read additional data from a file is useful in comparing experimental and calculated values of a property, for example. Before this can be done, the 'Add a comment column to tables' option must be specified in the *Preferences*, and a new table containing the comment field must be created. The comment field can then be specified in one of the boxes under the 'Insert data into field' label. They can then be plotted along with calculated values.

Save Plot Data Points

The Save Plot Data Points command will save to a file all of the data points used in generating a plot. The first column is the x-axis value, the second column is the y-axis value, and the third column indicates whether or not to connect the point with the previous one. This command is not intended for general use.

Select All

The Select All command selects all of the cells in a table. The selected cells are shown in inverted color. They can then be copied to the Clipboard using the *Copy All or Selected Table Data* command.

1.2.3 File Menu

The File Menu contains commands to save, retrieve, and print results. Click on a menu item below for additional information on that topic.

Close

The Close command deletes all tables and plots from the display, but the program itself continues. The program will prompt you to *Save* all open windows and settings.

Save

The Save command saves the current state of the program to a file with a .RFP filename extension. This file can later be recalled with the *Open* command. All current information is saved, including the *Units*, *Reference State*, *Properties*, *Property Order* and *Substance* as well as all *Isoproperty Tables* and *Plotting Data*. You will be asked if you want to save when you exit the program if the 'Prompt to save session when closing' option is selected in the *Preferences*.

Open

The Open command reads a file previously saved with the *Save* command. The files are identified by a .RFP filename extension. The Open command restores the program to the state it was in when the *Save* command was issued. Thus, the *Units*, *Reference State*, *Properties*, *Substance*, and all *Isoproperty Tables* and *Plotting Data* are restored. Note that the Open command will replace all settings and windows of the current session with those from the previously saved file; REFPROP will prompt the user to *Save* the current session before opening the saved session.

Print Setup and Dialog

The Print Setup command allows the selection of the printer and printer options, such as landscape or portrait formats. The selected printer and options are applied to the output when the *Save Tables and Print Tables* command is issued.

The Print Dialog command allows for a few additional options while printing, most important is the 'Print to file' option. This option sends the output to a file that can then be used in such applications as making PDFs. Unlike the Print Setup command, the Print Dialog command first displays the menu for selecting tables or plots to print, then activates the dialog menu before printing.

Save Tables and Print Tables

The Save Tables command displays a dialog that allows the data in one or more of the current tables to be saved to a file in ASCII format. The selected delimiter is placed between each value in a row. Most spreadsheet programs understand the tab character as a separator for items in the same row. The return character is used to signify the end of the row. When the data in this format are read from the file into a spreadsheet application, each value goes into a separate cell. If 'Spaces' is selected as the delimiter, the width of the columns can be specified.

The Print command allows one or more of the tables and/or plots to be printed. Use the *Print Setup and Dialog* command before selecting the Print command if the default printer or printer options (paper size, orientation, etc.) need to be changed.

When printing or saving tables, the previously generated tables (and plots when printing) are shown in a list at the left. Click on the names of the tables or plots you want to print and then click the '>' button (or press the period or '>' key) to move them from the left list to the right list. If you click the '>>' button, all of the tables and plots in the left list will be moved. To remove tables or plots from the selected list, select the names and then click the '<' button (or press the comma or '<' key). The tables and plots in the selected list are saved or printed in the order in which they appear in the list.

To save or print the column headings (the property type and units), select the 'include column heading' option.

When saving tables, the 'Full width' option prints each line of the table on a single line of the file. The 'Page width' option formats the tables so that all the properties fit within 80 columns in the file. The lines are broken up and printed in different sections, similar to that done during printing.

1.2.4 Help Menu

The Help menu holds commands for accessing the Help processor. In addition, a command is provided to display the startup screen, which contains the program version number and other information.

Citing REFPROP

The following reference can be used to cite the REFPROP program in publications:

Lemmon, E.W., Bell, I.H., Huber, M.L., McLinden, M.O. NIST Standard Reference Database 23: Reference Fluid Thermodynamic and Transport Properties-REFPROP, Version 10.0, National Institute of Standards and Technology, Standard Reference Data Program, Gaithersburg, 2018.

Or in BibTeX form:

```
@Misc{LEMMON-RP10,
  Title            = {{NIST Standard Reference Database 23: Reference Fluid␣
↪Thermodynamic and Transport Properties-REFPROP, Version 10.0, National Institute of␣
↪Standards and Technology}},
  Author           = {E. W. Lemmon and I. H. Bell and M. L. Huber and M. O. McLinden},
  Year             = {2018},
  Doi              = {https://doi.org/10.18434/T4/1502528},
  Url              = {https://www.nist.gov/srd/refprop}
}
```

Additionally, users should cite the reference for either the equation of state or the transport equations used in their work, for example, if you used calculations for carbon dioxoide, you will find the reference for the equation of state under the `Options/Fluid` Information option in the REFPROP GUI and you should cite the reference given under that option, like this:

Span, R. and Wagner, W., *J. Phys. Chem. Ref. Data*, 25(6):1509-1596, 1996.

1.2.5 Options

The items in the Options menu allow selection of the unit system, reference state, preferences, properties to be calculated, and the order that the properties are shown in the tables. Click on the desired command for additional information:

Preferences

The Preferences dialog allows various options to be selected:

If the 'Prompt to save session when closing' option is selected, you will be prompted when the program is terminated to *Save* before closing. This file can then be retrieved using the *Open* command when the program is run again.

The 'Prompt before deleting row' option asks for confirmation (after either pressing Shift-del or selecting the *Inserting and Deleting Rows* option while editing a table) before a row is deleted. A row cannot be recovered after it has been deleted.

The 'Ignore all error messages' option will cause all error and warning messages to be ignored that occur during property calculations. While this option may relieve you of having to clear each error message, it may cause confusion when properties are not returned after entering inputs out of range or when the program does not converge. The 'Ignore all warning messages' option will cause all warning messages to be ignored, but will still display the serious error messages.

The 'Copy table headers to clipboard with table data' option can be used to select whether or not the headers at the top of each table are included when *copying tables*.

When the 'Show saturation boundaries in tables' option is selected, a blank line will appear between the single and two-phase state points and the boundary between the liquid and vapor phases.

Steam conversion option. This option determines which conversion is used for calorie (cal) and British thermal unit (Btu). The International Table (IT) definition is generally used in the literature with the calculation of water and steam properties. The thermochemical (th) definition is generally used with other fluids and mixtures.

Steam conversion option	Conversion	Corresponding Btu variant
Not selected	1 cal(th) = 4.184 J	1 Btu(th) = 1054.35026448889 J
Selected	1 cal(IT) = 4.1868 J	1 Btu(IT) = 1055.05585262000 J
For reference:		
4.1868 · 453.59237 · 5 / 9 = 1055.05585262000		
4.184 · 453.59237 · 5 / 9 = 1054.35026448889		

In certain applications, such as using Asian settings with Microsoft Windows, the degree sign, the superscript 2, and the superscript 3 may not be properly displayed, causing unit conversions to not work and displaying units with funny symbols. In such cases, the nonstandard characters should be turned off (by clicking on the 'Use nonstandard symbols' option), resulting in the loss of the degree signs. The text '^2' and '^3' will be used in place of the superscript 2 and 3.

The 'Reset bounds when units or fluid are changed' option will reset the starting and ending default values that are displayed when the saturation or isoproperty table menus are brought up after the units or fluid have been changed. The values in the plot menus will also be changed to better default values based on the fluid or units selected. Deselect this option to preserve the bounds that you have set up.

The 'Flip usage of commas and periods in numbers' allows the use of a comma as a decimal point. For the most part, REFPROP will recognize the user's country settings and this option will not be needed.

The 'Add Notes button to graphs' option will add a small command button in the upper left corner of each plot that allows the user to add notes to a particular graph. These notes will be printed out under the plot when File/Print is selected. If this option is deselected, the notes that correspond to each plot will not be lost, but will not be accessible.

The option to 'add a comment column to tables' will add one or more extra columns (up to 10) in each table that can be used to enter comments or user data. The values in these fields will be ignored, but numerical values can be plotted in the same manner as calculated properties. Adding text in each field can aid in labeling each state point, and the size of the comment field can be enlarged as explained in *Tables*.

The 'Show options used for analyzing Equations of State' will add the rectilinear diameter to all plots, additional plot items, and the 'Show 2-phase' option for plotting metastable states for pure fluids.

The mixture models used in REFPROP vary by type of mixture (e.g., refrigerant mixtures use different models than hydrocarbon mixtures). The mixture model can be specified by selecting the appropriate radio button in the Preferences menu. Constituents typically found in natural gas are modeled by the latest GERG model (completed in 2004 and extended in 2008). The mixing part of the model (the excess contribution) is similar to that used for the refrigerant mixtures. The pure fluid equations of state have been shortened and differ from the default equations in REFPROP. The calculations using these shorter equations are somewhat faster than the default equations, but are slightly less accurate. If the user wishes to use the shorter equations specified by the GERG-2008 model, the 'Use full GERG-2008 natural gas mixture model' option should be selected. This will result in slightly different values than the default values. In either situation, the mixing interaction parameters remain the same. The documentation for this model was published in the following:

Kunz, O., Klimeck, R., Wagner, W., Jaeschke, M. The GERG-2004 Wide-Range Reference Equation of State for Natural Gases and Other Mixtures. GERG Technical Monograph 15, Fortschr.-Ber. VDI, VDI-Verlag, Düsseldorf, 2007.

Kunz, O. and Wagner, W. The GERG-2008 Wide-Range Equation of State for Natural Gases and Other Mixtures: An Expansion of GERG-2004. J. Chem. Eng. Data, 57(11):3032-3091, 2012.

Calculations from the American Gas Association program for natural gas mixtures can also be made by selecting the 'Calculate mixture properties using the AGA8 equation'. This equation is not valid in the liquid phase or in the extended region near the critical point and users of this model should be aware of the uncertainties in the equation in the various regions where they calculate numbers. The default equations in REFPROP are used to calculate the phase boundaries since the AGA8 model does not allow this calculation.

The 'Use Peng-Robinson cubic equation of state for all calculations' option switches from the default equations to the less accurate Peng-Robinson equation. This equation is not recommended for general use in REFPROP.

In addition to these options, the number of digits in the tables can be selected on a global basis, rather than individually selecting the number of digits as described in the *Format, Column Position, and Automatic Data Entry* menu. Changes in the number of digits displayed will be applied to the currently displayed tables if the 'Apply to existing tables' option is selected. The 'Fixed digits' option allows a fixed number of total digits, including those before and after the decimal point. The 'Fixed decimal' option allows a fixed number of digits to be specified to the right of the decimal point. The 'Scientific notation' option allows the number to be displayed in that format.

The font size used in the tables can be specified. The size in previously generated tables will be changed if the 'Apply to existing tables' option is selected. The label font size used in the graphs can also be modified.

The preferences are saved when the *Save Current Options* command is issued. You can restore options at any time with the *Retrieve Options* command.

Properties

The Properties dialog provides a check box for each of the properties that can be calculated. If the box to the left of the property name is checked, calculations are done for that property and the values of the property are displayed in the tables. To select or unselect a property, click the left mouse button in the box.

Most thermodynamic and transport properties can be calculated. The properties that can be displayed are divided into five categories: thermodynamic properties; transport properties plus surface tension, dielectric constant, and heating values; derivatives of the thermodynamic properties (including isothermal compressibility and volume expansivity); special properties; and mixture properties.

Thermodynamic properties include temperature (T), pressure (p), density (D), volume (V), internal energy (e), enthalpy (h), entropy (s), isochoric (Cv) and isobaric (Cp) heat capacities, ideal gas isobaric heat capacity (Cp0), ratio of Cp to Cv, speed of sound (w), compressibility factor (Z=p/DRT), Joule-Thomson coefficient (JT=dT/dp at constant h), quality (q=ratio of vapor moles to total moles), 2nd virial coefficient (B), 3rd virial coefficient (C), Helmholtz energy (a), Gibbs energy (g), and heat of vaporization (hvap). Most of these properties are given on a molar or mass basis, e.g., V=volume/number of moles.

Transport properties include the thermal conductivity (ThC), viscosity (Vis), kinematic viscosity (Vis/D), thermal diffusivity (ThC/D/Cp), and Prandtl number (Vis*Cp/ThC). The surface tension, dielectric constant, and heating values are included under the transport properties menu. The gross heating value is the amount of heat produced by the complete combustion of a unit quantity of fuel, while the net heating value is obtained by subtracting the latent heat of vaporization of water vapor from the gross heating value.

Derivative thermodynamic properties include the isothermal compressibility [kappa=1/kt/p= -(dV/dp)/V at constant T], volumetric expansivity (also known as the thermal expansion coefficient) [beta=(dV/dT)/V at constant p], isentropic expansion coefficient [k=w^2*D/p=-V/p(dp/dV) at constant s], isothermal expansion coefficient [kt=D/p(dp/dD) at constant T], adiabatic compressibility [betas=1/k/p=-(dV/dp)/V at constant s], adiabatic bulk modulus [Bs=k*p=-V(dp/dV) at constant s], isothermal bulk modulus [Kt=kt*p=-V(dp/dV) at constant T], and isothermal throttling coefficient (-JT*Cp = dh/dp at constant T). Additionally, properties contained under the derivative tab include most of the common derivatives of density, pressure, temperature, and enthalpy with respect to each other. The property dp/dT [sat] is the derivative of the vapor pressure with respect to temperature. The property dh/dZ [sat] is the Waring function (as given in Ind. Eng. Chem., 46:762, 1954) and dB/dT is the derivative of the second virial coefficient with respect to temperature (B is only a function of T).

Special properties include the acoustic virial coefficients, the negative reciprocal temperature (-1/T), exergy on a flow basis (H - T0S), exergy for a closed system (H - P0V - T0S) (see *Reference State* about setting the exergy reference state), specific heat input [V(dh/dV) at constant P], Cv in the two-phase region (Cv2phase) (only calculated for pure fluids), supercompressibility factor (Fpv=square root of the compressibility factor at 60 F and 14.73 psia divided by the square root of the compressibility factor at T and p), critical flow factor (the normalized sonic mass flux for inviscid, one-dimensional, steady, isentropic flow), and the internal pressure [T(dp/dT)-p at constant D]. The ideal gas values of

density, enthalpy, entropy, Gibbs energy, and Helmholtz energy are also available (note that these are the ideal values at T and p, not the ideal values at zero pressure such as is the case with Cp0).

Mixture properties include fugacity (f), fugacity coefficients [f(i)/x(i)p], chemical potential, K value (vaporization equilibrium ratio - the ratio of the vapor mole fraction of a species to the liquid mole fraction of the species), molar mass (molecular weight), composition (on either a mole basis or mass basis), and several different excess functions (volume, energy, enthalpy, entropy, Helmholtz energy, and Gibbs energy).

Within the 'Special' properties tab is the ability to calculate values for metastable fluid states (such as a liquid that has been superheated beyond the saturated state). These include pressure, enthalpy, entropy, Helmholtz energy, Gibbs energy, and compressibility factor. For single phase states, the values of these properties will be identical to those calculated under the 'Thermodynamic' properties tab. However, in the two-phase region, the values given are the metastable fluid states, or more precisely, the properties calculated from the equation of state at the given temperature and density. The easiest way to generate these values is to calculate an isotherm in the *Isoproperty Tables* command, varying the density. The two-phase pressure for a pure fluid, calculated from the equation of state, can also be viewed by plotting a p-d *diagram* with the 'Show 2-phase' option checked.

The 'Bulk, liquid, and vapor properties' option displays three columns for most properties during calculations. The first column gives the overall, or bulk, value of the property, the second shows the properties of the liquid phase, and the third column gives the properties of the vapor phase. This is generally useful only when calculating state points within the two-phase region, such as a temperature and a two-phase density.

Several properties can only be calculated at the saturation states, such as the heat of vaporization and surface tension. Other properties, such as heat capacities and sound speeds, are not available in the two-phase region since they are not thermodynamically defined for two-phase states. However, the property Cv2phase is available, but its definition is quite different from that of Cv, and the two properties are discontinuous at the saturated state.

The properties are saved with other *Preferences* when the *Save Current Options* command is issued. They are restored to a previously saved option with the *Retrieve Options* command, or *Open* command.

References for the equations that calculate all of these properties can be found in the fluid information screen, with one exception: the dielectric constant. The equations for this property are documented in: Harvey, A.H., Lemmon, E.W. Method for Estimating the Dielectric Constant of Natural Gas Mixtures, *Int. J. Thermophys.*, 26(1):31-46, 2005, https://doi.org/10.1007/s10765-005-2351-5 and Harvey, A.H., Mountain, R.D., Correlations for the Dielectric Constants of H2S, SO2, and SF6, *Int. J. Thermophys.*, 38, 2017, https://doi.org/10.1007/s10765-017-2279-6

Property Order

The property order dialog defines the order of the properties that will appear when a table is generated. Select the property that you wish to move, and use the 'Move up', 'Move down', 'Move to top', and 'Move to bottom' buttons to change the property order. The property order can also be changed by selecting a property with the up and down arrow keys, and then moving the property using the up and down arrow keys while holding down the shift key.

The property order is saved with other *Preferences* when the *Save Current Options* command is issued. It is restored to a previously saved option with the *Retrieve Options* command, or *Open* command.

Reference State

Reference State

When large differences in enthalpy and entropy occur between REFPROP and tables of properties given in handbooks, please note:

The absolute values of enthalpy, entropy, and energy at a single state point are meaningless. It is only the difference between two different state points that matter. Thus, the value for a single state point can be set to any arbitrary value. Many handbooks set the arbitrary state point so that the values of these properties are positive for most liquid or gas states. You can change the values of the arbitrary state points by going to the Options/Reference State menu. Your

choice of options can be permanently saved in the graphical interface by selecting Options/Save Options, and then saving the options under the file name 'defaults.prf'.

To change the default in the Excel file, press Alt-F11 to bring up the VB code. If the code does not appear, make sure the project explorer is visible (View/Project Explorer), then click on modules, and then on module1. Then search for the call to SETREF, and change the second input from 2& to 1&. More information on this can be found in your REFPROPFortran directory in the file SETUP.FOR under the SETREF subroutine.

The reference states for enthalpy and entropy are entered in the Reference dialog. There are three common choices for the reference state on which the values of enthalpy and entropy are based. These three common choices are:

1. Setting enthalpy and entropy to zero for the saturated liquid at the normal boiling point (designated as NBP).
2. Setting enthalpy and entropy to zero for the saturated liquid at -40 °C (designated as ASHRAE).
3. Setting enthalpy to 200 kJ/kg and entropy to 1.0 kJ/(kg-K) for the saturated liquid at 0 °C (designated as IIR).

In addition, the reference state can be set manually by specifying values of the enthalpy (h) and entropy (s) at arbitrary values of temperature (T) and pressure (P).

When working with mixtures, the default setting can be used for each pure fluid, or the reference state values of enthalpy and entropy can be specified at the currently defined composition.

The exergy reference state and definition can be selected based on the user's preference.

The reference state choices are saved with other *Preferences* when the *Save Current Options* command is selected. It is restored to a previously saved option with the *Retrieve Options* command or *Open* command.

Retrieve Options

The Retrieve Options command brings up a dialog from which a preference file (identified with a .Prf filename extension), previously stored with the *Save Current Options* command, can be selected. The *Units*, *Reference State*, *Properties*, *Preferences*, and *Pure Fluid* are changed to the specifications in the selected options file.

The Defaults.Prf file is automatically loaded at program startup. After you have set up the reference state, properties, preferences, and fluid that you would like to use on a general basis, save these options by choosing the *Save Current Options* command and enter Defaults.Prf as the file name.

Save Current Options

The Save Current Options command brings up a dialog from which a file name can be chosen, identified by a .Prf filename extension. The file stores the choices made in the *Units*, *Reference State*, *Preferences*, *Properties*, and *Property Order* dialogs. In addition, the current fluid, selected with the *Pure Fluid*, *Predefined Mixture*, or *Define New Mixture* commands, and other various settings are stored.

The program reads the Defaults.Prf file at startup. If you want to change the preferences that appear when the program is started, save your preferences in the Defaults.Prf file. You can restore options with the *Retrieve Options* command.

Units

The Units command will bring up the Unit System dialog in which the units of each *Properties* can be specified. To change the individual units, click on the arrow to the right of the current unit setting. A drop down list of possible units appears. Click on the desired unit. The changes made in this dialog only affect subsequent calculations. The data in existing tables are not affected by the changes.

Multiple buttons are available for selecting default sets of unit; for example, standard SI or English units. Other sets include standard SI units with Celsius for temperature, a mixed set of SI and English units (with pressure in psia, but

other units in SI), and an option to use unitless dimensions for some variables, including temperature divided by the critical temperature, pressure divided by the critical pressure, and so on.

Specific (mass) or molar properties can be selected under the 'Properties' section. When calculating properties of mixtures, the composition can be entered on either a molar or a mass basis, depending on the selection in the 'Composition' section.

Pressures can be expressed as gage pressures by selecting the 'Use Gage Pressure' option. The atmospheric pressure or elevation at the location of interest is required to use this option.

The units are saved with other *Preferences* when the *Save Current Options* command is issued. They are restored to a previously saved option with the *Retrieve Options* command, or *Open* command.

1.2.6 Plot Menu

The Plot menu provides commands to plot values in any existing table. In addition, many plots, such as temperature-entropy, temperature-enthalpy, and pressure-enthalpy plots can be generated automatically. The commands in this menu are listed below. Click on the command name for more information.

The appearance of the plot window can also be changed by the *Plotting Data* controls.

Add Label

The add label window allows text to be placed on a plot window or allows text on a current plot to be modified. The plot window where the label will be added must be the active window. The text to be placed on the plot window is entered in the edit field at the top. Other text characteristics may be entered, such as the font type, font size, and style. When the OK button is clicked, the text appears at the center of the plot window. It can then be dragged to any location in the plot by pressing and holding the mouse button down on the text item and then by dragging it to its new location. The coordinates of the label are displayed if the shift key is held down as well.

The degree character can be entered into the text of the label by pressing Alt-248 (or Alt-0176) on the numeric keypad. Alt-253 will add a superscript '2' to the text. Also, Alt-0178 will add a superscript '2' and Alt-0179 will add a superscript '3'. Other special characters of interest using the Alt key include 225: ß, 230 (or 0181): µ, 241 (or 0177): ±, and 246: ÷.

You can change the characteristics of the text item at any time by double clicking on it. This window will reappear and allow you to make the changes. The label can also be deleted. To delete all labels on a plot, double click one of the labels and select the Delete All button. See the *Plotting Data* description for additional plot window controls.

Diagrams

Diagrams can be generated for the selected fluid. The types of diagrams that can be plotted include:

- Temperature vs. Entropy
- Temperature vs. Enthalpy
- Temperature vs. Density
- Pressure vs. Enthalpy
- Pressure vs. Density
- Pressure vs. Volume
- Pressure vs. Temperature
- Compressibility Factor vs. Pressure

- Enthalpy vs. Entropy
- Isochoric Heat Capacity vs. Temperature
- Isobaric Heat Capacity vs. Temperature
- Speed of Sound vs. Temperature
- Exergy vs. Enthalpy
- Isothermal Compressibility vs. Temperature
- Viscosity vs. Temperature
- Thermal Conductivity vs. Temperature
- Temperature vs. Composition (for binary mixtures only)
- Pressure vs. Composition (for binary mixtures only)

Each diagram type will allow different combinations. In the case of a pressure vs. enthalpy diagram, lines of constant temperature, density, entropy, and quality can be displayed. To specify plotting isotherms, place a check in the box to the left of the label. You can then specify a range of temperatures to be plotted and/or up to 12 individual isotherms. In addition, values in the two-phase region can also be plotted. Generally, this option is not selected, and a straight line is drawn between the saturated liquid state and the saturated vapor states. For mixtures, showing two-phase values will be very computationally intensive. When plotting isotherms for a pure fluid, the two-phase option works differently from what might be expected: values of the metastable fluid states are shown, both real and inaccessible. However, you should realize that there are virtually no data for metastable states of most fluids, and values shown will be extrapolations of the equations of state. In addition, calculated values can be extremely large, and plotting these values can result in unexpected plots in the two-phase region. See the *Properties* dialog for additional information.

For cases where multipliers are preferred rather than increments in the 'From... To... Step' inputs, the character '*' can be added in front of the step input. Thus, inputs of 'From: 0.001, To: 1000, Step: *10' would produce isolines of 0.001, 0.01, 0.1, 1, 10, 100, and 1000. The More/Less button can be selected to include/exclude additional boxes where property values can be individually added.

The starting and ending values of the x and y axis can be modified. These values can be changed later with the *Modify Plot* command; however, some data may not be calculated beyond the ranges indicated in these boxes, and a new diagram will have to be generated with larger bounds.

Labels identifying the isoproperties are automatically generated if the 'Include labels' option is selected. You can change or delete these text items or add additional text using the *Add Label* command. If too many labels are being placed on a plot, the 'Unlabeled lines' option can be used to specify that labels should occur on every other line, every third line, and so on. The appearance of the plot can be modified with the *Modify Plot* command and with the controls in the *Plotting Data* window.

The amount of data calculated, resulting in fine or coarse plots, can be selected in the 'Point spacing' option. Several additional options are available to control whether the saturation lines are drawn, the melting line is drawn, and the saturated liquid and vapor states are connected by lines. The 'Swap density for specific volume' option will change the density inputs to volume inputs. The 'Add s and v lines at saturated temperature' option will draw a vapor phase isochore and a vapor phase isentrope for each isotherm that is plotted, using the value of volume and entropy at the saturated vapor state point.

Once a plot has been generated, other data may be superimposed on it using the *Overlay Plot* command.

Other diagrams

Other properties can be plotted as well using these same techniques. Under the 'Axis scaling' label in the plot dialog, the x and y properties can be changed by selecting the down arrows and clicking on the desired property. Most of the properties available in the *Properties* dialog can be plotted on the y axis. Properties allowed on the x axis include temperature, pressure, density, volume, energy, enthalpy, entropy, compressibility factor, exergy, -1/T, and composition. In order to avoid problems where one of the four columns at the top of the plot dialog matches one the

properties on the x or y axis, you should carefully select as a starting point one of the diagrams in the Plot pull down menu that contains your x or y property. For example, to generate a plot showing the Prandtl number vs. temperature, you could start with a T-s diagram so that the upper four columns show pressure, density, enthalpy, and quality. If a T-h plot was selected, then the third column would contain entropy instead of enthalpy. Once this is done, the x axis property should be changed to temperature and the y axis property should be changed to the Prandtl number. Even better in this case would have been to select a Cv vs. T plot so that temperature is already placed in the x-axis option. Once the initial plot is setup, you can then go to the Plot/Other Diagrams option to modify your previous settings.

Modify Plot

Many of the attributes of a plot can be changed through the Modify Plot command. This command can be accessed by the pull-down menu item or by double-clicking anywhere within an existing plot. In the former case, the plot in the topmost window is affected. If the topmost window is not a plot, the most recent plot is acted on.

The dialog for this command is similar to that used to specify a new plot, except that the variables plotted cannot be changed (although the x- and y-axis labels can be edited). As with the New Plot command, linear or logarithmic scaling can be selected, and the plot limits and axis label formats can be changed. Options to modify the size of the axis labels are provided below the "X-Axis" and "Y-Axis" portions of the dialog.

A list box in the upper right lists the individual data sets plotted. Selecting an item allows editing of the line and symbol attributes for that data set. The Remove button permanently deletes the selected data set. (Note that for the predefined plots, there is an entry for each of the lines composing these diagrams.) Select the data set for which the line type, symbol, and/or color specifications are to be applied. These changes can be made to one data set at a time or to a multiple selection (by shift-clicking). After changes have been made, either the Apply button or the OK button can be selected. By pressing the Apply button, other changes can be made as well. The changes will not appear on the plot until the OK button has been pressed.

The Highlight button allows you to quickly change the selected line or lines to red with a larger line width. This button immediately exits the Modify Plot command and emphasizes the desired data.

Other options in the Modify Plot menu are described in the *New Plot* menu.

New Plot

This command generates a new plot window in which the data in a column of a table can be plotted against data in any other column of the same table. The *Overlay Plot* command plots data within an existing plot window.

By default, it is assumed that data from the active table will be plotted. (The 'active table' is the table that was last accessed and is on top of the other tables.) However, the table from which the data are taken can be selected in the table list at the upper right of the window. It is also assumed that all of the data in the table are to be plotted. If this is not the case, it will be necessary to enter the starting and ending row numbers in the 'From row' and 'To row' fields.

Select the variables to be plotted on the x axis and y axis by clicking on their names in the respective lists. Appropriate minimum, maximum, and interval values are supplied for the selected x and y variables. However, these values may be changed before the plot is generated by modifying the default values, or later with the *Modify Plot* command. Other plot options, such as the scale type, the line type, and the plot symbol, may also be selected in this dialog, or later with the *Modify Plot* command.

The 'Format' option specifies that the numbers on the axis will be printed in either a fixed format or a scientific format, using the number of digits selected in the second box after the 'Format' label.

Data from multiple tables can be plotted simultaneously by selecting multiple items in the 'Plot data from' list and pressing 'OK'. You must be careful that the selected items all contain the same columns. Plotting isoproperty tables and saturation tables together must be done in two steps (once with New Plot, then with Overlay Plot).

Pressing the Save As Default button will add the current settings to the Defaults.Prf file. Subsequent uses of the program will then start with these defaults. The settings can also be saved by choosing *Save Current Options*.

Overlay Plot

This command works exactly like the *New Plot* command except that the additional data will be placed in an existing plot window. Overlaid plots use the existing scales for the ordinate and abscissa. However, if the overlaid plot has values that cause part or all of the plot to be off scale, a dialog box is presented that asks if you want to rescale the plot.

See the description of the *Plotting Data* for information on how to modify the appearance of an existing plot.

Plotting Data

There are a number of controls on the plot window:

Change Plot Size

Place the cursor over the bottom right corner of the plot window; the cursor should change from a '+' to a double-ended arrow. To resize the plot, hold down the left mouse button while dragging the lower right corner of the plot window to a different position (just as you would resize most windows on your computer).

Move Plot Labels

Plot labels can be placed on a plot window with the *Add Label* command. The text is initially placed in the center of the plot window, but it can be moved to any position. To move the text item, place the cursor over the text item. Hold the mouse down while dragging the text to its new location.

Cross Hairs

Hold the Shift key down to display cross hair lines on the plot. The coordinates of the cross hairs (in units corresponding to the plot axes) are displayed at the top of the plot.

Zoom

Click and drag the mouse to create a box. When the mouse button is released, the plot within the box will become the new plotting area. To zoom out, click 'Zoom Out' on the *Plot Menu*, or type 'Ctrl-O' (hold down the 'control' button, labeled 'Ctrl', and press the letter 'O'). To return to the original, un-zoomed plot, click 'Zoom Full Frame' on the *Plot Menu*, or press 'Ctrl-F' (hold down the 'control' button, labeled 'Ctrl', and press the letter 'F').

Modify Plot Characteristics

Double click the mouse anywhere within the plot rectangle. The Modify Plot dialog appears, just as if the *Modify Plot* command were issued.

Add a Point to a Table

When the right mouse button is clicked within the plotting area, the spot will be marked with a circle, and the values of the properties at the selected state point will appear on the most recently used table (created from *Specified State Points*). If no table is active, the right mouse button marks the spot with a circle only. Points added to the plot in this fashion can then be connected with lines or deleted by entering the *Modify Plot* option.

Zoom In, Out, Full Frame

There are several ways to zoom in or out of a plot. The easiest way is to click the mouse on the plot and, while holding down the mouse key, drag to a new location. The plot will zoom to the locations where the mouse was pressed and then released. Dragging outside the plot window will increase the plot size to the point where you released the mouse.

You can also zoom in or out by using the commands on the menu bar. If the Zoom Full Frame option is used, the plot will return to either its original size, or the size that was selected from the last use of the *Modify Plot* command.

1.2.7 Substance

The substance menu holds commands for selecting and defining a fluid. The current fluid is identified in the *Status Line* at the bottom of the screen. The menu options are:

Define New Mixture

The Define Mixture dialog provides a means for defining a fluid system consisting of 2 to 20 components. Select the mixture constituents from the list on the left by clicking on the fluid name, and then on the Add button (or simply double click). That fluid then moves from the list on the left to the list on the right. The fluids in the right list become part of the selected fluid system. To remove a fluid from the right list, click on the fluid name and then click the Remove button (or simply double click).

In addition, multiple selections can be made by either holding down the Ctrl key while selecting multiple fluids, or by holding down the Shift key to mark all fluids between the currently marked fluid, and the selected one. You can also use the arrow keys to select a fluid, and press Alt-A to add it. The period or '>' keys will also add a fluid. Removing the fluid can be done with either Alt-R, a comma, or '<'. Fluids can be highlighted by typing in the first few letters of their names; for example, pressing 'pr' will select propane; to get propylene, type 'propy'.

These alternate ways of adding fluids make it easier to find and select a fluid. For example, defining a mixture of R32, R125, R134a, and Propane is quickest by typing the following sequence: 'r3.r125.r134.pr.' If a wrong letter or number is entered, use the backspace, thus if 'r124' is entered in this example, press the backspace and then a '5', and the program will jump to R125.

Clicking the Info button displays key fixed point parameters for the selected pure fluid. More information in given in the *Fluid Information* dialog.

The fluid boxes use the same sorting techniques that were set up in the *Pure Fluid* dialog and the fluid set defined in the *Specify Fluid Set* dialog.

After the OK button is selected, the *Specify Composition* window appears, in which the mass or mole fractions of the components must be specified. In addition, the mixture can be given a name and optionally stored for later use.

Fluid Information

For a pure fluid, this window shows reference information, including key fixed point parameters (e.g., the critical temperature), for the selected pure fluid. Information on the model, its limits of applicability, and reference citations are provided separately for the equation of state, viscosity equation, thermal conductivity equation, surface tension equation, and melting and sublimation lines by clicking on the tab near the bottom.

The literature citations describing the selected model and the uncertainty of the equation are provided in the box at the bottom. Some or all of this information can be copied to the Clipboard or sent to the printer by selecting the text and pressing one of the Copy or Print buttons. Copied information can be pasted into other applications.

If more than one model is available for any of these properties, the list box directly below the box containing information on the range of applicability of the model can be used to select alternative models. Click on the arrow to the

right of the box to display the available models. The recommended model is identified with the term 'NIST Rec'. An alternate model may be selected for subsequent calculations by clicking on its name.

When a mixture has been selected, the fluid information screen will display a window containing the mixture name, molar mass (molecular weight), critical properties, and mixture composition. The mixture composition can be viewed using either mole fractions or mass fractions. Clicking on a mixture constituent name will bring up the fluid information screen described above for that constituent. If a grid has been previously opened using Calculate/Specified State Points, the Add to Table button can be used to copy the critical point and maximum temperature and pressure values to the table.

Fluid Search

The Fluid Search dialog can be used to find fluids that fit within a range of values. Either all of the fluids in the database can be searched, or only those in the *Specify Fluid Set*. To search for fluids within a specified property range, click on the check box for that property and enter the ranges desired in the unit system displayed. Multiple properties can be checked simultaneously. The properties that can be searched include the critical temperature, pressure, and density, the normal boiling point, the molar mass (molecular weight), the triple point temperature, and the liquid density at the normal boiling point.

Predefined Mixture

The Predefined Mixture dialog will display a list of the currently defined mixtures available. A number of predefined mixtures are provided with the program. These include the refrigerant blends defined in ASHRAE Standard 34, air, and several standard natural gas blends.

You can specify new mixtures consisting of up to twenty pure components using the *Define New Mixture* command. Clicking the Info button will bring up the *Fluid Information* window in which the critical point information, components, and composition of the mixture can be viewed.

Pseudo Pure Fluid

The Pseudo Pure Fluid dialog lists all of the mixtures that are available as pseudo pure fluids. Select the fluid you want to use by clicking on the fluid name. The properties that are calculated for pseudo pure fluids are based on equations of state that have been developed solely for a specific composition. Click the Info button to display the *Fluid Information* dialog, which provides critical constants and other property information for the selected fluid.

These calculations are generally based on mixture models that were fit to a wide variety of properties at various compositions. The uncertainties of the pseudo pure fluid equations are often higher than the corresponding mixture model. However, the calculational speed will be much faster, especially for saturation properties.

Pure Fluid

The Pure Fluid dialog lists the pure fluids available. By default, only the more commonly used fluids are displayed; use the *Specify Fluid Set* command to modify the displayed list of fluids. Select the fluid you want to use by clicking on the fluid name. Click the Info button to display the *Fluid Information* dialog, which provides critical constants and other property information for the selected fluid.

If a *Specify Fluid Set* has been specified, then the 'All fluids' button will display all of the fluids available, not just those in the specified set. The pure fluids can be sorted by short name (or nickname), full name, CAS number, full chemical formula, short chemical formula, synonym, or UN number. This is useful for fluids such as acetone, which is often called propanone, where the name known to the user may not be listed in one of the categories.

Specify Composition

The Specify Composition dialog serves two purposes. First, it provides a means of specifying the composition of a mixture. The composition can be specified in terms of either mass or mole fractions. The choice is made by clicking in the box containing the words 'Mass fraction' or 'Mole fraction' followed by clicking on the name of the desired base. If you click the mouse on the name of a pure component, the *Fluid Information* dialog appears, providing property information for that fluid.

The mass or mole fractions must be entered for each component and they must sum to 1 (or 100 if percentages are used) if the 'Normalize compositions to one' checkbox is not selected. If the 'Normalize compositions to one' checkbox is selected, the value of each mass or mole fraction is set to the entered value divided by the sum of the values. This option is most convenient if you want to specify the composition of a mixture by specifying the mass of each component.

The second function served by this dialog provides an opportunity to store the fluid system so that it can later be selected with the *Predefined Mixture* command. The name appearing in the Predefined Mixture window and the *Status Line* at the bottom of the screen is entered in the 'Mixture name' field. Click the Store button to save this information.

The Add or Remove Fluid buttons allow you to enter or delete fluids in the list without having to start back at the *Define New Mixture* dialog. The Add Fluid button will bring up the pure fluid list and the desired component should be selected. To remove a fluid, put the cursor in the composition box for that fluid and press the Remove Fluid button. You will be prompted for confirmation before the fluid is removed.

The Copy button places the fluid names and compositions (separated by tabs) onto the clipboard so that they can be pasted into other applications. The Paste button will paste the values that are on the clipboard into the composition boxes. The paste function is smart enough to search through multiple columns in the information stored on the clipboard for a column that contains numbers that sum to 1. For example, the copy button pastes both the fluid names and compositions to the clipboard. If the paste button is then pressed, the program realizes that the first column does not contain compositions and will paste the second column into the composition boxes.

Specify Fluid Set

With the increase in the number of fluids contained in REFPROP, it is often desirable to display only selected fluids of interest to the user. This is done through the Specify Fluid Set dialog. Six default sets are available, and can be changed by the user to match their own preferences. To use one of the six sets, simply click on it and press OK.

To enter a new set, press the Add New Set button and enter a descriptive name. The Rename button allows you to rename a file set, and the Delete Set button allows you to delete the set. To specify which fluids are in the set, press the Edit Set button and enter/delete fluids in a similar manner as is done in the *Define New Mixture* dialog.

The set labeled 'calibration fluids' contains those fluids that generally have both high accuracy equations of state and transport equations. The user should consult the *Fluid Information* screen to determine what the actual uncertainties in the equations are.

The sets are saved to the hard drive once you press OK. There is no need to save the options to keep the changes just made.

View Mixing Parameters

The database applies mixing rules to the pure-fluid Helmholtz energies. In some cases, it may be possible to improve the agreement between the calculated properties and experimental measurements by altering these rules. However, great care must be used in altering these rules, because they can greatly change the calculated properties. This command is not intended for the casual user.

The mixing rules are applied to binary pairs of mixture constituents. It may be necessary to select the pair you wish to view or alter. The fluids in the currently defined mixture are listed at the left. Only one pair at a time can be selected.

To change the fluids, simply select a new fluid, and the fluid selected first will be unmarked so that only two will be checked.

Shown at the right is the currently defined mixing rule for the selected binary pair and the associated parameter values. Different rules can be chosen by clicking on the down arrow button at the right of the rule box. The parameter values can also be changed. No error checking is provided on these values, so take care when changing the mixing rules. Also, note that changing the mixing parameters can change the critical parameters associated with the mixture.

Clicking the Apply button accepts the mixing rules and values as displayed, replacing the stored values. The binary pair can then be changed so that additional mixing rules can be entered. The OK button accepts the 'Applied' mixing rules, and closes the dialog. The Reset button restores the mixing rules for each binary pair to their original (as shipped) state. Note that the 'Apply' button must be clicked before viewing the next binary pair, or any changes will not be saved.

At the bottom of the dialog, documentation is provided briefly describing the mixing rule. You can select part or all of the text in this box. The Copy button copies the selected text to the Clipboard that can then be pasted into another application.

1.2.8 Window Menu

The Window menu contains the following four menu items

Cascade

The Cascade command will arrange all of the currently defined windows so that their title bars are visible, making it easy to select any of the windows.

Close Window

The active window (data or plot) will be closed when this option is selected. The window will no longer be available once closed.

Rename Window

Each table and plot window is provided with a default title when it is generated. You can change the default window title by entering an alternative in the edit space provided. This title is used in the *Save Tables and Print Tables*, *Print Setup and Dialog*, *New Plot*, and *Overlay Plot* dialogs for identification.

Tile

The Tile command resizes and positions the existing windows so that all are displayed. Scroll bars are provided, if needed.

1.3 DLLs

Information about the functions exposed in the DLL can be found here: *REFPROP DLL documentation*

More information about REFPROP can be found on github, in particular in the following repositories:

- Build the DLL yourself : For users that need to compile the DLL (shared library) themselves, a CMake build system is available

- Load multiple copies of the DLL in memory : You can load multiple copies of the REFPROP dll into memory with the code at https://github.com/usnistgov/REFPROP-manager
- Call the REFPROP library (wrappers) : Want to call REFPROP from Python, Excel, C#, Labview, etc.? There are wrappers available at https://github.com/usnistgov/REFPROP-wrappers
- Test the REFPROP library : There are C++-based tests of the REFPROP DLL available at https://github.com/usnistgov/REFPROP-tests . They run on all major platforms

CHAPTER 2

REFPROP DLL documentation

2.1 High-Level API

2.1.1 Function Listing

- *ABFLSHdll()*
- *ALLPROPS0dll()*
- *ALLPROPS1dll()*
- *ALLPROPSdll()*
- *ERRMSGdll()*
- *FLAGSdll()*
- *GETENUMdll()*
- *REFPROP1dll()*
- *REFPROP2dll()*
- *REFPROPdll()*
- *SETFLUIDSdll()*
- *SETMIXTUREdll()*
- *SETPATHdll()*

2.1.2 Function Documentation

subroutine ABFLSHdll (*ab*, *a*, *b*, *z*, *iFlag*, *T*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *ab_length*, *herr_length*)

General flash calculation that handles all inputs of T, P, D, h, e, s, and q.

Includes both blind flash calculations, and situations where the phase is known to be liquid, vapor, or 2-phase, and thus the calculation time will be much faster.

Many of the 2-phase flash routines can accept initial estimates to decrease calculation time and improve convergence. ABFLSH does not accept these, and ABFL2 or other routines will need to be called to use the initial estimates. These routines end in the letters FL2.

Notes:

- Cp and w are not defined for 2-phase states; the flag -9999980 is returned.
- Cv for 2-phase states is not calculated (use CV2PK); the flag -9999990 is returned.

Information on ab

Valid character codes for ab are:

- T - Temperature [K]
- P - Pressure [kPa]
- D - Density [mol/L or kg/m^3]
- E - Internal energy [J/mol or kJ/kg]
- H - Enthalpy [J/mol or kJ/kg]
- S - Entropy [J/mol-K or kJ/kg]
- Q - Quality [mol/mol or kJ/kg]

- For example, 'PH' indicates pressure and enthalpy inputs.
- For saturation properties, use codes of 'TQ' or 'PQ' for ab, and send b=1.
- The order of the letters does not matter, for example 'DH' = 'HD' for saturated vapor values and b=0 for saturated liquid values.

Information on iFlags

Three flags are currently allowed, and are sent combined in a three digit integer value. The digit on the right is the mass flag (iMass) defined below, the middle digit is the phase flag (kph), and the digit on the left specifies other flags (k).

- **iMass: Molar or mass flag**
  - 0 - All inputs and outputs are given on a mole basis.
  - 1 - All inputs and outputs are given on a mass basis.
  - 2 - All inputs and outputs are given on a mass basis except composition, which is given on a mole basis.
- **kph: Phase flag (except for inputs of Q)**
  - 0 - Unknown phase, the saturation routines will be called to determine the phase, which adds a substantial amount of time needed to calculate the properties.
  - 1 - State point is in the liquid phase, do not call saturation routine to determine state.
  - 2 - State point is in the vapor phase, do not call saturation routine to determine state.
  - 3 - State point is in the two-phase region.
- **kr,kq: Other flags for inputs of quality and either temperature or pressure (kq flag)**
  - 0 - Default

  - 1 - Quality on a molar basis (moles vapor/total moles) (default, the value of 1 is not necessarily needed).
  - 2 - Quality on a mass basis (mass vapor/total mass); for inputs of T and either h or e (kr flag)
  - 3 - Return lower density root.
  - 4 - Return higher density root.

Examples:

- 000 - Default - Phase of state is unknown, molar units will be used everywhere, higher density root will be returned.
- 002 - Use mass based properties for everything except composition.
- 011 - State is in the liquid, properties are mass based.
- 300 - Return the lower density root for TH or TE inputs.
- 200 - All inputs are on a mole basis, but quality is sent on a mass basis.

Parameters

- **ab** *[char ,in]* :: Character string composed of two letters that indicate the input properties.
- **a** *[double ,in]* :: Value of the property identified by the first letter in ab
- **b** *[double ,in]* :: Value of the property identified by the second letter in ab
- **z** (20) *[double ,in]* :: Composition (array of mole fractions) For TQ and PQ inputs, send b=-99 for melting line states and b=-98 for sublimation line states.
- **iFlag** *[int ,in]* :: Multiple flags combined into one variable (see above)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L or kg/m^3]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L or kg/m^3]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L or kg/m^3]; if only one phase is present, Dl = Dv = D.
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole or mass fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole or mass fractions); if only one phase is present, x = y = z.
- **q** *[double ,out]* :: Vapor quality on a MOLAR basis (moles of vapor/total moles)
- **e** *[double ,out]* :: Overall internal energy [J/mol or kJ/kg]
- **h** *[double ,out]* :: Overall enthalpy [J/mol or kJ/kg]
- **s** *[double ,out]* :: Overall entropy [J/mol-K or kJ/kg-K]
- **Cv** *[double ,out]* :: Isochoric (constant D) heat capacity [J/mol-K or kJ/kg-K]
- **Cp** *[double ,out]* :: Isobaric (constant P) heat capacity [J/mol-K or kJ/kg-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)

- **ab_length** *[int]* :: length of variable `ab` (default: 2)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `q` flags

q > 0 and q < 1 indicates a 2-phase state

q < 0 Subcooled (compressed) liquid

q = 0 Saturated liquid

q = 1 Saturated vapor

q > 1 Superheated vapor

q = -998 Subcooled liquid, but quality not defined (usually P > Pc)

q = 998 Superheated vapor, but quality not defined (usually T > Tc)

q = 999 Supercritical state (T>Tc and P>Pc)

subroutine ALLPROPS0dll (*iIn*, *iOut*, *iFlag*, *T*, *D*, *z*, *Output*, *ierr*, *herr*, *herr_length*)

Calculate any single phase property defined in the iOut array and return the values in the Output array. This routine should NOT be called for two-phase states!

The output array is not reset so that several passes can be made to fill in holes left by the previous pass (such as entries at different T, D, or z). The caller can zero out this array if so desired.

This routine is designed with the "superuser" in mind. It removes all string comparisons to approach the speed that could be obtained by calling the dedicated functions (such as THERM), but making it easy by allowing all inputs to be calculated with one routine. Since the units are not returned here, look in the ALLPROPS documentation under the molar column.

These values of iOut are defined in the COMMONS.INC file and are obtained by a call to GETENUM, as such for the enthalpy:

```
call GETENUM (0,'H',iEnum,ierr,herr)
```

To obtain the pure fluid value for some of the inputs, add 10000*ic (where ic is the component number) to the value of the enumerated value. The properties that can be used for this are given the bottom of the comments section in the ALLPROPS routine.

Parameters

- **iIn** *[int ,in]* :: Number of properties to calculate.
- **iOut** (200) *[int ,in]* :: Array of enumerated values that identify the property to be calculated (see above)
- **iFlag** *[int ,in]* :: Not yet used.
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Overall composition (array of mole fractions)
- **Output** (200) *[double ,out]* :: Values of the calculated properties.
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine ALLPROPS1dll (*hOut*, *iUnits*, *T*, *D*, *z*, *c*, *ierr*, *herr*, *hOut_length*, *herr_length*)

Short version of subroutine ALLPROPS that eliminates the arrays but allows the calculation of only one property at a time. All inputs and outputs are described in the ALLPROPS routine.

Parameters

- **hOut** *[char ,in]* :: Input string of properties to calculate (of any length). Inputs can be separated by spaces, commas, semicolons, or bars, but should not be mixed. For example, a proper string would be hOut='T,P,D,H,E,S', whereas an improperly defined string would be hOut='T,P;D H|E,S'. Use of lower or upper case is not important. Some properties will return multiple values, for example, hOut='F,Fc,XMOLE' will return 12 properties for a four component system, these being F(1), F(2), F(3), F(4), Fc(1), Fc(2), etc. To retrieve the property of a single component, use, for example, hOut='XMOLE(2),XMOLE(3)'
- **iUnits** *[int ,in]* :: See subroutine REFPROP for a complete description of the iUnits input value. A negative value for iUnits indicates that the input temperature is given in K and density in mol/dm^3, (Refprop default units), otherwise T and D will be converted first to K and mol/dm^3. Do not use the negative value for the iUnits parameter everywhere, only in this one situation.
- **T** *[double ,in]* :: Temperature, with units based on the value of iUnits.
- **D** *[double ,in]* :: Density, with units based on the value of iUnits.
- **z** (20) *[double ,in]* :: Composition on a mole or mass basis (array of size ncmax=20)
- **c** *[double ,out]* :: Output value (array of size 200 dimensioned as double precision). The number -9999970 will be returned when errors occur or no input was requested.
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hOut_length** *[int]* :: length of variable `hOut` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine ALLPROPSdll (*hOut*, *iUnits*, *iMass*, *iFlag*, *T*, *D*, *z*, *Output*, *hUnits*, *iUCodeArray*, *ierr*, *herr*, *hOut_length*, *hUnitsArray_length*, *herr_length*)

Calculate the single-phase properties identified in the hOut string at the temperature, density, and composition sent to the routine. Return the properties in mass or molar units depending on iUnits.

Warning: Do NOT call this routine for two-phase states, otherwise it will return metastable states if near but inside the phase boundary, and complete nonsense at other conditions. The value of q that is returned from the flash routines will indicate a two phase state by returning a value between 0 and 1. In such a situation, properties can only be calculated for the saturated liquid and vapor states. For example, when calling PHFLSH:

```
call PHFLSH (P,h,z,T,D,Dl,Dv,x,y,q,e,s,Cv,Cp,w,ierr,herr)
```

If q>0 and q<1, then values of the liquid and vapor compositions will be returned in the x and y arrays, and the properties of the liquid and vapor states can be calculated, for example:

```
call ENTRO (T,Dl,x,sliq)
call ENTRO (T,Dv,x,svap)
```

ALLPROPS was the name of a program developed at the University of Idaho under the direction of R.B. Stewart and R.T Jacobsen at the Center for Applied Thermodynamic Studies (CATS), with S.G. Penoncello and S.W. Beyerlein as professors at this institution. The software was distributed for about 10 years until around 2000 when it was officially replaced by the Refprop program. Some of the techniques from ALLPROPS was used

in the development of Version 6 of Refprop, and were in some ways its forerunner. The original code was DOS based and distributed on 3 1/2" floppy disks by regular mail. A Visual Basic version of ALLPROPS was developed in about 1995, and, although rarely distributed, inspired the graphical interface included with Version 7.0 and above of Refprop.

The ALLPROPS code was used to develop equations of state at the University of Idaho, and many of these are still in use today, such as ethanol, neon, R-11, R-12, R-22, R-23, R-143a, and the mixture air, along with the architecture behind the GERG-2008 mixture model. The equations of state for ethylene, nitrogen, and oxygen were developed in conjunction with the Ruhr University in Bochum, Germany, including a six month stay by R. Span from Bochum with E.W. Lemmon at Idaho while both worked on their upper degrees. The underlying code in the fitting program developed at CATS is still in use today, and has been used in nearly all equations of state developed over the last 20 years.

The name ALLPROPS was revived at NIST in 2017 in memory of the old but not forgotten program whose roots still form the foundation of much that goes on behind the scenes in the development of equations of state and property software.

Calling from the DLL

Two routines are available in the DLL, these are ALLPROPSdll and ALLPRP200dll. Both compress the `hUnitsArray` array so that it can be passed back as a single string. The segments are divided by the character 'l'. Both routines use the same list of arguments:

```
(hOut,iUnits,iMass,iFlag,T,D,z,Output,hUnits,iUCode,ierr,herr)
```

In ALLPROPSdll, the hOut string is 255 characters long, the hUnits string is 1000 characters long, and the Output and iUCode arrays each have a length of 20. In ALLPRP200dll, the hOut and hUnits strings are 10000 characters long, and the Output and iUCode arrays each have a length of 200.

Below are the labels that can be sent in the hOut string and a very short description of the property and units based on either a SI molar system (iUnits=1) or SI mass system (iUnits=2, or 3 with temperature in C).

Note about criticals: The items TC,PC,DC will return the critical point of a pure fluid, or, when SATSPLN has been called, the critical point of the mixture (or a very close approximation). When the splines have not been set up, the values are the same as TCEST below. For the critical points of the pure fluids in a mixture, use TCRIT, etc., explained below, which is useful when multiple fluids have been loaded. Parameters in the HMX.BNC file, which, for a binary mixture, are close for Type I mixtures, but for a multi-component or non-Type I mixture, can be significantly wrong.

Label	Description	SI Molar Units	SI Mass Units
Regular properties			
T	Temperature	[K]	[K]
P	Pressure	[kPa]	[kPa]
D	Density	[mol/dm^3]	[kg/m^3]
V	Volume	[dm^3/mol]	[m^3/kg]
E	Internal energy	[J/mol]	[kJ/kg]
H	Enthalpy	[J/mol]	[kJ/kg]
S	Entropy	[J/(mol*K)]	[(kJ/kg)/K]
CV	Isochoric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CP	Isobaric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CP/CV	Heat capacity ratio	[-]	[-]
W	Speed of sound	[m/s]	[m/s]
Z	Compressibility factor	[-]	[-]
JT	Isenthalpic Joule-Thomson coefficient	[K/kPa]	[K/kPa]
A	Helmholtz energy	[J/mol]	[kJ/kg]

Continued on next page

Table 1 – continued from previous page

G	Gibbs energy	[J/mol]	[kJ/kg]
R	Gas constant	[J/(mol*K)]	[(kJ/kg)/K]
M	Molar mass (or of the mixture)	[g/mol]	[g/mol]
QMASS	Quality (not implemented, q not known)	N.A.	[kg/kg]
QMOLE	Quality (not implemented, q not known)	[mol/mol]	N.A.
Not so regular properties			
KAPPA	Isothermal compressibility	[1/kPa]	[1/kPa]
BETA	Volume expansivity	[1/K]	[1/K]
ISENK	Isentropic expansion coefficient	[-]	[-]
KT	Isothermal expansion coefficient	[-]	[-]
BETAS	Adiabatic compressibility	[1/kPa]	[1/kPa]
BS	Adiabatic bulk modulus	[kPa]	[kPa]
KKT	Isothermal bulk modulus	[kPa]	[kPa]
THROTT	Isothermal throttling coefficient	[dm^3/mol]	[m^3/kg]
Derivatives			
DPDD	dP/dD at constant T	[kPa/(dm^3/mol)]	[kPa/(m^3/kg)]
DPDT	dP/dT at constant D	[kPa/K]	[kPa/K]
DDDP	dD/dP at constant T	[(mol/dm^3)/kPa]	[(kg/m^3)/kPa]
DDDT	dD/dT at constant P	[(mol/dm^3)/K]	[(kg/m^3)/K]
DTDP	dT/dP at constant D	[K/kPa]	[K/kPa]
DTDD	dT/dD at constant P	[(dm^3/mol)*K]	[(m^3/kg)*K]
D2PDD2	d^2P/dD^2 at constant T	[kPa/(dm^3/mol)^2]	[kPa/(m^3/kg)^2]
D2PDT2	d^2P/dT^2 at constant D	[kPa/K^2]	[kPa/K^2]
D2PDTD	d^2P/dTdD	[kPa/(dm^3/mol)/K]	[kPa/(m^3/kg)/K]
D2DDP2	d^2D/dP^2 at constant T	[(mol/dm^3)/kPa^2]	[(kg/m^3)/kPa^2]
D2DDT2	d^2D/dT^2 at constant P	[(mol/dm^3)/K^2]	[(kg/m^3)/K^2]
D2DDPT	d^2D/dPdT	[(mol/dm^3)/(kPa*K)]	[(kg/m^3)/[kPa*K]]
D2TDP2	d^2T/dP^2 at constant D	[K/kPa^2]	[K/kPa^2]
D2TDD2	d^2T/dD^2 at constant P	[(dm^3/mol)^2*K]	[(m^3/kg)^2*K]
D2TDPD	d^2T/dPdD	[K/(dm^3/mol)/kPa]	[K/(m^3/kg)/kPa]
Enthalpy derivatives			
DHDT_D	dH/dT at constant D	[(J/mol)/K]	[(kJ/kg)/K]
DHDT_P	dH/dT at constant P	[(J/mol)/K]	[(kJ/kg)/K]
DHDD_P	dH/dD at constant P	[(J/mol)*(dm^3/mol)]	[(kJ/kg)*(m^3/kg)]
DHDD_T	dH/dD at constant T	[(J/mol)*(dm^3/mol)]	[(kJ/kg)*(m^3/kg)]
DHDP_T	dH/dP at constant T	[(J/mol)/kPa]	[(kJ/kg)/kPa]
DHDP_D	dH/dP at constant D	[(J/mol)/kPa]	[(kJ/kg)/kPa]
Entropy derivatives			
DSDT_D	dS/dT at constant D	[(J/mol)/K^2]	[(kJ/kg)/K^2]
DSDT_P	dS/dT at constant P	[(J/mol)/K^2]	[(kJ/kg)/K^2]
DSDD_T	dS/dD at constant T	[(J/mol)*(dm^3/mol)/K]	[(kJ/kg)*(m^3/kg)/K]
DSDD_P	dS/dD at constant P	[(J/mol)*(dm^3/mol)/K]	[(kJ/kg)*(m^3/kg)/K]
DSDP_T	dS/dP at constant T	[(J/mol)/(kPa*K)]	[(kJ/kg)/[kPa*K]]
DSDP_D	dS/dP at constant D	[(J/mol)/(kPa*K)]	[(kJ/kg)/[kPa*K]]
Virial Coefficients			
Bvir	Second virial coefficient	[dm^3/mol]	[m^3/kg]
Cvir	Third virial coefficient	[(dm^3/mol)^2]	[(m^3/kg)^2]
Dvir	Fourth virial coefficient	[(dm^3/mol)^3]	[(m^3/kg)^3]
Evir	Fifth virial coefficient	[(dm^3/mol)^4]	[(m^3/kg)^4]
dBvirdT	1st derivative of B with respect to T	[(dm^3/mol)/K]	[(m^3/kg)/K]

Continued on next page

Table 1 – continued from previous page

d2BvirdT2	2nd derivative of B with respect to T	[(dm^3/mol)/K^2]	[(m^3/kg)/K^2]
dCvirdT	1st derivative of C with respect to T	[(dm^3/mol)^2/K]	[(m^3/kg)^2/K]
d2CvirdT2	2nd derivative of C with respect to T	[(dm^3/mol)^2/K^2]	[(m^3/kg)^2/K^2]
dDvirdT	1st derivative of D with respect to T	[(dm^3/mol)^3/K]	[(m^3/kg)^3/K]
d2DvirdT2	2nd derivative of D with respect to T	[(dm^3/mol)^3/K^2]	[(m^3/kg)^3/K^2]
BA	Second acoustic virial coefficient	[dm^3/mol]	[m^3/kg]
CA	Third acoustic virial coefficient	[(dm^3/mol)^2]	[(m^3/kg)^2]
	EOS testing properties		
GRUN	Gruneisen parameter	[-]	[-]
PIP	Phase identification parameter	[-]	[-]
RIEM	Thermodyn. curvature (nm^3/molecule)		
(Z-1)/D	(Z-1) over the density	[dm^3/mol]	[m^3/kg]
(Z-1)/P	(Z-1) over the pressure	[1/kPa]	[1/kPa]
P*V	Pressure times volume	[(dm^3/mol)*kPa]	[(m^3/kg)*kPa]
S*D	Entropy times density	[J/(mol*K)*(mol/dm^3)]	[(kJ/kg)*(kg/m^3)/K]
N1/T	Negative reciprocal temperature	[1/K]	[1/K]
RD	Rectilinear diameter (Dl+Dv)/2	[mol/dm^3]	[kg/m^3]
	Properties from ancillary equations		
ANC-TP	Vapor pressure from ancillary given T	[kPa]	[kPa]
ANC-TDL	Sat. liquid dens. from ancillary given T	[mol/dm^3]	[kg/m^3]
ANC-TDV	Sat. vapor dens. from ancillary given T	[mol/dm^3]	[kg/m^3]
ANC-PT	Vapor temp. from ancillary given P	[K]	[K]
ANC-DT	Vapor temp. from ancillary given D	[K]	[K]
MELT-TP	Melting pressure given T	[kPa]	[kPa]
MELT-PT	Melting temperature given P	[K]	[K]
SUBL-TP	Sublimation pressure given T	[kPa]	[kPa]
SUBL-PT	Sublimation temperature given P	[K]	[K]
	Less common saturation properties		
CSAT	Saturated heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CV2P	Isochoric two-phase heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
DPDTSAT	dP/dT along the saturation line	[kPa/K]	[kPa/K]
DHDZSAT	dH/dZ along the sat. line (Waring)	[J/mol]	[kJ/kg]
LIQSPNDL	Density at the liquid spinodal	[mol/dm^3]	[kg/m^3]
VAPSPNDL	Density at the vapor spinodal	[mol/dm^3]	[kg/m^3]
	Excess properties		
VE	Excess volume	[dm^3/mol]	[m^3/kg]
EE	Excess energy	[J/mol]	[kJ/kg]
HE	Excess enthalpy	[J/mol]	[kJ/kg]
SE	Excess entropy	[J/(mol*K)]	[(kJ/kg)/K]
AE	Excess Helmholtz energy	[J/mol]	[kJ/kg]
GE	Excess Gibbs energy	[J/mol]	[kJ/kg]
B12	B12	[dm^3/mol]	[m^3/kg]
	Ideal gas properties		
P0	Ideal gas pressure	[kPa]	[kPa]
E0	Ideal gas internal energy	[J/mol]	[kJ/kg]
H0	Ideal gas enthalpy	[J/mol]	[kJ/kg]
S0	Ideal gas entropy	[J/(mol*K)]	[(kJ/kg)/K]
CV0	Ideal gas isochoric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CP0	Ideal gas isobaric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CP0/CV0	Ideal gas heat capacity ratio	[-]	[-]

Continued on next page

Table 1 – continued from previous page

W0	Ideal gas speed of sound	[m/s]	[m/s]
A0	Ideal gas Helmholtz energy	[J/mol]	[kJ/kg]
G0	Ideal gas Gibbs energy	[J/mol]	[kJ/kg]
P-P0	Pressure minus ideal gas pressure	[kPa]	[kPa]
	Residual properties		
PR	Residual pressure (P-D*Rxgas*T)	[kPa]	[kPa]
ER	Residual internal energy	[J/mol]	[kJ/kg]
HR	Residual enthalpy	[J/mol]	[kJ/kg]
SR	Residual entropy	[J/(mol*K)]	[(kJ/kg)/K]
CVR	Residual isochoric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
CPR	Residual isobaric heat capacity	[J/(mol*K)]	[(kJ/kg)/K]
AR	Residual Helmholtz energy	[J/mol]	[kJ/kg]
GR	Residual Gibbs energy	[J/mol]	[kJ/kg]
	Ideal-gas contributions to the Helmholtz energy		
PHIG00	Red. IG Helmholtz energy A0/RT	[-]	[-]
PHIG10	tau*[d(A0/RT)/d(tau)]	[-]	[-]
PHIG20	tau^2*[d^2(A0/RT)/d(tau)^2]	[-]	[-]
PHIG30	tau^3*[d^3(A0/RT)/d(tau)^3]	[-]	[-]
PHIG01	del*[d(A0/RT)/d(del)]	[-]	[-]
PHIG02	del^2*[d^2(A0/RT)/d(del)^2]	[-]	[-]
PHIG03	del^3*[d^3(A0/RT)/d(del)^3]	[-]	[-]
PHIG11	tau*del*[d^2(A0/RT)/d(tau)d(del)]	[-]	[-]
PHIG12	tau*del^2*[d^3(A0/RT)/d(tau)d(del)^2]	[-]	[-]
PHIG21	tau^2*del*[d^3(A0/RT)/d(tau)^2d(del)]	[-]	[-]
	Residual contributions to the Helmholtz energy		
PHIR00	Red. resid. Helmholtz energy Ar/RT	[-]	[-]
PHIR10	tau*[d(Ar/RT)/d(tau)]	[-]	[-]
PHIR20	tau^2*[d^2(Ar/RT)/d(tau)^2]	[-]	[-]
PHIR30	tau^3*[d^3(Ar/RT)/d(tau)^3]	[-]	[-]
PHIR01	del*[d(Ar/RT)/d(del)]	[-]	[-]
PHIR02	del^2*[d^2(Ar/RT)/d(del)^2]	[-]	[-]
PHIR03	del^3*[d^3(Ar/RT)/d(del)^3]	[-]	[-]
PHIR11	tau*del*[d^2(Ar/RT)/d(tau)d(del)]	[-]	[-]
PHIR12	tau*del^2*[d^3(Ar/RT)/d(tau)d(del)^2]	[-]	[-]
PHIR21	tau^2*del*[d^3(Ar/RT)/d(tau)^2d(del)]	[-]	[-]
	Critical point and P,T maximums along isopleth (see above)		
TC	Critical temperature of a pure fluid	[K]	[K]
PC	Critical pressure of a pure fluid	[kPa]	[kPa]
DC	Critical density of a pure fluid	[mol/dm^3]	[kg/m^3]
TCEST	Estimated critical temperature	[K]	[K]
PCEST	Estimated critical temperature	[kPa]	[kPa]
DCEST	Estimated critical density	[mol/dm^3]	[kg/m^3]
TMAXT	Temperature at cricondentherm	[K]	[K]
PMAXT	Pressure at cricondentherm	[kPa]	[kPa]
DMAXT	Density at cricondentherm	[mol/dm^3]	[kg/m^3]
TMAXP	Temperature at cricondenbar	[K]	[K]
PMAXP	Pressure at cricondenbar	[kPa]	[kPa]
DMAXP	Density at cricondenbar	[mol/dm^3]	[kg/m^3]
	Reducing parameters		
TRED	Reducing temperature	[K]	[K]

Continued on next page

Table 1 – continued from previous page

DRED	Reducing density	[mol/dm^3]	[kg/m^3]
TAU	Tc/T (or Tred/T)	[-]	[-]
DEL	D/Dc (or D/Dred)	[-]	[-]
	Limits		
TMIN	Minimum temperature of the EOS	[K]	[K]
TMAX	Maximum temperature of the EOS	[K]	[K]
DMAX	Maximum density of the EOS	[mol/dm^3]	[kg/m^3]
PMAX	Maximum pressure of the EOS	[kPa]	[kPa]
	Transport, etc.		
VIS	Viscosity	[uPa*s]	[uPa*s]
TCX	Thermal conductivity	[W/(m*K)]	[W/(m*K)]
PRANDTL	Prandlt number	[-]	[-]
TD	Thermal diffusivity	[cm^2/s]	[cm^2/s]
KV	Kinematic Viscosity	[cm^2/s]	[cm^2/s]
STN	Surface tension	[mN/m]	[mN/m]
DE	Dielectric constant	[-]	[-]
	Heating values		
SPHT	Specific heat input	[J/mol]	[kJ/kg]
HFRM	Heat of formation	[J/mol]	[kJ/kg]
HG	Gross (or superior) heating value	[J/mol]	[kJ/kg]
HN	Net (or inferior) heating value	[J/mol]	[kJ/kg]
HGLQ	Gross Heat. Val. (Liquid)	[J/mol]	[kJ/kg]
HNLQ	Net Heat. Val. (Liquid)	[J/mol]	[kJ/kg]
HGVOL	Gross HV (Ideal gas volume basis)	[MJ/m^3]	[MJ/m^3]
HNVOL	Net HV (Ideal gas volume basis)	[MJ/m^3]	[MJ/m^3]
HEATVAPZ	Heat of vaporization (for pure fluids)	[J/mol]	[kJ/kg]
HEATVAPZ_T	. . . at constant temperature (for mixtures)	[J/mol]	[kJ/kg]
HEATVAPZ_P	. . . at constant pressure (for mixtures)	[J/mol]	[kJ/kg]
HEATVALUE	Heating value (mass or molar basis)	[J/mol]	[kJ/kg]
	Other properties		
PINT	Internal pressure	[kPa]	[kPa]
PREP	Repulsive part of pressure	[kPa]	[kPa]
PATT	Attractive part of pressure	[kPa]	[kPa]
EXERGY	Flow Exergy	[J/mol]	[kJ/kg]
CEXERGY	Closed System Exergy	[J/mol]	[kJ/kg]
CSTAR	Critical flow factor	[-]	[-]
TMF	Throat mass flux	[kg/(m^2*s)]	[kg/(m^2*s)]
FPV	Supercompressibility	[-]	[-]
SUMFACT	Summation Factor	[-]	[-]
RDAIR	Relative Density in air (specific gravity)	[-]	[-]
RDH2O	Relative Density in water (specific gravity)	[-]	[-]
API	API Gravity	[-]	[-]
	Fluid fixed points for mixtures		
	At the “true” critical point of the EOS dP/dD=0 and d^P/dD^2=0 at constant temperature		
TCRIT	Critical temperature of component i	[K]	[K]
PCRIT	Critical pressure of component i	[kPa]	[kPa]
DCRIT	Critical density of component i	[mol/dm^3]	[kg/m^3]
TCTRUE	True EOS critical temp. of component i	[K]	[K]
DCTRUE	True EOS critical density of component i	[mol/dm^3]	[kg/m^3]
TTRP	Triple point temperature of component i	[K]	[K]

Continued on next page

Table 1 – continued from previous page

PTRP	Triple point pressure of component i	[kPa]	[kPa]
DTRP	Triple point density of component i	[mol/dm^3]	[kg/m^3]
TNBP	Normal boiling point temp. of comp. i	[K]	[K]
REOS	Gas constant of component i for EOS	[J/(mol*K)]	[(kJ/kg)/K]
MM	Molar mass of component i	[g/mol]	[g/mol]
ACF	Acentric factor of component i	[-]	[-]
DIPOLE	Dipole moment of component i	[debye]	[debye]
TREF	Ref. state temperature of component i	[K]	[K]
DREF	Ref. state pressure of component i	[kPa]	[kPa]
HREF	Ref. state enthalpy of comp. i at T0 and P0	[J/mol]	[kJ/kg]
SREF	Ref. state entropy of comp. i at T0 and P0	[J/(mol*K)]	[(kJ/kg)/K]
	Transport properties as a function of component number		
	Viscosity=ETA0+ETAB2+ETAR+ETAC		
	Thermal conductivity=TCX0+TCXR+TCXC		
ETA0	Dilute gas viscosity of component i	[uPa*s]	[uPa*s]
ETAB2	2nd virial viscosity of component i	[uPa*s]	[uPa*s]
ETAR	Residual viscosity of component i	[uPa*s]	[uPa*s]
ETAC	Viscosity critical enhance. of comp. i	[uPa*s]	[uPa*s]
TCX0	Dilute gas thermal cond. of comp. i	[W/(m*K)]	[W/(m*K)]
TCXR	Residual (background) cond. of comp. i	[W/(m*K)]	[W/(m*K)]
TCXC	Cond. crit. enhancement of comp. i	[W/(m*K)]	[W/(m*K)]
	Mixture properties as a function of component number		
K	K value (y/x) (not implemented, y unknown)	[-]	[-]
F	Fugacities	[kPa]	[kPa]
FC	Fugacity coefficients	[-]	[-]
CPOT	Chemical potentials	[J/mol]	[kJ/kg]
DADN	n*partial(alphar)/partial(ni)	[-]	[-]
DNADN	partial(n*alphar)/partial(ni)	[-]	[-]
XMOLE	Composition on a mole basis	[-]	[-]
XMASS	Composition on a mass basis	[-]	[-]
FIJMIX	Binary parameters (see REFPROP)		

The dimension statements for these variables are (in Fortran):

```
parameter (ncmax=20)      ! Maximum number of components in the mixture
parameter (iPropMax=200)  ! Number of output properties available in ALLPROPS.
character*10000 hOut      ! hOut can actually be of any length.
character herr*255,hUnitsArray(iPropMax)*50
integer ierr,iUnits,iMass,iFlag,iUCodeArray(iPropMax) ! Note: as integer*4
double precision T,D,z(ncmax),Output(iPropMax)
```

Parameters

- **hOut** *[char ,in]* :: Input string of properties to calculate. Inputs can be separated by spaces, commas, semicolons, or bars, but should not be mixed. For example, a proper string would be hOut='T,P,D,H,E,S', whereas an improperly defined string would be hOut='T,P;D H|E,S'. Use of lower or upper case is not important. Some properties will return multiple values, for example, hOut='F,Fc,XMOLE' will return 12 properties for a four component system, these being F(1), F(2), F(3), F(4), Fc(1), Fc(2), etc. To retrieve the property of a single component, use, for example, hOut='XMOLE(2),XMOLE(3)'.
- **iUnits** *[int ,in]* :: See subroutine REFPROP for a complete description of the iUnits input value. A negative value for iUnits indicates that the input temperature is given in K and

density in mol/dm^3, (Refprop default units), otherwise T and D will be converted first to K and mol/dm^3. Do not use the negative value for the iUnits parameter everywhere, only in this one situation.

- **iMass** *[int ,in]* :: Specifies if the input composition is mole or mass based
- **iFlag** *[int ,in]* :: Turn on or off writing of labels and units to hUnitsArray (eventually may be multiple flags combined into one variable, similar to ABFLSH)
- **T** *[double ,in]* :: Temperature, with units based on the value of iUnits.
- **D** *[double ,in]* :: Density, with units based on the value of iUnits.
- **z** (20) *[double ,in]* :: Composition on a mole or mass basis (array of size ncmax=20)
- **Output** (200) *[double ,out]* :: Array of properties that were specified in the hOut string (array of size 200 dimensioned as double precision). The number -9999970 will be returned when errors occur or no input was requested.
- **hUnits** *[char ,out]* :: String with units. Will also contain error messages when necessary. The string will be empty if iFlag=0.
- **iUCodeArray** (200) *[int ,out]* :: Array (of size 200) with the values of iUCode(n) described in the REFPROP subroutine.
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hOut_length** *[int]* :: length of variable `hOut` (default: 10000)
- **hUnitsArray_length** *[int]* :: length of variable `hUnitsArray` (default: 10000)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `imass` flags

0 Input compositions given in mole fractions

1 Input compositions given in mass fractions

`iflag` flags

0 Do not write anything to the hUnitsArray array, thus increasing the calculation speed. (String handling in Fortran is very computationally expensive.)

1 Write labels and units to the hUnitsArray array.

2 Return only the string number described under "iUCodeArray" below and the units. (No properties will be calculated.)

-1 Write labels and units for only the first item.

`subroutine ERRMSGdll` (*ierr*, *herr*, *herr_length*)

Retrieve the last error message saved in calls to ERRNUM (but only if the ierr variable is not equal to zero). Write error messages to default output if iErrPrnt is active. The variable iErrPrnt in the common blocks must always be zero when compiling the DLL.

Outputs depend on variable iErrPrnt in the common blocks

- iErrPrnt= 0 - Error string not written (default)
- iErrPrnt=-1 - Error string written to screen
- iErrPrnt= 1 - Error string written to screen only if ierr is positive
- iErrPrnt=3,-3 - Same as 1 and -1, but program also pauses

Parameters

- **ierr** *[int ,in]* :: Error number from the last call to ERRNUM
- **herr** *[char ,out]* :: Associated error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine FLAGSdll (*hFlag*, *jFlag*, *kFlag*, *ierr*, *herr*, *hFlag_length*, *herr_length*)

Set flags for desired behavior from the program.

Table 2: Table of flags in FLAGS function

hFlag	jFlag
Return errors	• 0 - Return only final messages (default). • 1 - Return all intermediate messages. • 2 - Do not return messages. This flag is not reset with a new call to SETUP.
Write errors	• 0 - Error strings not written to screen (default). • -1 - Error string written to screen. • 1 - Error string written to screen only if ierr is positive. • 3,-3 - Same as 1 and -1, but program also pauses. This flag is not reset with a new call to SETUP.
Dir search	• 0 - Search for fluid files in alternate directories (as defined in OPENFL) (default). • 1 - Do not search in directories other than the one set by the call to SETPATH, except for a 'fluids' subdirectory within the folder given in SETPATH. If the fluid files for the reference fluid(s) are not in the SETPATH directory, then transport properties may not be calculated. • 2 - Make no additional checks if the fluid file is not found after the first attempt to open the file (for example, checking upper and lower case). This flag is never reset.
Cp0Ph0	• 1 - Change the ideal gas equation to Cp0. • 2 - Change the ideal gas equation to PH0. The default is set by the value in the fluid file. Calling SETUP resets the equation to its default state as given in the fluid file.

Continued on next page

Table 2 – continued from previous page

hFlag	jFlag
PX0	• 0 - Use the fluid file as is for the ideal gas equation (default). • 1 - Use the PX0 (or PH0 when no PX0 is available) for all calculations and turn off the call to SETREF. For mixtures, the reference state of "each pure component" will be used. This flag is never reset. When setting up the fluids through a call to the REFPROP subroutine, the SETREF flag described below will override this flag if turned on. Warning: Don't use the Cp0Ph0 flag to attempt to switch back to Cp0 (h and s will be wrong)
Skip SETREF	• 0 - Call the SETREF routine to setup the reference state (default). • 1 - Skip the call to SETREF. However, this means energy, enthalpy, and entropy will not be correct (but only by an offset to their usual values). This must be called before the call to SETUP, and is never reset.
Mixture reference or SETREF	• 0 - Do nothing (default). • 2 - When calling subroutine REFPROP, call SETREF first with a value of 2 for the second entry. (See subroutine SETREF for details.) This must be called before the call to REFPROP (the subroutine in PROP_SUB.FOR), and is never reset.
Skip ECS	• 0 - Load the ECS fluids required for transport properties (for pure fluids in slots 21-40, and mixtures in slot 41). • 1 - Don't load the ECS fluids, only the requested fluids (this may deactivate pure fluid transport properties, and will deactivate all mixture transport calculations. This must be called before the call to SETUP, and is never reset.
Splines off	• 1 - Turn the splines off (assuming that they were turned on initially by a call to SATSPLN). Calling SETUP again will also turn off the splines.

Continued on next page

Table 2 – continued from previous page

hFlag	jFlag
Ignore bounds or Bounds	• 0 - Check all errors and respond accordingly (default). • 1 - Ignore bounds for certain situations, such as calling SATT below the triple point or states above the melting line. This flag is never reset.
Cache	• 0 - Cache all calculated values (default). • 1 - Cache only low level calculations, such as derivatives calculated in PHIFEQ. • 2 - Cache only calculated properties in major subroutines such as SATT and SATP. • 3 - No caching. This flag is not reset with a new call to SETUP.
Reset all	• 2 - Call RESETA to reset all cached values. This includes all flags set by calls to this routine, except for the use of a pure fluid in a mixture or reducing nc. Subroutine RESETA is always called by SETUP, but does not reset many of the flags set by calls to this routine.
Reset HMX or HMX	• 1 - Reset the caching flag so that the HMX.BNC file is read again on the next call to SETUP. This option is only useful during fitting mixture models or modifying the HMX.BNC file to add new interaction parameters, otherwise this flag will only slow down the program by forcing a reread of the mixture file. The output variable kFlag will be 0 or 1 to indicate whether or not the HMX.BNC will be read on the next call to SETUP.
Pure fluid	• 0 - Use full mixture equation of state loaded (default). • <>0 - Use the pure fluid loaded in the slot specified by jFlag. This option is reset during the call to SETUP.
Component number or nc	• nc - Reduce the number of fluids being used to nc. See SETNC routine for details. The output in kFlag will give the number of fluids in use, which can be useful even if this option has not been called to set nc. This option is reset during the call to SETUP.

Continued on next page

Table 2 – continued from previous page

hFlag	jFlag
Peng-Robinson or PR	• 0 - Turn off the Peng-Robinson equation of state (default). • 2 - Use Peng-Robinson equation for all calculations. • 3 - Use Peng-Robinson with translation term deactivated. This option is never reset.
kij Zero	• 0 - Use the fitted kij values found in the HMX.BNC file on the lines with PR1 (default). • 1 - Set all kij values to those estimated in ESTPR (thus ignoring the ones on the PR1 lines in the HMX.BNC file). • 2 - Set all kij values to zero. This option is never reset.
AGA8	• 0 - Turn off AGA8 and return to the fluids loaded from the call to SETUP (default) • 1 - Turn on the use of the AGA8 DETAIL equation of state. If the AGA8 option is active, it overrides all other models. Unlike the GERG 2008 option, this model is active (or deactivated) immediately upon calling this routine. The AGA8 flag is never reset, thus recalling SETUP only changes the fluids, not the model.
GERG 2008 or GERG	• 0 - Set a flag to turn off GERG 2008 next time SETUP is called. • 1 - Turn on the flag that will cause the GERG 2008 equation to be loaded next time SETUP is called This option MUST be called before SETUP. When turning off the GERG, call the SETUP routine again after calling this routine. Because the GERG model is not activated until SETUP is called, the value of kflag will be 1 until the next call to setup, at which time it will be set to 2 to indicate that it is fully active. When turning off the GERG model, the value of kflag will be -1 until the next call to setup, and then it will be reset to zero. The -1 indicates that it is still in use but waiting to be reset. This flag is never reset.

Continued on next page

Table 2 – continued from previous page

hFlag	jFlag
Gas constant or R	• 0 - Default is to use the most current gas constant for all fluids except nitrogen, argon, oxygen, ethylene, CO2, methane, and ethane. • 1 - Use most current gas constant for all fluids (must be called after call to SETUP). • 2 - Use gas constant from fluid files for each equation of state (must be called after call to SETUP). This option is reset during the call to SETUP.
Calorie	• 0 - Use a calorie to joule conversion value of 4.184 cal/J (default). • 1 - Use the IT value of 4.1868 cal/J. This option is never reset.
Debug	• 0 - Turn off all debugging. • 1 - In the REFPROP subroutine, write all input variables to a file called input.dat, and all output values to a file called output.dat • 2 - In SETUP, write out the full path of the files that were either opened or tried to open. This option is never reset.

Parameters

- **hFlag** *[char ,in]* :: Indicator for the option to set (letters in the string are case insensitive).
- **jFlag** *[int ,in]* :: Flag to choose what to do in each option. Send -999 to just obtain the current value of the flag.
- **kFlag** *[int ,out]* :: Current setting of the flag for the option identified by iFlag. (Returned regardless of the value of jFlag.)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hFlag_length** *[int]* :: length of variable `hFlag` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine GETENUMdll (*iFlag*, *hEnum*, *iEnum*, *ierr*, *herr*, *hEnum_length*, *herr_length*)

Translate a string of letters into an integer value that can be used in calls to ALLPROPS0 to increase the speed of property calculations by eliminating string comparisons (which are time expensive in Fortran). This can be done once at the beginning of a program for all properties that will be used, and stored for later use as needed.

The input strings possible are described in subroutines ALLPROPS and GETUNIT.

Parameters

- **iFlag** *[int ,in]* :: Flag to specify which type of enumerated value to return
- **hEnum** *[char ,in]* :: The string that will be used to return the enumerated value. Only uppercase letters are allowed to decrease the time required to process the values.
- **iEnum** *[int ,out]* :: The enumerated value that matches the string sent in hEnum.

- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hEnum_length** *[int]* :: length of variable `hEnum` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iflag` flags

0 Check all strings possible.

1 Check strings for property units only (e.g., SI, English, etc.).

2 Check property strings and those in #3 only.

3 Check property strings only that are not functions of T and D (for example, the critical point, acentric factor, limits of the EOS, etc.).

subroutine REFPROP1dll (*hIn*, *hOut*, *iUnits*, *iMass*, *a*, *b*, *z*, *c*, *q*, *ierr*, *herr*, *hIn_length*, *hOut_length*, *herr_length*)

Short version of subroutine REFPROP that eliminates the arrays and the need to send the fluid names each time.

The variable Output (which is an array) is not included here, rather the variable c returns the calculated value as a double precision variable, and thus only one value can be returned at a time. If the error number (ierr) is zero, the string contained in hUnits will be sent back in herr.

If q (quality) returns a value between zero and one (and thus the state is two-phase), the REFPROP routine will be needed to obtain the equilibrium compositions.

Parameters

- **hIn** *[char ,in]* :: Input string of properties being sent to the routine.
- **hOut** *[char ,in]* :: Output string of properties to be calculated.
- **iUnits** *[int ,in]* :: The unit system to be used for the input and output properties (such as SI, English, etc.) See the details in the REFPROP subroutine for a complete description of the iUnits input value. **NOTE**: A mass based value for iUnits does not imply that the input and output compositions are on a mass basis, this is specified with the iMass variable.
- **iMass** *[int ,in]* :: Specifies if the input composition is mole or mass based
- **a** *[double ,in]* :: First input property as specified in the hIn variable.
- **b** *[double ,in]* :: Second input property as specified in the hIn variable.
- **z** (20) *[double ,in]* :: Composition on a mole or mass basis depending on the value sent in iMass (array of size ncmax=20).
- **c** *[double ,out]* :: Output value. The number -9999970 will be returned when errors occur, and the number -9999990 will be returned when nothing was calculated. Read the comments in the ALLPROPS routine for more information.
- **q** *[double ,out]* :: Vapor quality on a mole or mass basis depending on the value of iMass. (See subroutine ABFLSH for the definitions of values returned for this variable). To obtain the molar quality regardless of iMass, send "qmole" as an input in hIn, and vice-versa for "qmass".
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hIn_length** *[int]* :: length of variable `hIn` (default: 255)
- **hOut_length** *[int]* :: length of variable `hOut` (default: 255)

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `imass` flags

0 Input compositions given in mole fractions, quality on a molar basis.

1 Input compositions given in mass fractions, quality on a mass basis. For two-phase states, the values in x and y will be returned on a mass basis if iMass=1. **NOTE** If the fluid string sent to this routine contains the word "mass" at the end (and thus contains the composition as well as the names of the fluids), this will have preference over the value of iMass when converting those compositions from a mass to a molar basis. However, compositions sent back will still be based on the value in iMass.

subroutine REFPROP2dll (*hFld*, *hIn*, *hOut*, *iUnits*, *iFlag*, *a*, *b*, *z*, *Output*, *q*, *ierr*, *herr*, *hFld_length*, *hIn_length*, *hOut_length*, *herr_length*)

Short version of subroutine REFPROP that eliminates the less used variables such as the x and y composition arrays. If the error number (ierr) is zero, the string contained in hUnits will be sent back in herr.

If q (quality) returns a value between zero and one (and thus the state is two-phase), the REFPROP routine will be needed to obtain the equilibrium compositions.

See subroutine REFPROP for further information on the input and output variables below.

Parameters

- **hFld** *[char ,in]* :: Fluid string.
- **hIn** *[char ,in]* :: Input string of properties being sent to the routine.
- **hOut** *[char ,in]* :: Output string of properties to be calculated.
- **iUnits** *[int ,in]* :: The unit system to be used for the input and output properties (such as SI, English, etc.)
- **iFlag** *[int ,in]* :: Flag to specify if the routine SATSPLN should be called (where a value of 1 activates the call).
- **a** *[double ,in]* :: First input property as specified in the hIn variable.
- **b** *[double ,in]* :: Second input property as specified in the hIn variable.
- **z** (20) *[double ,in]* :: Molar composition (array of size ncmax=20).
- **Output** (200) *[double ,out]* :: Array of properties specified by the hOut string (array of size 200 dimensioned as double precision). The number -9999970 will be returned when errors occur, and the number -9999990 will be returned when nothing was calculated. Read the comments in the ALLPROPS routine to fully understand the contents of this array.
- **q** *[double ,out]* :: Vapor quality on a mole basis.
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hFld_length** *[int]* :: length of variable `hFld` (default: 10000)
- **hIn_length** *[int]* :: length of variable `hIn` (default: 255)
- **hOut_length** *[int]* :: length of variable `hOut` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine REFPROPdll (*hFld*, *hIn*, *hOut*, *iUnits*, *iMass*, *iFlag*, *a*, *b*, *z*, *Output*, *hUnits*, *iUCode*, *x*, *y*, *x3*, *q*, *ierr*, *herr*, *hFld_length*, *hIn_length*, *hOut_length*, *hUnits_length*, *herr_length*)

Calculate the properties identified in the hOut string for the inputs specified in the hIn string for the fluid or

mixture given in the hFld string. The unit identifier for the properties should be passed in the iUnits variable (as described below). Compositions can be sent as mole fractions or mass fractions in the z array depending on the value of iMass.

Several items must be considered before using this routine. The most important is the speed of calculations. The original fortran code that called dedicated functions such as TPRHO, TPFLSH, PHFLSH, and so on (mostly given in FLSH_SUB.FOR) and the non-iterative functions such as THERM and TRNPRP requires very few (or no) string comparisons and are quite fast. Multiple string comparisons are made to determine the inputs and outputs the user has selected. Due to the limitation of Fortran in string parsing, this will cause a dramatic increase in the time required to make the calculations, such as two to three times as long as the dedicated functions. Thus the ease of use of this REFPROP subroutine versus the speed of calculation from the older routines must be considered before developing any application.

Information on hFld

For a pure fluid, hfld contains the name of the fluid file (with a path if needed).

For a mixture, it contains the names of the constituents in the mixture separated by semicolons or asterisks. Once the routine has been called with hFld set to the desired fluids, a space can be sent for all other calls that use the same fluid(s). For a predefined mixture, the extension ".mix" must be included. If the composition is included in the hFld variable, or if a predefined mixture is selected, the composition will be returned in the z array (on a molar or mass basis depending on iMass). That composition (or other compositions) must be sent in z in all subsequent calls to this routine. See subroutines SETFLUIDS and SETMIXTURE further below for additional information and examples.

Note: The speed of the program will be increased (sometimes substantially) if you call this routine only once with the name of the fluid and then never again unless your fluid or mixture changes. If your composition changes, send the new composition in the z array rather then sending a new string in the hfld variable.

Examples:

```
hFld='NITROGEN'
hFld='C:/Program Files (x86)/REFPROP/FLUIDS/R1234YF.FLD'
hFld='CARBON DIOXIDE'
hFld='METHANE;ETHANE;PROPANE;BUTANE;ISOBUTANE'
hFld='METHANE*ETHANE*PROPANE*BUTANE*ISOBUTANE'
hFld='R134a;0.3; R1234yf;0.3; R1234ze(Z);0.4'
hFld='CO2;0.2 * isobutane;0.3 * propadiene;0.5'
hFld='Nitrogen;Oxygen;Argon|0.4;0.3;0.3'
hFld='Nitrogen; Oxygen; Argon   ~   0.4; 0.3; 0.3'
hFld='R410A.MIX'
```

Information on hIn

Valid codes are T, P, D, E, H, S, and Q (temperature, pressure, density, energy, enthalpy, entropy, and quality). Two of these should be sent together to identify the contents of the a and b variables. For example, 'TP' would indicate inputs of temperature and pressure, and 'TQ' would indicate inputs of temperature and quality. A value of 0 for the quality will return a saturated liquid state, and a value of 1 will return a saturated vapor state. A value between 0 and 1 will return a two-phase state. Valid inputs are: TP, TD, TE, TH, TS, TQ, PD, PE, PH, PS, PQ, DE, DH, DS, DQ, ES, EQ, HS, HQ, SQ (or the inverse of any of these, e.g., QT) (hIn is not case sensitive, e.g., 'TQ' = 'tq'). When q is >0 and <1, then the quality uses a molar basis when iMass=0, and a mass basis when iMass=1. The value of iUnits has no effect on the value of q (as either an input or output). The shortcuts Tsat and Psat can be used to specify a saturation state for the liquid for a pure fluid. To return, for example, the saturated vapor density, Dvap would be used as an output variable. The order of the properties being sent to the routine in the variables a and b has to be the same as the letters sent to hIn; for example, if hIn is 'QT', then a=q and b=T.

The ABFLSH routine is called to determine the phase of the inputs (liquid, vapor, or 2-phase), and then the appropriate iterative routine will be called to obtain the independent properties of the equations of state: tem-

perature and density. For subsequent calculations for properties that are in the single phase, use the code TD&, where the symbol & indicates the single phase state. The time required with the use of TD& is negligible compared to that required for the iterative solution called by ABFLSH. However, the properties sent to this routine and the calculated outputs are cached to avoid additional iterative calls when the solution has already been determined. Be sure to read the warnings at the top of the ALLPROPS routine for additional information.

Flags to specify certain phases are listed below, for example, 'TD>' would specify an input state in the liquid phase but which would normally be two-phase. Those available are:

- `**>` or `**L`: When the letter 'L' is attached after the two letters that specify the input properties (such as 'TP'), the routine will assume that the input properties are in the single phase liquid region, or are within the two-phase area as a metastable state. For example: TP>, PH>, HSL.
- `**<` or `**V`: The letter 'V' (or the sign '<') specifies that the input state for the properties listed in the first two letters is in the single phase vapor (including metastable states). For example: TP<, PH<, HSV.
- `TH<` or `TH>`: Inputs of temperature and enthalpy (or occasionally temperature and internal energy) generally have two valid states. To obtain the root with the higher pressure, use TH> or TE>, and for the lower pressure use TH< or TE<.
- `*MELT`: Return properties at the melting point where the input property is specified by the `*`, for example TMELT requires the temperature for the input variable a, PMELT requires the pressure for input variable a, and so on.
- `*SUBL`: Return properties at the sublimation point, as described above for the melting point.
- `CRIT`: Return properties at the critical point (for example, hIn='CRIT' and hOut='S' would return the entropy at Tc and Dc). For a mixture, the critical point defined by the equation of state is only available if the SATSPLN routine has been called, otherwise an estimated value is returned.
- `TRIP`: Return liquid phase properties at the triple point.
- `NBP`: Return properties at the normal boiling point.
- `DSAT`: Return the saturation properties for the input density.
- `HSAT`, `HSAT2`: Enthalpy can be doubled valued in the vapor phase for some fluids. In such a situation, HSAT2 will return the root with the lower temperature.
- `SSAT`, `SSAT2`, `SSAT3`: Entropy can be doubled or triple valued in the vapor phase for some fluids (see butane for example). In such a situation, SSAT will return the root at the highest temperature, SSAT2 will return the middle root, and SSAT3 will return the root with the lowest temperature.

Various flags are available that can be sent to this routine in the variable hIn to gain access to all other features of the Refprop program. These cannot be combined as multiple inputs in hIn:

- `FLAGS`: Call the FLAGS routine at the bottom of this file to initialize the options available for controlling certain aspects of the Refprop program. Some of these include caching properties, turning on/off different types of equations of state (Peng-Robinson, GERG-2008, and AGA-8), the calorie to Joule definition, and so on. The flag string (the first input to the FLAGS routine) should be sent in hOut and the flag option should be sent in iFlag. The output (3rd variable in the routine) is returned in iUCode. See the FLAGS routine for further information. The variable hFld for the REFPROP routine should be left blank when using this option.
- `EOSMIN`: Return the property specified in hOut at the minimum temperature allowed in the equation of state. This is generally at the triple point in the liquid phase. Note that an input of P or D will not return the obvious minimum (zero), but the pressure and density at the liquid phase triple point (or lower temperature limit for a mixture). For water, the triple point T is still returned, even though lower temperatures are possible.
- `EOSMAX`: Return the maximum temperature, pressure, or density (as specified in hOut) for the equation of state. The maximum density of the equation of state does not occur at the maximum pressure and

temperature. Only T, P, or D can be returned one at a time to emphasize that properties at Tmax and Pmax are not the same as at Tmax and Dmax.

- `SETREF`: Call the SETREF routine. The reference state (DEF, NBP, IIR, ASH, OTH, OT0, or NA) should be sent in hOut. For the OTH and OT0 options, the values of h0, s0, T0, and P0 should be included in the hOut variable, separated by semicolons. For example:

```
hOut='OTH;10.;1.;323.15;101.325'
```

  This would set the enthalpy to 10 J/mol and the entropy to 1 J/mol-K at 323.15 K and 101.325 kPa. In the GUI is an option for mixtures to set the reference state to either the composition in use or to each pure fluid. To set this option through the DLL, a value of either 1 or 2 should be sent to this routine in the variable a. This will set the variable labeled ixflag in subroutione SETREF in the SETUP.FOR file. All other options for this command are also explained in the SETUP.FOR file.

- `SETREFOFF`: Turn off the inputs that were sent in the option above.
- `PATH`: Call the SETPATH routine with the path given in hFld.
- `SATSPLN`: Call the SATSPLN routine for the input composition (as described in the SAT_SUB.FOR file). The fluid name or mixture names and the composition must be sent with this command or have already been setup before this is called. This command is identical to calling this routine with iFlag=1, except that it can be issued at any time.

Information on hOut

String output is returned in hUnits. Numerical output is returned in Output(1). For flags used to obtain a value of a particular fluid in a mixture, the component number should be added after the command, such as NAME(3) or FDIR(1). Only one string output can be requested at a time for the following flags, down to the line that says DLL#. Use the ALLPROPS routine to return multiple strings for all the components in the mixture. This is done without using the component number, e.g., sending "NAME" to that routine. For numerical values, multiple inputs can be requested here, and must be separated by spaces, commas, semicolons, or bars, but these separators should not be mixed. See subroutine ALLPROPS (which follows this routine) for further information.

- `ALTID`: Return the alternative fluid whose mixing rules are used when others are not available.
- `CAS#`: Return the CAS number.
- `CHEMFORM`: Return the short chemical formula.
- `SYNONYM`: Return the synonym found on the fifth line in the fluid files.
- `FAMILY`: Return the family class used for several predictive schemes.
- `FLDNAME`: Return the fluid file name sent to the SETUP routine.
- `HASH`: Return the hash number.
- `INCHI`: Return the INCHI string.
- `INCHIKEY`: Return the INCHI key.
- `LONGNAME`: Return the long fluid name given in the 3rd line of the fluid files.
- `SAFETY`: Return the ASHRAE 34 classification.
- `NAME`: Return the fluid short name.
- `NCOMP`: Return the number of components.
- `UNNUMBER`: Return the UN number.
- `DOI_###(#)`: Return the DOI of the equation given by the three letters following the underscore, where the valid letters are EOS for equation of state, VIS for viscosity, TCX for thermal conductivity, STN for surface tension, DIE for dielectic constant, MLT for melting line, and SBL for sublimation line. For

example, DOI_EOS would return the DOI for the equation of state. For mixtures, the component must be specified at the end in the parenthesis, for example, DOI_VIS(3).

- `WEB_###(#)`: Return the web address for the equation given by the three letters following the underscore, as explained in the DOI section.
- `REFSTATE`: Return the reference state in use (NBP, IIR, ASH, OTH, etc.).
- `GWP`: Return the global warming potential (found in the fluid file header).
- `ODP`: Return the ozone depletion potential (found in the fluid file header).
- `FDIR`: Return the location (directory) of the fluid file. The directory is returned in both the hUnits string and in herr if no other error occurred (paths that are more than 50 characters long are truncated in hUnits). For mixtures, send FDIR(2), etc., to get the path of the second fluid and so on.
- `UNITSTRING`: Return the units of the property (e.g., K, psia, kg/m^3, J/mol, etc.) identified in hIn for the unit system defined in hFld (e.g., SI, E, etc.). The input values for hIn are the labels described in the ALLPROPS routine. For example, 'D2DDP2' would return '(kg/m^3)/MPa^2' for 'SI' inputs.
- `UNITNUMB`: Return in iUCode the integer value associated with a particular set of units defined in hFld (SI, E, etc.). This integer value can then be used in subsequent calls for the iUnits variable.
- `UNITS`: Perform both operations in UNITSTRING and UNITNUMB.
- `UNITCONV`: Convert the property contained in the variable a from units given in hFLD to units given in hIn. The unit strings are given much further below. When converting from mole to mass units (or vice versa), the molar mass must be sent in the variable b. The type of property (as specified in the CONVUNITS subroutine) must be appended to the string in hOut, for example, hOut='UNITCONV_T' or hOut='UNITCONV_D'.
- `UNITUSER`, `UNITUSER2`: Set a predefined set of units based on the user's need. Two different sets can be assigned depending on the input sent to the routine. The variable hIn contains the numbers that are specified by the enumerations in the CONSTS.INC file, separated by semicolons. For example, hIn='0;157;0;0;0;403;0;0;0;0' would set the pressure to use units of atm and the speed of sound to use units of km/h. The numbers are listed in the order of T, P, D, H, S, W, I, E, K, and N (temperature, pressure, density, enthalpy, entropy, speed of sound, kinematic viscosity, viscosity, thermal conductivity, and surface tension). Because the enumerations might change, it is best to build this string with the enumerations listed in the CONSTS.INC file rather than hard coding the numbers as shown above.
- `DLL#`: Return the version number of the DLL in iUCode and the string value in hUnits.
- `PHASE`: Return the phase of the state for the input fluids and properties. See subroutine PHASE for a listing of all possibilities. The output is sent back in the hUnits variable. No other command can be sent with this one since hUnits is not an array.
- `FULLCHEMFORM`: Return the long chemical formula.
- `HEATINGVALUE`: Return the upper heating value.
- `LIQUIDFLUIDSTRING`: Return a string that contains the fluid names and compositions for the liquid phase of a two-phase state.
- `VAPORFLUIDSTRING`: Return a string that contains the fluid names and compositions for the vapor phase of a two-phase state. For example, "R32;R125|0.25;0.75". The string is passed back in hUnits.
- `QMOLE`: Return the molar quality for 2-phase states.
- `QMASS`: Return the mass quality for 2-phase states.
- `XMASS`: Return the mass compositions in the Output array as with the X command. See comment about `Qmass`.
- `XLIQ`: Return the mass or molar liquid compositions (depending on the value of iMass) for 2-phase states.

- XVAP: Return the mass or molar vapor compositions (depending on the value of iMass) for 2-phase states.
- XMOLELIQ: Return the liquid compositions for 2-phase states on a mole basis regardless of the iMass variable.
- XMOLEVAP: Return the vapor compositions for 2-phase states on a mole basis regardless of the iMass variable.
- XMASSLIQ: Return the liquid compositions for 2-phase states on a mass basis regardless of the iMass variable.
- XMASSVAP: Return the vapor compositions for 2-phase states on a mass basis regardless of the iMass variable.
- *LIQ: (where * is T, P, D, etc.) Return the liquid saturation properties for the property listed as the first letter. This is only valid for saturation states or 2-phase states.
- *VAP: (where * is T, P, D, etc.) Return the vapor saturation properties for the property listed as the first letter. This is only valid for saturation states or 2-phase states.
- FIJMIX: Return the mixing parameters in the first six slots of the variable Output for the binary mixture identified by the values in the a and b variables (i.e., integer values are sent in double precision variables). The mixing rule is returned in the hUnits string.

Information on iUnits

Multiple unit systems are available for use in property values, such as the SI system, English system, mixed sets, and so forth. Each set is identified with an enumerated value, which is sent as an input code in iUnits.

<FORTRAN ONLY> The enumerated value for the different unit systems are listed below and in the CONSTS.INC file, which can be included in your FORTRAN program, as such:

```
include 'CONSTS.INC'
```

> **Warning:** Do NOT include any other INC file in your programs

The enumerated values for the unit systems are given by the parameters

- iUnitsMolSI
- iUnitsSI
- ...

</FORTRAN ONLY>

In all environments other than FORTRAN, the iUnits variable should be retrieved from the GETENUM function with a call like:

```
GETENUMdll(0,'MOLAR BASE SI',iEnum,ierr,herr)
```

> **Warning:** The integer values for iUnits given below should **NEVER** be used directly, you should always retrieve the enumerated value from GETENUM. This is to allow the developers of Refprop flexibility in the future.

The unit systems used in Refprop are as follows:

	DEFAULT	MOLAR SI	MASS SI	SI WITH C
iUnits --->	0	1	2	3
Temperature	K	K	K	C
Pressure	kPa	MPa	MPa	MPa
Density	mol/dm^3	mol/dm^3	kg/m^3	kg/m^3
Enthalpy	J/mol	J/mol	J/g	J/g
Entropy	(J/mol)/K	(J/mol)/K	(J/g)/K	(J/g)/K
Speed	m/s	m/s	m/s	m/s
Kinematic vis.	cm^2/s	cm^2/s	cm^2/s	cm^2/s
Viscosity	uPa-s	uPa-s	uPa-s	uPa-s
Thermal cond.	W/(m-K)	mW/(m-K)	mW/(m-K)	mW/(m-K)
Surface tension	N/m	mN/m	mN/m	mN/m
Molar Mass	g/mol	g/mol	g/mol	g/mol

	MOLAR BASE SI	MASS BASE SI	ENGLISH	MOLAR ENGLISH
iUnits --->	20	21	5	6
Temperature	K	K	F	F
Pressure	Pa	Pa	psia	psia
Density	mol/m^3	kg/m^3	lbm/ft^3	lbmol/ft^3
Enthalpy	J/mol	J/kg	Btu/lbm	Btu/lbmol
Entropy	(J/mol)/K	(J/kg)/K	(Btu/lbm)/R	(Btu/lbmol)/R
Speed	m/s	m/s	ft/s	ft/s
Kinematic vis.	m^2/s	m^2/s	ft^2/s	ft^2/s
Viscosity	Pa-s	Pa-s	lbm/(ft-s)	lbm/(ft-s)
Thermal cond.	W/(m-K)	W/(m-K)	Btu/(h-ft-R)	Btu/(h-ft-R)
Surface tension	N/m	N/m	lbf/ft	lbf/ft
Molar Mass	kg/mol	kg/mol	lbm/lbmol	lbm/lbmol

	MKS	CGS	MIXED	MEUNITS
iUnits --->	7	8	9	10
Temperature	K	K	K	C
Pressure	kPa	MPa	psia	bar
Density	kg/m^3	g/cm^3	g/cm^3	g/cm^3
Enthalpy	J/g	J/g	J/g	J/g
Entropy	(J/g)/K	(J/g)/K	(J/g)/K	(J/g)/K
Speed	m/s	cm/s	m/s	cm/s
Kinematic vis.	cm^2/s	cm^2/s	cm^2/s	cm^2/s
Viscosity	uPa-s	uPa-s	uPa-s	cpoise
Thermal cond.	W/(m-K)	mW/(m-K)	mW/(m-K)	mW/(m-K)
Surface tension	mN/m	dyne/cm	mN/m	mN/m
Molar Mass	g/mol	g/mol	g/mol	g/mol

	USER (can be changed by calling the REFPROP subroutine)
iUnits --->	11
Temperature	C
Pressure	psig
Density	kg/m^3
Enthalpy	J/g
Entropy	(J/g)/K
Speed	m/s
Kinematic vis.	cm^2/s
Viscosity	mPa-s
Thermal cond.	W/(m-K)
Surface tension	N/m
Molar Mass	g/mol

Information on iUCode output

The iUCode variable uses a four digit code that specifies the units of the property:

- Left digit : Energy unit in J/mol or kJ/kg
- Left middle digit : Density unit in mol/dm^3 or kg/m^3
- Right middle digit : Pressure unit in kPa
- Right digit : Temperature unit in K

Each digit indicates the power of the unit, for example, a value of 2 for the temperature digit corresponding to K^2. Values from 6 to 9 specify a negative power digit, for example, a value of 8 would be 1/kPa^2.

The following values give other examples:

```
1000    J/mol
0100    mol/dm^3
0010    kPa
0001    K
0000    -  (a value of zero assumes a dimensionless unit)
9000    1/(J/mol)
0910    kPa/(mol/dm^3)
0190    (mol/dm^3)/kPa
8765    K^5/[(J/mol)^2*(mol/dm^3)^3*kPa^4]
2082    (J/mol)^2*K^2/kPa^2
0830    kPa^3/(mol/dm^3)^2
9281    (mol/dm^3)^2*K/[(J/mol)*kPa^2]
8139    (mol/dm^3)*kPa^3/[(J/mol)^2*K]
2288    (J/mol)^2*(mol/dm^3)^2/[kPa^2*K^2]
1764    (J/mol)*K^4/[(mol/dm^3)^3*kPa^4]
4857    (J/mol)^4*kPa^5/[(mol/dm^3)^2*K^3]
2730    (J/mol)^2*kPa^3/(mol/dm^3)^3
6666    1/[(J/mol)^4*(mol/dm^3)^4*kPa^4*K^4]
```

Negative values represent special units not built on these four property types:

Property	Parameter	Current Value (but subject to change)
Speed of sound	iUTypeW	-9
Viscosity	iUTypeU	-10
Thermal conductivity	iUTypeK	-11
Surface tension	iUTypeN	-12
Quality	iUType0	-13
Molar mass	iUTypeM	-14
Kinematic viscosity	iUTypeI	-17
Mass flux	iUTypeF	-27
Heating value (volume)	iUTypeG	-37
Dipole moment	iUTypeB	-38

The dimension statements for these variables are (in Fortran):

```
parameter (ncmax=20)                        !Maximum number of components in the␣
↪mixture
parameter (iPropMax=200)                    !Number of output properties available␣
↪in ALLPROPS.
character*255 hFld,hIn,hOut,hUnits,herr             !hFld, hIn, and hOut can␣
↪actually be of any length.
integer iUnits,iMass,iFlag,ierr,iUCode              !Note: as integer*4
double precision a,b,q,Output(iPropMax),z(ncmax),x(ncmax),y(ncmax),x3(ncmax)
```

Parameters

- **hFld** *[char ,in]* :: Fluid string. See above
- **hIn** *[char ,in]* :: Input string of properties being sent to the routine.
- **hOut** *[char ,in]* :: Various flags are available to gain access to all other features of the Refprop program.
- **iUnits** *[int ,in]* :: The unit system to be used for the input and output properties (such as SI, English, etc.) See the details much further below for a complete description of the iUnits input value. **NOTE** A mass based value for iUnits does not imply that the input and output compositions are on a mass basis, this is specified with the iMass variable.
- **iMass** *[int ,in]* :: Specifies if the input composition is mole or mass based
- **iFlag** *[int ,in]* :: Flag to specify if the routine SATSPLN should be called (where a value of 1 activates the call). (Eventually this variable may be used to send multiple flags combined in this flag.)
- **a** *[double ,in]* :: First input property as specified in the hIn variable
- **b** *[double ,in]* :: Second input property as specified in the hIn variable
- **z** (20) *[double ,in]* :: Composition on a mole or mass basis depending on the value sent in iMass (array of size ncmax=20).
- **Output** (200) *[double ,out]* :: Array of properties specified by the hOut string (array of size 200 dimensioned as double precision). The number -9999970 will be returned when errors occur, and the number -9999990 will be returned when nothing was calculated. Read the comments in the ALLPROPS routine to fully understand the contents of this array.
- **hUnits** *[char ,out]* :: The units for the first property in the Output array. Strings such as a fluid name may also be passed back in this position. To obtain the units for all of the properties sent to the string, call the ALLPROPS routine instead.
- **iUCode** *[int ,out]* :: Unit code that represents the units of the first property in the Output array. See below for further details.
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions of size 20) for two-phase states on a mole or mass basis depending on the value of iMass.
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions of size 20) for two-phase states on a mole or mass basis depending on the value of iMass.
- **x3** (20) *[double ,out]* :: Reserved for returning the composition of a second liquid phase for LLE or VLLE.
- **q** *[double ,out]* :: Vapor quality on a mole or mass basis depending on the value of iMass. (See subroutine ABFLSH for the definitions of values returned for this variable). To obtain the molar quality regardless of iMass, send "qmole" as an input in hIn, and vice-versa for "qmass".
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hFld_length** *[int]* :: length of variable `hFld` (default: 10000)
- **hIn_length** *[int]* :: length of variable `hIn` (default: 255)
- **hOut_length** *[int]* :: length of variable `hOut` (default: 255)
- **hUnits_length** *[int]* :: length of variable `hUnits` (default: 255)

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `imass` flags

0 Input compositions given in mole fractions, quality on a molar basis.

1 Input compositions given in mass fractions, quality on a mass basis. For two-phase states, the values in x and y will be returned on a mass basis if iMass=1. **NOTE** If the fluid string sent to this routine contains the word "mass" at the end (and thus contains the composition as well as the names of the fluids), this will have preference over the value of iMass when converting those compositions from a mass to a molar basis. However, compositions sent back will still be based on the value in iMass.

subroutine SETFLUIDSdll (*hFld*, *ierr*, *hFld_length*)

Call the SETUP routine without the need to pass ncomp, hrf, hFmix, or herr, or to declare the length of hfld as 255 or 10000 bytes long. For a pure fluid, hfld simply contains the name of the fluid file (with a path if needed). For a mixture, it contains the names of the constituents in the mixture separated by a `|`, a semicolon, or an asterisk. To load a predefined mixture, call the SETMIXTURE subroutine (which must return the composition array and thus cannot be included here). If it is necessary to set the reference state, call SETUP instead. If ierr comes back non-zero, call the ERRMSG routine to obtain it.

Examples:

```
call SETFLUIDS ('ARGON',ierr)     (load argon as a pure fluid)
call SETFLUIDS ('FLUIDS/NITROGEN.FLD|FLUIDS/ARGON.FLD|FLUIDS/OXYGEN.FLD|',ierr)
↪(for the air mixture, but giving a path as well)
call SETFLUIDS ('AIR.PPF',ierr)  (load the air mixture, but read from the pseudo-
↪pure file; properties will be slightly different from the *.mix file since they
↪are different models)
call SETFLUIDS ('methane * ethane * propane * butane',ierr)
```

Parameters

- **hFld** *[char ,in]* :: String of any character length containing the fluid file names
- **ierr** *[int ,out]* :: Error flag
- **hFld_length** *[int]* :: length of variable `hFld` (default: 10000)

Flags `ierr` flags

0 Successful (Values are identical to SETUP; a 109 is returned if the number of fluids in hfld is less than icomp.)

subroutine SETMIXTUREdll (*hMixNme*, *z*, *ierr*, *hMixNme_length*)

Call the SETMIX routine for a predefined mixture without the need to pass hFmix, hrf, ncc, hf, or herr. It is not necessary to declare the length of hMixNme as 255 bytes long. A path can be included if needed. The extension ".mix" is not required. If it is necessary to set the reference state, call subroutine FLAGS first. The composition of the mixture will be returned in the z array. If ierr comes back non-zero, call the ERRMSG routine to obtain it.

Examples:

```
call SETMIXTURE ('AIR.MIX',z,ierr) ! load the air mixture from the AIR.MIX file
call SETMIXTURE ('C:/REFPROP/MIXTURES/AIR.MIX',z,ierr)    read the AIR.MIX file
↪from the C:/REFPROP/MIXTURES directory
call SETMIXTURE ('R410A.MIX',z,ierr) ! load the R410A mixture, the composition
↪will be returned on a mole percent basis in the z array.
call SETMIXTURE ('R410A',z,ierr)  ! works the same as above for predefined
↪refrigerant mixtures that start with R4 or R5.
```

Parameters

- **hMixNme** *[char ,in]* :: String of any character length containing the mixture file name
- **z** (20) *[double ,out]* :: Composition array (mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **hMixNme_length** *[int]* :: length of variable `hMixNme` (default: 10000)

Flags `ierr` flags

0 Successful (Values are identical to SETMIX)

subroutine SETPATHdll (*hpth*, *hpth_length*)

Set the path where the fluid files are located.

Parameters

- **hpth** *[char ,in]* :: Location of the fluid files (character*255) The path does not need to contain the ending "/" and it can point directly to the location where the DLL is stored if a fluids subdirectory (with the corresponding fluid files) is located there, for example, hpth='C:/Program Files (x86)/REFPROP'
- **hpth_length** *[int]* :: length of variable `hpth` (default: 255)

2.2 Legacy API

Warning: The functions in this legacy application program interface (API) have all been deprecated and will be removed in some future release. Please use the *High-Level API*:

2.2.1 Function Listing

- *ABFL1dll()*
- *ABFL2dll()*
- *AGdll()*
- *B12dll()*
- *BLCRVdll()*
- *CSTARdll()*
- *CHEMPOTdll()*
- *CP0dll()*
- *CRITPdll()*
- *CRTPNTdll()*
- *CSATKdll()*
- *CSTARdll()*
- *CV2PKdll()*
- *CVCPdll()*
- *DBDTdll()*

- *DBFL1dll()*
- *DEFL1dll()*
- *DEFLSHdll()*
- *DERVPVTdll()*
- *DHD1dll()*
- *DHFL1dll()*
- *DHFLSHdll()*
- *DIELECdll()*
- *DLSATKdll()*
- *DPDD2dll()*
- *DPTSATKdll()*
- *DQFL2dll()*
- *DSD1dll()*
- *DSFL1dll()*
- *DSFLSHdll()*
- *DVSATKdll()*
- *ENTHALdll()*
- *ENTROdll()*
- *ESFLSHdll()*
- *EXCESSdll()*
- *FGCTY2dll()*
- *FGCTYdll()*
- *FPVdll()*
- *FUGCOFdll()*
- *GERG04dll()*
- *GERG08dll()*
- *GETFIJdll()*
- *GETKTVdll()*
- *GETMODdll()*
- *GETREFDIRdll()*
- *GIBBSdll()*
- *HEATFRMdll()*
- *HEATdll()*
- *HMXORDERdll()*
- *HSFL1dll()*
- *HSFLSHdll()*

- *IDCRVdll()*
- *INFOdll()*
- *JICRVdll()*
- *JTCRVdll()*
- *LIMITKdll()*
- *LIMITSdll()*
- *LIMITXdll()*
- *LIQSPNDLdll()*
- *MASSFLUXdll()*
- *MAXPdll()*
- *MAXTdll()*
- *MELTKdll()*
- *MELTPdll()*
- *MELTTdll()*
- *MLTH2Odll()*
- *NAMEdll()*
- *PASSCMNdll()*
- *PDFL1dll()*
- *PDFLSHdll()*
- *PEFL1dll()*
- *PEFLSHdll()*
- *PHFL1dll()*
- *PHFLSHdll()*
- *PHI0dll()*
- *PHIDERVdll()*
- *PHIHMXdll()*
- *PHIKdll()*
- *PHIMIXdll()*
- *PHIXdll()*
- *PQFLSHdll()*
- *PREOSdll()*
- *PRESSdll()*
- *PSATKdll()*
- *PSFL1dll()*
- *PSFLSHdll()*
- *PUREFLDdll()*

- *QMASSdll()*
- *QMOLEdll()*
- *RDXHMXdll()*
- *REDXdll()*
- *RESIDUALdll()*
- *RIEMdll()*
- *RMIX2dll()*
- *SATDdll()*
- *SATESTdll()*
- *SATEdll()*
- *SATGUESSdll()*
- *SATGVdll()*
- *SATHdll()*
- *SATPdll()*
- *SATSPLNdll()*
- *SATSdll()*
- *SATTPdll()*
- *SATTdll()*
- *SETAGAdll()*
- *SETKTVdll()*
- *SETMIXdll()*
- *SETMODdll()*
- *SETNCdll()*
- *SETREFDIRdll()*
- *SETREFdll()*
- *SETUPdll()*
- *SPLNROOTdll()*
- *SPLNVALdll()*
- *STNdll()*
- *SUBLPdll()*
- *SUBLTdll()*
- *SURFTdll()*
- *SURTENdll()*
- *TDFLSHdll()*
- *TEFL1dll()*
- *TEFLSHdll()*

- *THERM0dll()*
- *THERM2dll()*
- *THERM3dll()*
- *THERMdll()*
- *THFL1dll()*
- *THFLSHdll()*
- *TPFL2dll()*
- *TPFLSHdll()*
- *TPRHOPRdll()*
- *TPRHOdll()*
- *TQFLSHdll()*
- *TRNPRPdll()*
- *TSATDdll()*
- *TSATPdll()*
- *TSFL1dll()*
- *TSFLSHdll()*
- *UNSETAGAdll()*
- *VAPSPNDLdll()*
- *VIRBAdll()*
- *VIRBCD12dll()*
- *VIRBCDdll()*
- *VIRBdll()*
- *VIRCAdll()*
- *VIRCdll()*
- *VIRTAUdll()*
- *WMOLIdll()*
- *WMOLdll()*
- *XMASSdll()*
- *XMOLEdll()*

2.2.2 Function Documentation

subroutine ABFL1dll (*a*, *b*, *z*, *kph*, *ab*, *Dmin*, *Dmax*, *T*, *P*, *D*, *ierr*, *herr*, *ab_length*, *herr_length*)

General single-phase flash routine given two inputs and composition. Valid input properties are temperature, pressure, density, energy, enthalpy, or entropy. The character string ab specifies the inputs, which can be T, P, D, E, H, S. An input of 'EH' (or 'HE') is not supported. The letters in this string must be uppercase.

Care must be taken when sending inputs of T, P, or D, so that the same variable is not sent twice. For example, the following would be wrong:

```
call ABFL1 ('TH',T,H,z,kph,0,0,Dmin,Dmax,T,P,D,ierr,herr)
```

Rather, the following are examples of correct inputs:: call ABFL1 ('TH',T,H,z,kph,0,Dmin,Dmax,tt,P, D,ierr,herr) call ABFL1 ('TP',T,P,z,kph,0,Dmin,Dmax,tt,pp,D,ierr,herr) call ABFL1 ('DS',D,S,z,kph,0,Dmin,Dmax,T, P,dd,ierr,herr)

When calling the ABFL1dll routine, the inputs are different:: call ABFL1dll (a,b,z,kph,ab,Dmin,Dmax,T,P,D,ierr,herr)

The examples above are repeated below for this call:: call ABFL1dll (T,H,z,kph,'TH',Dmin,Dmax,tt,P, D,ierr,herr) call ABFL1dll (T,P,z,kph,'TP',Dmin,Dmax,tt,pp,D,ierr,herr) call ABFL1dll (D,S,z,kph,'DS',Dmin,Dmax,T, P,dd,ierr,herr)

This routine accepts only single-phase inputs, it is intended primarily for use with the more general flash routine ABFLSH, but can be called independently for increased calculation speed if the inputs are known to be single-phase. This will avoid the call to the flash routines to determine the phase of the inputs. If this routine is called, but the inputs are 2-phase, either an incorrect root or a metastable state will be returned (which is OK if the metastable state is desired).

However! This routine is not necessary because all flash calls can be made through a call to ABFLSH or ABFLSHdll. It is recommended to use either the REFPROP or ABFLSH routines for all flash calculations even for single phase calls when it is not necessary to check the phase of the input properties. For example, the TH call above for a known liquid phase state could be called like this:

```
 iFlag=10  !Indicates that the input state is in the liquid.
           !Use iFlag=20 for vapor phase states.
 call ABFLSHdll ('TH',T,H,z,iFlag,tt,P,D,Dl,Dv,
&                   x,y,q,e,hh,s,Cv,Cp,w,ierr,herr)
```

Further information is given in the comments under the ABFLSH and REFPROP subroutines.

Parameters

- **a** *[double ,in]* :: First property (either temperature, pressure, density, entropy)
- **b** *[double ,in]* :: Second property (pressure, density, energy, enthalpy, or entropy) Possible inputs for these two variables are
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Phase flag
- **ab** *[char ,in]* :: Character*2 string defining the inputs, e.g., 'TH' or 'PS' Valid characters are T, P, D, E, H, S
- **Dmin** *[double ,in]* :: Lower bound on density [mol/L] (for T inputs)
- **Dmax** *[double ,in]* :: Upper bound on density [mol/L] (for T inputs)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Molar density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **ab_length** *[int]* :: length of variable `ab` (default: 2)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Liquid (kph is only needed for TP inputs)

2 Vapor

`ierr` flags

0 Successful

248 Single-phase iteration did not converge

subroutine ABFL2dll (*a*, *b*, *z*, *kq*, *ksat*, *ab*, *Tbub*, *Tdew*, *Pbub*, *Pdew*, *Dlbub*, *Dvdew*, *ybub*, *xdew*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *q*, *ierr*, *herr*, *ab_length*, *herr_length*)

General flash calculation given two inputs and composition. Valid properties for the first input are temperature, pressure, and density. Valid properties for the second are pressure, density, energy, enthalpy, entropy, or quality. The character string ab specifies the inputs.

This routine accepts only two-phase states as inputs; it is intended primarily for use by the general flash routines such as THFLSH or TSFLSH. It may be called independently if the state is known to be two-phase. But beware - this routine does not check limits, and it will be significantly faster than TSFLSH, etc., when the bubble and dew point limits can be provided (ksat=1 option).

This routine calls TPFL2 within a secant-method iteration to find a solution. Initial guesses are based on the liquid density at the bubble point and the vapor density at the dew point.

Parameters

- **a** *[double ,in]* :: First property (either temperature, pressure, or density)
- **b** *[double ,in]* :: Second property (pressure, density, energy, enthalpy, entropy, or quality)
- **z** (20) *[double ,in]* :: Overall composition (array of mole fractions)
- **kq** *[int ,in]* :: Flag specifying units for input quality when b=quality
- **ksat** *[int ,in]* :: Flag for bubble and dew point limits
- **ab** *[char ,in]* :: Character*2 string defining the inputs, e.g., 'TD' or 'PQ'
- **Tbub** *[double ,in]* :: Bubble point temperature [K] at (P and x=z)
- **Tdew** *[double ,in]* :: Dew point temperature [K] at (P and y=z) For temperature inputs
- **Pbub** *[double ,in]* :: Bubble point pressure [kPa] at (T and x=z)
- **Pdew** *[double ,in]* :: Dew point pressure [kPa] at (T and y=z) For either case
- **Dlbub** *[double ,in]* :: Liquid density [mol/L] at bubble point
- **Dvdew** *[double ,in]* :: Vapor density [mol/L] at dew point
- **ybub** (20) *[double ,in]* :: Vapor composition (array of mole fractions) at bubble point
- **xdew** (20) *[double ,in]* :: Liquid composition (array of mole fractions) at dew point
- **T** *[double ,out]* :: Temperature [K] (if not an input)
- **P** *[double ,out]* :: Pressure [kPa] (if not an input)
- **Dl** *[double ,out]* :: Liquid density [mol/L] at bubble point
- **Dv** *[double ,out]* :: Vapor density [mol/L] at dew point
- **x** (20) *[double ,out]* :: Liquid composition (array of mole fractions)
- **y** (20) *[double ,out]* :: Vapor composition (array of mole fractions)

- **q** *[double ,out]* :: Vapor quality, the definitions of the values for q are given in the ABFLSH routine.
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **ab_length** *[int]* :: length of variable `ab` (default: 2)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kq` flags

1 Quality on molar basis (moles vapor/total moles)

2 Quality on mass basis (mass vapor/total mass)

`ksat` flags

0 Dew and bubble point limits computed here

1 Must provide values for the following: For pressure and density inputs

`ierr` flags

0 Successful

223 Bubble point calculation did not converge

224 Dew point calculation did not converge

226 2-phase iteration did not converge

subroutine AGdll (*T*, *D*, *z*, *a*, *g*)

Compute Helmholtz and Gibbs energies as functions of temperature, density, and composition. These are not residual values (those are calculated by GIBBS). See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **a** *[double ,out]* :: Helmholtz energy [J/mol]
- **g** *[double ,out]* :: Gibbs free energy [J/mol]

subroutine B12dll (*T*, *z*, *B*)

Compute B12 as a function of temperature and composition for a binary mixture.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **B** *[double ,out]* :: B12 [L/mol]

subroutine BLCRVdll (*D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Calculate the temperature along the Boyle curve for the input density. This line starts at zero density at the temperature where B=0, and passes into the liquid phase without crossing the two-phase. It ends at a saturated liquid state very close to the critical point. The argument z in this routine is an array with the mole fractions of the mixture. If the input T is non-zero, it is used as the initial guess.

Parameters

- **D** *[double ,in]* :: Density [mol/l]

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Iteration failed to converge

subroutine CSTARdll (*T*, *P*, *v*, *z*, *Cs*, *Ts*, *Ds*, *Ps*, *ws*, *ierr*, *herr*, *herr_length*)

Calculate the critical flow factor, C*, for nozzle flow of a gas (subroutine was originally named CCRIT).

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **v** *[double ,in]* :: Plenum velocity [m/s] (Should generally be set to 0 for calculating stagnation conditions.)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Cs** *[double ,out]* :: Critical flow factor [dimensionless]
- **Ts** *[double ,out]* :: Nozzle throat temperature [K]
- **Ds** *[double ,out]* :: Nozzle throat molar density [mol/L]
- **Ps** *[double ,out]* :: Nozzle throat pressure [kPa]
- **ws** *[double ,out]* :: Nozzle throat speed of sound [m/s]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 CSTAR did not converge

subroutine CHEMPOTdll (*T*, *D*, *z*, *u*, *ierr*, *herr*, *herr_length*)

Compute the chemical potentials for each of the nc components of a mixture.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **u** (20) *[double ,out]* :: Array (1..nc) of the chemical potentials [J/mol]
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine CP0dll (*T*, *z*, *Cp*)

Calculate Cp0 for a mixture given temperature and composition.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **Cp** *[double]* :: XXXXXXXXXX

subroutine CRITPdll (*z*, *Tc*, *Pc*, *Dc*, *ierr*, *herr*, *herr_length*)

Calculate critical parameters as a function of composition. The critical parameters are estimates based on polynomial fits to the binary critical lines. For 3 or more components, combining rules are applied to the constituent binaries.

If SATSPLN has been called and the input composition sent here is the same as that sent to SATSPLN, the values calculated from the splines are returned, which are nearly exact. During the call to SATSPLN, the true critical point, maximum pressure point, and maximum temperature point along the saturation lines are determined. Without the splines and for a system with three or more components, the values from this routine are only rough estimates.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Tc** *[double ,out]* :: Critical temperature [K]
- **Pc** *[double ,out]* :: Critical pressure [kPa]
- **Dc** *[double ,out]* :: Critical density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful (See subroutine CRTHMX for error numbers.)

subroutine CRTPNTdll (*z*, *Tc*, *Pc*, *Dc*, *ierr*, *herr*, *herr_length*)

Subroutine for the determination of the true critical point of a mixture with the use of the method of Michelsen (1984).

The routine requires good initial guess values of Pc and Tc.

On convergence, the values of bb and cc should be close to zero and dd > 0 for a two-phase critical point. bb=0, cc=0, and dd <= 0 for an unstable critical point.

Parameters

- **z** (20) *[double ,in]* :: Composition [array of mole fractions]
- **Tc** *[double ,out]* :: Critical temperature [K]
- **Pc** *[double ,out]* :: Critical pressure [kPa]
- **Dc** *[double ,out]* :: Critical density [mol/l]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine CSATKdll (*icomp*, *T*, *kph*, *P*, *D*, *Csat*, *ierr*, *herr*, *herr_length*)
Compute the heat capacity along the saturation line as a function of temperature for a given component.

Csat can be calculated in several ways Csat = T*(dS/dT[sat]) Csat = Cp - T*(dV/dT)(dP/dT[sat]) with dVdT at constant pressure Csat = Cp - beta/D*hvap/(vliq - vvap) where beta is the volume expansivity

Parameters

- **icomp** *[int ,in]* :: Component number in mixture (1..nc); 1 for pure fluid
- **T** *[double ,in]* :: Temperature [K]
- **kph** *[int ,in]* :: Phase flag
- **P** *[double ,out]* :: Saturated pressure [kPa]
- **D** *[double ,out]* :: Saturated molar density [mol/L]
- **Csat** *[double ,out]* :: Saturated heat capacity [J/mol-K]
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Liquid calculation

2 Vapor calculation

subroutine CSTARdll (*T*, *P*, *v*, *z*, *Cs*, *Ts*, *Ds*, *Ps*, *ws*, *ierr*, *herr*, *herr_length*)
Calculate the critical flow factor, C*, for nozzle flow of a gas (subroutine was originally named CCRIT).

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **v** *[double ,in]* :: Plenum velocity [m/s] (Should generally be set to 0 for calculating stagnation conditions.)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Cs** *[double ,out]* :: Critical flow factor [dimensionless]
- **Ts** *[double ,out]* :: Nozzle throat temperature [K]
- **Ds** *[double ,out]* :: Nozzle throat molar density [mol/L]
- **Ps** *[double ,out]* :: Nozzle throat pressure [kPa]
- **ws** *[double ,out]* :: Nozzle throat speed of sound [m/s]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 CSTAR did not converge

subroutine CV2PKdll (*icomp*, *T*, *D*, *Cv2p*, *Csat*, *ierr*, *herr*, *herr_length*)
Compute the isochoric heat capacity in the two phase (liquid+vapor) region.

Parameters

- **icomp** *[int ,in]* :: Component number in mixture (1..nc); 1 for pure fluid
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Density [mol/L] if known If D=0, then a saturated liquid state is assumed.
- **Cv2p** *[double ,out]* :: Isochoric two-phase heat capacity [J/mol-K]
- **Csat** *[double ,out]* :: Saturation heat capacity [J/mol-K] (Although there is already a Csat routine in Refprop, it is also returned here. However, the calculation speed is slower than Csat.)
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine CVCPdll (*T*, *D*, *z*, *Cv*, *Cp*)

Parameters

- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **Cv** *[double ,in]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]

subroutine DBDTdll (*T*, *z*, *dBT*)

Compute the 1st derivative of B [dBT (L/mol-K)] as a function of temperature T (K) and composition x (array of mole fractions). This routine approximates dBT. For pure fluids, the routine VIRBCD is exact.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **dBT** *[double ,out]* :: 1st derivative of B with respect to T [L/(mol-K)]

subroutine DBFL1dll (*D*, *b*, *z*, *hab*, *T*, *P*, *ierr*, *herr*, *hab_length*, *herr_length*)

General single-phase calculation given density, composition, and either pressure, energy, enthalpy, or entropy. The character string ab specifies the inputs. This routine should ONLY be called by ABFL1.

Parameters

- **D** *[double ,in]* :: Molar density [mol/L]
- **b** *[double ,in]* :: Second property (pressure, energy, enthalpy, or entropy)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **hab** *[char]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hab_length** *[int]* :: length of variable `hab` (default: 2)

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

207 Density or pressure equal to zero, no solution available

208 Iteration did not converge

subroutine DEFL1dll (*D*, *e*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature as a function of density, energy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DEFLSHdll (*D*, *e*, *z*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *q*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given density, energy, and bulk composition. (See subroutines ABFLSH or DBFLSH for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DERVPVTdll (*T*, *D*, *z*, *dPdD*, *dPdT*, *d2PdD2*, *d2PdT2*, *d2PdTD*, *dDdP*, *dDdT*, *d2DdP2*, *d2DdT2*, *d2DdPT*, *dTdP*, *dTdD*, *d2TdP2*, *d2TdD2*, *d2TdPD*)

Compute 1st and 2nd order derivatives of temperature, pressure, and density from core functions for Helmholtz energy equations only. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **dPdD** *[double ,out]* :: dP/dD at constant T [kPa/(mol/dm^3)]
- **dPdT** *[double ,out]* :: dP/dT at constant D [kPa/K]
- **d2PdD2** *[double ,out]* :: d^2P/dD^2 at constant T [kPa/(mol/dm^3)^2]
- **d2PdT2** *[double ,out]* :: d^2P/dT^2 at constant D [kPa/K^2]
- **d2PdTD** *[double ,out]* :: d^2P/dTdD [J/mol-K] [kPa/K/(mol/dm^3)]
- **dDdP** *[double ,out]* :: dD/dP at constant T [mol/(dm^3-kPa)]
- **dDdT** *[double ,out]* :: dD/dT at constant P [mol/(dm^3-K)]
- **d2DdP2** *[double ,out]* :: d^2D/dP^2 at constant T [(mol/dm^3)/kPa^2]
- **d2DdT2** *[double ,out]* :: d^2D/dT^2 at constant P [(mol/dm^3)/K^2]
- **d2DdPT** *[double ,out]* :: d^2D/dPdT [J/mol-K] [(mol/dm^3)/(kPa-K)]
- **dTdP** *[double ,out]* :: dT/dP at constant D [K/kPa]
- **dTdD** *[double ,out]* :: dT/dD at constant P [K/(mol/dm^3)]
- **d2TdP2** *[double ,out]* :: d^2T/dP^2 at constant D [K/kPa^2]
- **d2TdD2** *[double ,out]* :: d^2T/dD^2 at constant P [K/(mol/dm^3)^2]
- **d2TdPD** *[double ,out]* :: d^2T/dPdD [J/mol-K] [K/kPa/(mol/dm^3)]

subroutine DHD1dll (*T*, *D*, *z*, *dhdt_d*, *dhdt_p*, *dhdd_t*, *dhdd_p*, *dhdp_t*, *dhdp_d*)

Compute partial derivatives of enthalpy w.r.t. T, P, or D at constant T, P, or D as a function of temperature, density, and composition. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **dhdt_d** *[double ,out]* :: DH/dT at constant density [J/mol-K]
- **dhdt_p** *[double ,out]* :: dH/dT at constant pressure [J/mol-K]
- **dhdd_t** *[double ,out]* :: dH/dD at constant temperature [(J/mol)/(mol/L)]
- **dhdd_p** *[double ,out]* :: dH/dD at constant pressure [(J/mol)/(mol/L)]
- **dhdp_t** *[double ,out]* :: dH/dP at constant temperature [J/(mol-kPa)]
- **dhdp_d** *[double ,out]* :: dH/dP at constant density [J/(mol-kPa)]

subroutine DHFL1dll (*D*, *h*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature as a function of density, enthalpy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DHFLSHdll (*D*, *h*, *z*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given density, enthalpy, and bulk composition. (See subroutines ABFLSH or DBFLSH for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DIELECdll (*T*, *D*, *z*, *de*)

Compute dielectric constant as a function of temperature, density, and composition.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **de** *[double ,out]* :: Dielectric constant [-]

subroutine DLSATKdll (*icomp*, *T*, *D*, *ierr*, *herr*, *herr_length*)
Compute pure fluid saturated liquid density with appropriate equation.

Parameters

- **icomp** *[int ,in]* :: Component i
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,out]* :: Saturated liquid density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

121 Temperature greater than critical point temperature

501 No equation available

subroutine DPDD2dll (*T*, *D*, *z*, *d2PdD2*)
Compute second partial derivative of pressure w.r.t. density at constant temperature as a function of temperature, density, and composition. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **d2PdD2** *[double ,out]* :: d^2P/dD^2 [kPa-L^2/mol^2]

subroutine DPTSATKdll (*icomp*, *T*, *kph*, *P*, *D*, *Csat*, *dPdT*, *ierr*, *herr*, *herr_length*)
Compute the heat capacity and dP/dT along the saturation line as a function of temperature for a given component. See also subroutine CSATK.

Parameters

- **icomp** *[int ,in]* :: Component number in mixture (1..nc); 1 for pure fluid
- **T** *[double ,in]* :: Temperature [K]
- **kph** *[int ,in]* :: Phase flag
- **P** *[double ,out]* :: Saturated pressure [kPa]
- **D** *[double ,out]* :: Saturated molar density [mol/L]
- **Csat** *[double ,out]* :: Saturated heat capacity [J/mol-K] (same as that called from CSATK)
- **dPdT** *[double ,out]* :: dP/dT along the saturation line [kPa/K] (this is not dP/dT at the saturation line for the single phase state, but the change in saturated vapor pressure as the saturation temperature changes.)
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Liquid calculation

2 Vapor calculation

subroutine DQFL2dll (*D*, *q*, *z*, *kq*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Flash calculation given bulk density, quality, and composition. (See subroutine ABFL2 for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **q** *[double ,in]* :: Vapor quality [mol/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kq** *[int]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DSD1dll (*T*, *D*, *z*, *dsdt_d*, *dsdt_p*, *dsdd_t*, *dsdd_p*, *dsdp_t*, *dsdp_d*)

Compute partial derivatives of entropy w.r.t. T, P, or D at constant T, P, or D as a function of temperature, density, and composition. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **dsdt_d** *[double ,out]* :: dS/dT at constant density [J/mol-K^2]
- **dsdt_p** *[double ,out]* :: dS/dT at constant pressure [J/mol-K^2]
- **dsdd_t** *[double ,out]* :: dS/dD at constant temperature [(J/mol-K)/(mol/L)]
- **dsdd_p** *[double ,out]* :: dS/dD at constant pressure [(J/mol-K)/(mol/L)]
- **dsdp_t** *[double ,out]* :: dS/dP at constant temperature [J/(mol-K-kPa)]
- **dsdp_d** *[double ,out]* :: dS/dP at constant density [J/(mol-K-kPa)]

subroutine DSFL1dll (*D*, *s*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature as a function of density, entropy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DSFLSHdll (*D*, *s*, *z*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given density, entropy, and bulk composition. (See subroutines ABFLSH or DBFLSH for the description of all variables.)

Parameters

- **D** *[double ,in]* :: Density [mol/L]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine DVSATKdll (*icomp*, *T*, *D*, *ierr*, *herr*, *herr_length*)

Compute pure fluid saturated vapor density with appropriate equation.

Parameters

- **icomp** *[int ,in]* :: Component i
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,out]* :: Saturated vapor density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

121 Temperature greater than critical point temperature

501 No equation available

subroutine ENTHALdll (*T*, *D*, *z*, *h*)

Parameters

- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **h** *[double ,out]* :: Enthalpy [J/mol]

subroutine ENTROdll (*T*, *D*, *z*, *s*)

Parameters

- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **s** *[double ,out]* :: Entropy [J/mol-K]

subroutine ESFLSHdll (*e*, *s*, *z*, *T*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *h*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given bulk energy, entropy, and composition. (See subroutines ABFLSH or DBFLSH for the description of all variables.)

Parameters

- **e** *[double ,in]* :: Internal energy [J/mol]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)

- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine EXCESSdll (*T*, *P*, *z*, *kph*, *D*, *vE*, *eE*, *hE*, *sE*, *aE*, *gE*, *ierr*, *herr*, *herr_length*)

Compute excess properties as a function of temperature, pressure, and composition.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Phase flag
- **D** *[double ,out]* :: Molar density [mol/L] (Send a negative density to the routine to use it as an initial guess.)
- **vE** *[double ,out]* :: Excess volume [L/mol]
- **eE** *[double ,out]* :: Excess energy [J/mol]
- **hE** *[double ,out]* :: Excess enthalpy [J/mol]
- **sE** *[double ,out]* :: Excess entropy [J/mol-K]
- **aE** *[double ,out]* :: Excess Helmholtz energy [J/mol]
- **gE** *[double ,out]* :: Excess Gibbs energy [J/mol]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Liquid

2 Vapor

0 Stable phase

subroutine FGCTY2dll (*T*, *D*, *z*, *f*, *ierr*, *herr*, *herr_length*)

Compute fugacity for each of the nc components of a mixture by analytical differentiation of the dimensionless residual Helmholtz energy. These are based on derivations in the GERG-2004 document for natural gas.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **f** (20) *[double ,out]* :: Array (1..nc) of fugacities [kPa]
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine FGCTYdll (*T*, *D*, *z*, *f*)

Old routine to compute fugacity for each of the nc components of a mixture by numerical differentiation (with central differences) of the dimensionless residual Helmholtz energy.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **f** (20) *[double ,out]* :: Array (1..nc) of fugacities [kPa]

subroutine FPVdll (*T*, *D*, *P*, *z*, *Fpvx*)
Compute the supercompressibility factor, Fpv.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Fpvx** *[double ,out]* :: Fpv = SQRT[Z(60 F, 14.73 psia)/Z(T,P)]

subroutine FUGCOFdll (*T*, *D*, *z*, *phi*, *ierr*, *herr*, *herr_length*)
Compute the fugacity coefficient for each of the nc components of a mixture.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **phi** (20) *[double ,out]* :: Array (1..nc) of the fugacity coefficients [-]
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine GERG04dll (*ncomp*, *iFlag*, *ierr*, *herr*, *herr_length*)
This is a duplicate of the GERG08 routine below, and is meant only for use with older versions of Refprop.

Parameters

- **ncomp** *[int ,in]* :: Number of components (no longer used)
- **iFlag** *[int ,in]* :: Set to 1 to load the GERG 2008 equations, set to 0 for defaults
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) (returned from SETMOD)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine GERG08dll (*ncomp*, *iFlag*, *ierr*, *herr*, *herr_length*)
Use the GERG 2008 formulation for all pure fluid and mixture calculations.

This subroutine must be called before SETUP; it need not be called at all if the default (NIST-recommended) models are desired. To turn off the GERG settings, call this routine again with iFlag=0, and then call the SETUP routine to reset the parameters of the equations of state. Once this routine is called, it need not be called again to keep the GERG08 model active, even when calling SETUP.

Parameters

- **ncomp** *[int ,in]* :: Number of components (no longer used)
- **iFlag** *[int ,in]* :: Set to 1 to load the GERG 2008 equations, set to 0 for defaults
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) (returned from SETMOD)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine GETFIJdll (*hmodij*, *fij*, *hfij*, *hmxrul*, *hmodij_length*, *hfij_length*, *hmxrul_length*)

Retrieve parameter info for a specified mixing rule.

Parameters

- **hmodij** *[char ,in]* :: Mixing rule for the binary pair i,j (e.g., LJ6 or KW0) (character*3)
- **fij** (6) *[double ,out]* :: Binary mixture parameters (array of dimension nmxpar; currently nmxpar is set to 6). The parameters will vary depending on hmodij.
- **hfij** *[char ,out]* :: Description of the binary mixture parameters (character*8 array of dimension nmxpar)
- **hmxrul** *[char ,out]* :: Description of the mixing rule (character*255)
- **hmodij_length** *[int]* :: length of variable `hmodij` (default: 3)
- **hfij_length** *[int]* :: length of variable `hfij` (default: 255)
- **hmxrul_length** *[int]* :: length of variable `hmxrul` (default: 255)

subroutine GETKTVdll (*icomp*, *jcomp*, *hmodij*, *fij*, *hFmix*, *hfij*, *hbinp*, *hmxrul*, *hmodij_length*, *hFmix_length*, *hfij_length*, *hbinp_length*, *hmxrul_length*)

Retrieve mixture model and parameters for a specified binary mixture. This subroutine should not be called until after SETUP has been called. The order of icomp and jcomp do not matter, the routine returns the parameters as stored in the HMX.BNC file. To determine if the compositions are backwards, call HMXORDER. If calling SETMIX with the same parameters, an error will be returned if the components are backwards.

Kunz-Wagner model (KW0)	Lemmon-Jacobsen model (LJ6)
fij(1) = betaT	fij(1) = zeta
fij(2) = gammaT	fij(2) = xi
fij(3) = betaV	fij(3) = Fij
fij(4) = gammaV	fij(4) = beta
fij(5) = Fij	fij(5) = gamma
fij(6) = 'not used'	fij(6) = 'not used'

Parameters

- **icomp** *[int ,in]* :: Component i
- **jcomp** *[int ,in]* :: Component j
- **hmodij** *[char ,out]* :: Mixing rule for the binary pair i,j (e.g., KW0, LJ6, XR0, or LIN) (character*3)
- **fij** (6) *[double ,out]* :: Binary mixture parameters (array of dimension nmxpar; currently nmxpar is set to 6); the parameters will vary depending on hmodij;
- **hFmix** *[char ,out]* :: File name (character*255) containing parameters for the binary mixture model
- **hfij** *[char ,out]* :: Description of the binary mixture parameters (character*8 array of dimension nmxpar) The parameters will vary depending on hmodij.

- **hbinp** *[char ,out]* :: Documentation for the binary parameters (character*255)
- **hmxrul** *[char ,out]* :: Description of the mixing rule (character*255)
- **hmodij_length** *[int]* :: length of variable `hmodij` (default: 3)
- **hFmix_length** *[int]* :: length of variable `hFmix` (default: 255)
- **hfij_length** *[int]* :: length of variable `hfij` (default: 255)
- **hbinp_length** *[int]* :: length of variable `hbinp` (default: 255)
- **hmxrul_length** *[int]* :: length of variable `hmxrul` (default: 255)

subroutine GETMODdll (*icomp*, *htype*, *hcode*, *hcite*, *htype_length*, *hcode_length*, *hcite_length*)

Retrieve citation information for the property models used.

Parameters

- **icomp** *[int ,in]* :: Pointer specifying component number; zero and negative values are used for ECS reference fluid(s)
- **htype** *[char ,in]* :: Flag indicating which model is to be retrieved (character*3)
- **hcode** *[char ,out]* :: Component model used for property specified in htype (character*3)
- **hcite** *[char ,out]* :: Component model used for property specified in htype; the first 3 characters repeat the model designation of hcode and the remaining are the citation for the source (character*255)
- **htype_length** *[int]* :: length of variable `htype` (default: 3)
- **hcode_length** *[int]* :: length of variable `hcode` (default: 3)
- **hcite_length** *[int]* :: length of variable `hcite` (default: 255)

Flags `htype` flags

'EOS' Equation of state

'CP0' Ideal-gas heat capacity

'ETA' Viscosity

'TCX' Thermal conductivity

'TKK' Thermal conductivity critical enhancement

'STN' Surface tension

'DE ' Dielectric constant

'MLT' Melting line (i.e., freezing line)

'SBL' Sublimation line

'PS ' Vapor pressure equation

'DL ' Saturated liquid density equation

'DV ' Saturated vapor density equation

`hcode` flags

'FEQ' Helmholtz energy model

'ECS' Extended corresponding states (all fluids)

'VS1' The 'composite' model for R134a, R152a, NH3, etc.

'VS2' Younglove-Ely model for hydrocarbons

'VS4' Generalized friction theory of Quinones-Cisneros and Dieters

'VS5' Chung et al. model

'VS6' Vesovic form of VS1 model

'VS7' Polynomial/exponential model

'TC1' The 'composite' model for R134a, R152a, etc.

'TC2' Younglove-Ely model for hydrocarbons

'TC5' Predictive model of Chung et al. (1988)

'ST1' surface tension as f(tau); tau = 1 - T/Tc

subroutine GETREFDIRdll (*hpth*, *hpth_length*)
Get the path where the original fluid files are located. See SETREFDIR for more information.

Parameters

- **hpth** *[char ,out]* :: Location of the original fluid files (character*255)
- **hpth_length** *[int]* :: length of variable `hpth` (default: 255)

subroutine GIBBSdll (*T*, *D*, *z*, *ar*, *gr*)
Compute residual Helmholtz and Gibbs energies as functions of temperature, density, and composition from core functions, calculated as:

```
G(T,D) - G0(T,P*) = G(T,D) - G0(T,D) + RTln(RTD/P*)
```

where G0 is the ideal-gas state and P* is a reference pressure that is equal to the current pressure of interest. Since Gr is used only as a difference in phase equilibria calculations where the temperature and pressure of the phases are equal, the (RT/P*) part of the log term will cancel and is omitted. Normal (not residual) A and G are computed by subroutine AG.

See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **ar** *[double ,out]* :: Residual Helmholtz energy [J/mol]
- **gr** *[double ,out]* :: Residual Gibbs free energy [J/mol]

subroutine HEATFRMdll (*T*, *D*, *z*, *hFrm*, *ierr*, *herr*, *herr_length*)
Compute the heat of formation.

The heat of formation is the heat required to form a compound from its constituent elements, with the standard state defined as 298.15 K for the ideal gas.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L] (not used)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **hFrm** *[double ,out]* :: Heat of formation [J/mol]

- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

662 Not all heating values available

664 Unknown species in chemical formula

665 Error in chemical formula

subroutine HEATdll (*T*, *D*, *z*, *hg*, *hn*, *ierr*, *herr*, *herr_length*)

Compute the ideal-gas gross and net heating values.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **hg** *[double ,out]* :: Gross (or superior) heating value [J/mol]
- **hn** *[double ,out]* :: Net (or inferior) heating value [J/mol]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

662 Not all heating values available

665 Error in chemical formula

subroutine HMXORDERdll (*i*, *j*, *hcasi*, *hcasj*, *iFlag*, *ierr*, *herr*, *hcasi_length*, *hcasj_length*, *herr_length*)

Return the ID numbers in the order given in the HMX.BNC file, and a flag that indicates if the loaded fluids are in the same order.

Parameters

- **i** *[int ,in]* :: Component i
- **j** *[int ,in]* :: Component j
- **hcasi** *[char]* :: XXXXXXXXXX
- **hcasj** *[char]* :: XXXXXXXXXX
- **iFlag** *[int ,out]* :: Flag to indicate if loaded fluids are in the same order as the i,j pair
- **ierr** *[int ,out]* :: Error number, not currently used here
- **herr** *[char ,out]* :: Error message, not currently used here (character*255)
- **hcasi_length** *[int]* :: length of variable `hcasi` (default: 255)
- **hcasj_length** *[int]* :: length of variable `hcasj` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iflag` flags

0 Pair is backwards

1 Pair is in correct order (or if i=j)

2 Pair is not in HMX.BNC

subroutine HSFL1dll (*h*, *s*, *z*, *Dmin*, *Dmax*, *T*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature and density as a function of enthalpy, entropy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **h** *[double ,in]* :: Enthalpy [J/mol]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **Dmin** *[double ,in]* :: Lower bound on density [mol/L]
- **Dmax** *[double ,in]* :: Upper bound on density [mol/L]
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine HSFLSHdll (*h*, *s*, *z*, *T*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given bulk enthalpy, entropy, and composition.

Parameters

- **h** *[double ,in]* :: Overall enthalpy [J/mol]
- **s** *[double ,in]* :: Overall entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Overall molar density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,in]* :: Overall internal energy [J/mol] But only if iflag in common blocks has been set to 1, in which case the value of the internal energy should be sent in h, and the value of the enthalpy will be returned in e.
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]

- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) (See subroutine ABFLSH for the description of all other output variables.)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

260 Iterative routine is not available to find a solution.

subroutine IDCRVdll (*D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Calculate the temperature at the input density where the compressibility factor crosses from less than 1 to greater than 1 (i.e., Z=1). This line starts at zero density at the temperature where B=0, and passes into the liquid phase without crossing the two-phase. The argument z in this routine is an array with the mole fractions of the mixture. If the input T is non-zero, it is used as the initial guess.

Parameters

- **D** *[double ,in]* :: Density [mol/l]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Iteration failed to converge

subroutine INFOdll (*icomp*, *wmm*, *Ttrp*, *Tnbpt*, *Tc*, *Pc*, *Dc*, *Zc*, *acf*, *dip*, *Rgas*)

Provides fluid constants for the specified component.

Parameters

- **icomp** *[int ,in]* :: Component number in mixture; 1 for pure fluid
- **wmm** *[double ,out]* :: Molar mass (molecular weight) [g/mol]
- **Ttrp** *[double ,out]* :: Triple point temperature [K]
- **Tnbpt** *[double ,out]* :: Normal boiling point temperature [K]
- **Tc** *[double ,out]* :: Critical temperature [K]
- **Pc** *[double ,out]* :: Critical pressure [kPa]
- **Dc** *[double ,out]* :: Critical density [mol/L]
- **Zc** *[double ,out]* :: Compressibility factor at critical point [Pc/(Rgas*Tc*Dc)]
- **acf** *[double ,out]* :: Acentric factor [-]
- **dip** *[double ,out]* :: Dipole moment [debye]
- **Rgas** *[double ,out]* :: Gas constant [J/mol-K]

subroutine JICRVdll (*D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Calculate the temperature along the Joule-Inversion curve for the input density. This line starts at zero density at the temperature where B is at a maximum, and passes into the liquid phase without crossing the two-phase. It ends at very high pressures. The argument z in this routine is an array with the mole fractions of the mixture. If the input T is non-zero, it is used as the initial guess.

JI is equal to d(Z)/d(T) at constant D del*d^2(alphar)/d(del)/d(T) -del*tau*d^2(alphar)/d(del)/d(tau)/T (can ignore the /T for finding JI=0)

d(JI)/dT -> tau2*del*d^3(alphar)/d(del)/d(tau)**2/T**2** (One of the /T must be removed to match the one removed in the function.)

Parameters

- **D** *[double ,in]* :: Density [mol/l]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Iteration failed to converge

subroutine JTCRVdll (*D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Calculate the temperature along the Joule-Thomson curve for the input density. This line starts at zero density at the temperature where the Joule-Thomson property (dH/dT) is zero, and passes into the liquid phase without crossing the two-phase. It ends at a saturated liquid state far from the critical point. The argument z in this routine is an array with the mole fractions of the mixture. If the input T is non-zero, it is used as the initial guess.

Only the top part in the calculation of hjt is required, the other parts do not go to zero and thus do not contribute to finding JT=0.

Parameters

- **D** *[double ,in]* :: Density [mol/l]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Iteration failed to converge

subroutine LIMITKdll (*htyp*, *icomp*, *T*, *D*, *P*, *Tmin*, *Tmax*, *Dmax*, *Pmax*, *ierr*, *herr*, *htyp_length*, *herr_length*)

This function is deprecated. Use subroutine LIMITX instead.

Parameters

- **htyp** *[char]* :: XXXXXXXXXX
- **icomp** *[int]* :: XXXXXXXXXX
- **T** *[double]* :: XXXXXXXXXX
- **D** *[double]* :: XXXXXXXXXX
- **P** *[double]* :: XXXXXXXXXX
- **Tmin** *[double]* :: XXXXXXXXXX
- **Tmax** *[double]* :: XXXXXXXXXX
- **Dmax** *[double]* :: XXXXXXXXXX
- **Pmax** *[double]* :: XXXXXXXXXX
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **htyp_length** *[int]* :: length of variable `htyp` (default: 3)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine LIMITSdll (*htyp*, *z*, *Tmin*, *Tmax*, *Dmax*, *Pmax*, *htyp_length*)

Returns limits of a property model as a function of composition. Pure fluid limits were read in from the *.fld files; for mixtures, a simple mole fraction weighting in reduced variables is used.

Parameters

- **htyp** *[char ,in]* :: Flag indicating which models are to be checked (character*3) 'EOS' - Equation of state for thermodynamic properties 'ETA' - Viscosity 'TCX' - Thermal conductivity 'STN' - Surface tension
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **Tmin** *[double ,out]* :: Minimum temperature for model specified by htyp [K]
- **Tmax** *[double ,out]* :: Maximum temperature [K]
- **Dmax** *[double ,out]* :: Maximum density [mol/L]
- **Pmax** *[double ,out]* :: Maximum pressure [kPa]
- **htyp_length** *[int]* :: length of variable `htyp` (default: 3)

subroutine LIMITXdll (*htyp*, *T*, *D*, *P*, *z*, *Tmin*, *Tmax*, *Dmax*, *Pmax*, *ierr*, *herr*, *htyp_length*, *herr_length*)

Returns limits of a property model as a function of composition and/or checks inputs T, D, and P against those limits.

Pure fluid limits are read in from the *.fld files; for mixtures, a simple mole fraction weighting of the reduced variables is used.

Attempting calculations below the minimum temperature and/or above the maximum density may result in an error. These will often correspond to a physically unreasonable state; also many equations of state do not extrapolate reliably to lower T's and higher D's.

A warning is issued if the temperature is above the maximum but below 1.5 times the maximum. Pressures up to twice the maximum result in only a warning. Most equations of state may be extrapolated to higher T's and P's. Temperatures and/or pressures outside these extended limits will result in an error.

When calling with an unknown temperature, set T to -1 to avoid performing the melting line check. If inputs are not available, use T=300, P=0, and D=0.

If multiple inputs are outside limits, ierr=SUM(ABS(ierr)), with a positive sign if any error greater than zero (calculations not possible), or a negative sign for warnings only.

Parameters

- **htyp** *[char ,in]* :: Flag indicating the model to check (character*3)
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **Tmin** *[double ,out]* :: Minimum temperature for model specified by htyp [K]
- **Tmax** *[double ,out]* :: Maximum temperature [K]
- **Dmax** *[double ,out]* :: Maximum density [mol/L]
- **Pmax** *[double ,out]* :: Maximum pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **htyp_length** *[int]* :: length of variable `htyp` (default: 3)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `htyp` flags

'EOS' Equation of state

'ETA' Viscosity

'TCX' Thermal conductivity

'STN' Surface tension

`ierr` flags

0 All inputs within limits

-1 1.5*Tmax > T > Tmax

1 T < Tmin or T > 1.5*Tmax

2 D > Dmax or D < 0

-4 2*Pmax > P > Pmax

4 P < 0 or P > 2*Pmax

8 Component composition < 0 or > 1 and/or composition sum <> 1

16 P>Pmelt

-16 T<Ttrp (important for water)

subroutine LIQSPNDLdll (*T*, *z*, *D*, *ierr*, *herr*, *herr_length*)

Find the liquid spinodal density for a given temperature. If no spinodal exists, return the point of zero curvature. This only happens with a few of the older equations, these being argon, ethane, nitrogen, R22, and R124.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)

- **D** *[double ,out]* :: Density at liquid spinodal [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

121 T>Tc

633 Failed to converge

-638 Spinodal not found, point of zero curvature returned

subroutine MASSFLUXdll (*Tm*, *P*, *z*, *beta*, *rf*, *fluxm*, *Cs*, *T0*, *P0*, *xMach*, *u*, *Ts*, *Ps*, *ierr*, *herr*, *herr_length*)

Calculate the theoretical mass flux for a CFV (critical flow venturi) of a gas. This is required for high beta; CSTAR can be used for low beta.

Parameters

- **Tm** *[double ,in]* :: Measured temperature [K]
- **P** *[double ,in]* :: Upstream (static) pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **beta** *[double ,in]* :: Ratio of throat diameter to pipe diameter [-]
- **rf** *[double ,in]* :: Recovery factor [(Tm-T)/(T0-T)] (T is static temperature, T0 is the stagnation temperature)
- **fluxm** *[double ,out]* :: Theoretical mass flux [kg/(m^2-s)]
- **Cs** *[double ,out]* :: Critical flow factor [-]
- **T0** *[double ,out]* :: Stagnation temperature [K]
- **P0** *[double ,out]* :: Stagnation pressure [kPa]
- **xMach** *[double ,out]* :: Mach number (u/speed of sound) [-]
- **u** *[double ,out]* :: Average axial velocity in approach pipe upstream of the CFV [m/s]
- **Ts** *[double ,out]* :: Temperature at throat [K]
- **Ps** *[double ,out]* :: Pressure at throat [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Iteration failed to converge

subroutine MAXPdll (*z*, *Tm*, *Pm*, *Dm*, *ierr*, *herr*, *herr_length*)

Calculate values at the maximum pressure along the saturation line; these are returned from the call to SATSPLN and apply only to the composition in the z() array sent to SATSPLN.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)

- **Tm** *[double ,out]* :: Temperature [K]
- **Pm** *[double ,out]* :: Pressure [kPa]
- **Dm** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

331 Splines not available for calculation

-362 Maximum pressure not known

subroutine MAXTdll (*z*, *Tm*, *Pm*, *Dm*, *ierr*, *herr*, *herr_length*)

Calculate values at the maximum temperature along the saturation line; these are returned from the call to SATSPLN and apply only to the composition in the z() array sent to SATSPLN.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Tm** *[double ,out]* :: Temperature [K]
- **Pm** *[double ,out]* :: Pressure [kPa]
- **Dm** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

331 Splines not available for calculation

-361 Maximum temperature not known

subroutine MELTKdll (*icomp*, *T*, *P*, *ierr*, *herr*, *herr_length*)

Compute melting line with appropriate core model.

Parameters

- **icomp** *[int ,in]* :: Component i (for water and heavy water, send -icomp to obtain the root with the lower pressure at T<Ttrp)
- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,out]* :: Melting line pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) There are two functional forms for the melting line, labeled in the fluid files as ML1 and ML2: ML1: P=Pred*Pr ML2: P=Pred*Exp(Pr) where: Pr=Sum[Nk*Tr^tk]+Sum[Nk*(Tr-1)^tk]+Sum[Nk*(Log Tr)^tk] Tr=T/Tred In the fluid file, Tred and Pred (the reducing values) are given first, followed by the number of terms in each of the summations, and then followed by the coefficients Nk and exponents tk (one term with Nk and tk listed per line).

- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

1 T<Ttrp

-4 P<Ptrp (for MELTP routine)

501 No equation available

502 Unknown melting line equation

subroutine MELTPdll (*P*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Compute the melting line temperature as a function of pressure and composition.

Parameters

- **P** *[double ,in]* :: Melting line pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

-4 Pressure below triple point pressure

501 No equation available

subroutine MELTTdll (*T*, *z*, *P*, *ierr*, *herr*, *herr_length*)

Compute the melting line pressure as a function of temperature and composition.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P** *[double ,out]* :: Melting line pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

501 No equation available

subroutine MLTH2Odll (*T*, *P1*, *P2*)

Compute melting line of water, see fluid file for reference.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P1** *[double ,out]* :: Higher melting line pressure [kPa]

- **P2** *[double ,out]* :: Lower melting line pressure [kPa] Above 273.16 K, only P1 returns a physical answer. Between 251.165 and 273.16 K, two pressures are returned. If flags of -998 or -999 are sent for the temperature, the value of the lowest temperature possible (251.165 K) is sent back in T, the pressure at that point is sent back in P1, and the density at that point is sent back in P2 if the flag -998 is used.

subroutine NAMEdll (*icomp*, *hnam*, *hn80*, *hcasn*, *hnam_length*, *hn80_length*, *hcasn_length*)

Provides name information for the specified component.

Parameters

- **icomp** *[int ,in]* :: Component number in mixture; 1 for pure fluid
- **hnam** *[char ,out]* :: Component name (character*12) (send icomp+1000 to get the fluid hash)
- **hn80** *[char ,out]* :: Component name - long form (character*80) To return the file name used when SETUP was called (without path), send -icomp. If path is also needed, use PASSCMN. For example: call PASSCMN ('hdir',0,1,0,hfl,i,xx,arr,ierr,herr)
- **hcasn** *[char ,out]* :: ID (Chemical Abstracts Service) number (character*12)
- **hnam_length** *[int]* :: length of variable `hnam` (default: 12)
- **hn80_length** *[int]* :: length of variable `hn80` (default: 80)
- **hcasn_length** *[int]* :: length of variable `hcasn` (default: 12)

subroutine PASSCMNdll (*hvr*, *iset*, *icomp*, *jcomp*, *hstr*, *ilng*, *dbl*, *arr*, *ierr*, *herr*, *hvr_length*, *hstr_length*, *herr_length*)

Get or set values of variables in the common blocks.

Examples (in FORTRAN):

```
call PASSCMN ('txeos',  0,3,0, h,i, tmx,z,   ierr,herr) ! get Tmax of component 3
call PASSCMN ('dxeos',  1,2,0, h,i, dmx,z,   ierr,herr) ! set Dmax of component 2
call PASSCMN ('tz',     0,1,0, h,i, Tc, z,   ierr,herr) ! get reducing␣
↪temperature of component 1
call PASSCMN ('ntermf', 0,1,0, h,nt,v,  z,   ierr,herr) ! get number of terms in␣
↪the Helmholtz equation for component 1
call PASSCMN ('coefhmx',1,1,0, h,i, v,  cf,  ierr,herr) ! set the coefficients in␣
↪the Helmholtz equation for component 1
call PASSCMN ('acp0',   1,5,0, h,i, v,  cp0, ierr,herr) ! set the coefficients in␣
↪the cp0 equation for component 5
call PASSCMN ('fPRkij', 1,1,2, h,i, v,  fpr, ierr,herr) ! set the PR coefficient␣
↪for the 1,2 binary
```

Parameters

- **hvr** *[char ,in]* :: Character string with the common variable's name
- **iset** *[int ,in]* :: Flag to indicate the get/set condition
- **icomp** *[int ,in]* :: Component number
- **jcomp** *[int ,in]* :: Second component number for binary mixture variables
- **hstr** *[char ,out]* :: Input or output for a character string
- **ilng** *[int ,out]* :: Input or output for a long variable
- **dbl** *[double ,out]* :: Input or output for a double precision variable
- **arr** (100) *[double ,out]* :: Input or output for a double precision array

- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hvr_length** *[int]* :: length of variable `hvr` (default: 255)
- **hstr_length** *[int]* :: length of variable `hstr` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iset` flags

0 Get variable value

1 Set variable value

`ierr` flags

0 Successful

113 Inputs out of bounds

601 Variable name not recognized

subroutine PDFL1dll(*P*, *D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature as a function of pressure, density, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **D** *[double ,in]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PDFLSHdll(*P*, *D*, *z*, *T*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given density, pressure, and bulk composition. This routine accepts both single-phase and two-phase states as inputs; for single-phase calculations, the subroutine PDFL1 is faster. (See subroutines ABFLSH or TPDFLSH for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **D** *[double ,in]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]

- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PEFL1dll (*P*, *e*, *z*, *kph*, *T*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature and density as a function of pressure, energy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kph** *[int]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PEFLSHdll (*P*, *e*, *z*, *T*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given pressure, bulk energy, and bulk composition. (See subroutines ABFLSH or PBFLSH for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]

- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PHFL1dll (*P*, *h*, *z*, *kph*, *T*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature and density as a function of pressure, enthalpy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kph** *[int]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PHFLSHdll (*P*, *h*, *z*, *T*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given pressure, bulk enthalpy, and bulk composition. (See subroutines ABFLSH or PBFLSH for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]

- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PHI0dll (*itau*, *idel*, *T*, *D*, *z*, *phi00*)

Compute the ideal-gas part of the reduced Helmholtz energy or its derivatives as functions of temperature and density for a mixture.

While the real-gas part of the Helmholtz energy is calculated in terms of dimensionless temperature and density, the ideal- gas part is calculated in terms of absolute temperature and density. (This distinction is necessary for mixtures.)

The Helmholtz energy consists of ideal-gas and residual (real-gas) terms; this routine calculates only the ideal part.

Parameters

- **itau** *[int ,in]* :: Flag specifying the order of the temperature derivative
- **idel** *[int ,in]* :: Flag specifying the order of the density derivative (The density derivatives are not used in the calculation of any property.) when itau = 0 and idel = 0, compute A0/RT when itau = 1 and idel = 0, compute 1st temperature derivative when itau = 2 and idel = 0, compute 2nd temperature derivative when itau = 0 and idel = 1, compute 1st density derivative (actually the derivatives are with respect to the dimensionless quantities tau and del)
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **phi00** *[double ,out]* :: Ideal-gas part of the reduced Helmholtz energy (A/RT); derivatives (as specified by itau and idel) are multiplied by the corresponding power of tau or del; i.e., when itau = 1, the quantity returned is tau*[d(PHI0)/d(tau)] when itau = 2, the quantity returned is tau^2*[d^2(PHI0)/d(tau)^2] when itau = 3, the quantity returned is tau^3*d^3(ph0cpp)/d(tau)^3 where tau=Tc/T and del=D/Dc are evaluated for each component. Similarly, the del derivatives (as specified by idel) are multiplied by the corresponding power of del (the derivatives usually appear with this factor and this approach neatly avoids a possible divide by zero).

subroutine PHIDERVdll (*iderv*, *T*, *D*, *z*, *dadn*, *dnadn*, *ierr*, *herr*, *herr_length*)

Calculate various derivatives required in the calculation of VLE for mixtures. Most of these are based on equations in the GERG-2004 document for natural gas, and are given below on the lines where the code corresponds directly to the equation in that document.

Only the partials of alpha or alpha*n with respect to mole number are returned here. All others are stored in the PHIDR common block for access by subroutine SATGV.

Parameters

- **iderv** *[int ,in]* :: Set to 1 for first order derivatives only (dadn and dnadn) Set to 2 for full calculations
- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Density [mol/L]

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **dadn** (20) *[double ,out]* :: n*partial(alphar)/partial(ni) Eq. 7.16
- **dnadn** (20) *[double ,out]* :: partial(n*alphar)/partial(ni) Eq. 7.15
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) The outputs below are stored in the PHIDR common block, and can be obtained by a call to PASSCMN. :text:
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful Error numbers are not set here, but are returned from either the PHIDERVPR (when Peng-Robinson is active) or RDXHMX routines.

subroutine PHIHMXdll (*itau*, *idel*, *tau*, *delta*, *z*, *phi*)

Compute reduced Helmholtz energy or its derivative as functions of dimensionless temperature and density for the mixture Helmholtz equation of state.

See notes in function PHIMIX.

Parameters

- **itau** *[int ,in]* :: Flag specifying the order of the temperature derivative
- **idel** *[int ,in]* :: Flag specifying the order of the density derivative when itau = 0 and idel = 0, compute A/RT when itau = 0 and idel = 1, compute 1st density derivative when itau = 1 and idel = 1, compute cross derivative etc.
- **tau** *[double ,in]* :: Dimensionless temperature (Tr/T)
- **delta** *[double ,in]* :: Dimensionless density (D/Dr)
- **z** (20) *[double ,in]* :: Composition array (mole fractions)
- **phi** *[double ,out]* :: Residual (real-gas) part of the Helmholtz energy, or one of its derivatives (as specified by itau and idel), in reduced form (A/RT)

subroutine PHIKdll (*icomp*, *itau*, *idel*, *tau*, *delta*, *phi*)

Compute reduced Helmholtz energy or a derivative as functions of dimensionless temperature and density.

The Helmholtz energy consists of ideal and residual (real-gas) terms; this routine calculates only the residual part.

This function computes pure component properties only; call PHIX instead for mixtures.

The reducing parameters Tr and Dr are often, but not necessarily, equal to the critical temperature and density for pure fluids.

Parameters

- **icomp** *[int ,in]* :: Pointer specifying component (1..nc)
- **itau** *[int ,in]* :: Flag specifying the order of the temperature derivative
- **idel** *[int ,in]* :: Flag specifying the order of the density derivative When itau = 0 and idel = 0, compute A/RT. When itau = 0 and idel = 1, compute 1st density derivative. When itau = 1 and idel = 1, compute cross derivative. etc.
- **tau** *[double ,in]* :: Dimensionless temperature (Tr/T)
- **delta** *[double ,in]* :: Dimensionless density (D/Dr)

- **phi** *[double ,out]* :: Residual (real-gas) part of the Helmholtz energy, or one of its derivatives (as specified by itau and idel), in reduced form (A/RT)

subroutine PHIMIXdll (*i*, *j*, *itau*, *idel*, *tau*, *delta*, *z*, *phi*)

Compute reduced Helmholtz energy of mixing (or its derivatives) for the binary interaction of components i and j as a function of composition and dimensionless temperature and density for the mixture Helmholtz equation of state.

The Helmholtz energy consists of ideal-gas and residual (real-gas) terms. The residual term consists of ideal-solution and mixing terms. This routine calculates only the residual term.

Parameters

- **i** *[int]* :: XXXXXXXXXX
- **j** *[int]* :: XXXXXXXXXX
- **itau** *[int ,in]* :: Flag specifying the order of the temperature derivative
- **idel** *[int ,in]* :: Flag specifying the order of the density derivative when itau = 0 and idel = 0, compute Amix/RT when itau = 0 and idel = 1, compute 1st density derivative when itau = 1 and idel = 1, compute cross derivative etc.
- **tau** *[double ,in]* :: Dimensionless temperature (Tr/T)
- **delta** *[double ,in]* :: Dimensionless density (D/Dr)
- **z** (20) *[double ,in]* :: Composition array (mole fractions)
- **phi** *[double ,out]* :: Mixture interaction (excess) part of the Helmholtz energy, or one of its derivatives (as specified by itau and idel), in reduced form (Amix/RT)

subroutine PHIXdll (*itau*, *idel*, *tau*, *delta*, *z*, *phixx*)

Compute reduced Helmholtz energy or a derivative as functions of dimensionless temperature and density by calling the appropriate mixture model.

The Helmholtz energy consists of ideal-gas and residual (real- gas) terms. The residual term consists of ideal-solution and mixing terms. This routine calculates only the residual term.

Parameters

- **itau** *[int ,in]* :: Flag specifying the order of the temperature derivative
- **idel** *[int ,in]* :: Flag specifying the order of the density derivative When itau = 0 and idel = 0, compute A/RT. When itau = 0 and idel = 1, compute 1st density derivative. When itau = 1 and idel = 1, compute cross derivative. etc.
- **tau** *[double ,in]* :: Dimensionless temperature (Tr/T)
- **delta** *[double ,in]* :: Dimensionless density (D/Dr)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **phixx** *[double ,out]* :: Residual (real-gas) part of the Helmholtz energy, or one of its derivatives (as specified by itau and idel), in reduced form (A/RT)

subroutine PQFLSHdll (*P*, *q*, *z*, *kq*, *T*, *D*, *Dl*, *Dv*, *x*, *y*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given pressure, quality, and bulk composition. This routine accepts saturation or two-phase states as inputs. (See subroutines ABFLSH or AQFLSH for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **q** *[double ,in]* :: Vapor quality [mol/mol]

- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kq** *[int]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PREOSdll (*i*)

Turn on or off the use of the PR cubic equation. Should be called after calling SETUP.

Parameters **i** *[int ,in]* :: Flag specifying use of PR

Flags `i` flags

0 Use full equation of state (Peng-Robinson off)

1 Use full equation of state with Peng-Robinson for sat. conditions (not currently working)

2 Use Peng-Robinson equation for all calculations

3 Peng-Robinson with translation term deactivated if i=-1, then i is returned with the current status of the PR EOS. A value of zero indicates that it is not in use. When in use, a 2 or 3 will be returned, depending on which option above was previously selected.

subroutine PRESSdll (*T*, *D*, *z*, *P*)

Compute pressure as a function of temperature, density, and composition. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P** *[double ,out]* :: Pressure [kPa]

subroutine PSATKdll (*icomp*, *T*, *P*, *ierr*, *herr*, *herr_length*)

Compute pure fluid vapor or liquid pressures with the appropriate ancillary equation.

Parameters

- **icomp** *[int ,in]* :: Component i. For liquid pressure equations (pseudo-pure fluids only), send -i.
- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,out]* :: Vapor or liquid pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

121 Temperature greater than critical point temperature

subroutine PSFL1dll (*P*, *s*, *z*, *kph*, *T*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase temperature and density as a function of pressure, entropy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kph** *[int]* :: XXXXXXXXXX
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PSFLSHdll (*P*, *s*, *z*, *T*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given pressure, bulk entropy, and bulk composition. (See subroutines ABFLSH or PBFLSH for the description of all variables.)

Parameters

- **P** *[double ,in]* :: Pressure [kPa]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]

- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine PUREFLDdll (*icomp*)

Change the standard mixture setup so that the properties of one fluid can be calculated as if SETUP had been called for a pure fluid. Calling this routine will disable all mixture calculations. To reset the mixture setup, call this routine with icomp=0.

Parameters **icomp** *[int ,in]* :: fluid number in a mixture to use as a pure fluid (set to zero to reset)

subroutine QMASSdll (*qmol*, *xl*, *xv*, *qkg*, *xlkg*, *xvkg*, *wliq*, *wvap*, *ierr*, *herr*, *herr_length*)

Converts quality and composition on a molar basis to a mass basis.

Parameters

- **qmol** *[double ,in]* :: Molar quality (moles of vapor/total moles) qmol = 0 indicates saturated liquid qmol = 1 indicates saturated vapor 0 < qmol < 1 indicates a two-phase state qmol < 0 or qmol > 1 are not allowed and will result in warning
- **xl** (20) *[double ,in]* :: Composition of liquid phase (array of mole fractions)
- **xv** (20) *[double ,in]* :: Composition of vapor phase (array of mole fractions)
- **qkg** *[double ,out]* :: Quality on mass basis (mass of vapor/total mass)
- **xlkg** (20) *[double ,out]* :: Mass composition of liquid phase (array of mass fractions)
- **xvkg** (20) *[double ,out]* :: Mass composition of vapor phase (array of mass fractions)
- **wliq** *[double ,out]* :: Molar mass of liquid phase [g/mol]
- **wvap** *[double ,out]* :: Molar mass of vapor phase [g/mol]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 All inputs within limits

-19 Input q < 0 or > 1

subroutine QMOLEdll (*qkg*, *xlkg*, *xvkg*, *qmol*, *xl*, *xv*, *wliq*, *wvap*, *ierr*, *herr*, *herr_length*)

Converts quality and composition on a mass basis to a molar basis.

Parameters

- **qkg** *[double ,in]* :: Quality on mass basis (mass of vapor/total mass) qkg = 0 indicates saturated liquid qkg = 1 indicates saturated vapor 0 < qkg < 1 indicates a two-phase state qkg < 0 or qkg > 1 are not allowed and will result in warning
- **xlkg** (20) *[double ,in]* :: Mass composition of liquid phase (array of mass fractions)

- **xvkg** (20) *[double ,in]* :: Mass composition of vapor phase (array of mass fractions)
- **qmol** *[double ,out]* :: Quality on molar basis (moles of vapor/total mass)
- **xl** (20) *[double ,out]* :: Molar composition of liquid phase (array of mole fractions)
- **xv** (20) *[double ,out]* :: Molar composition of vapor phase (array of mole fractions)
- **wliq** *[double ,out]* :: Molar mass of liquid phase [g/mol]
- **wvap** *[double ,out]* :: Molar mass of vapor phase [g/mol]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 All inputs within limits

-19 Input q < 0 or > 1

subroutine RDXHMXdll (*ix*, *icmp*, *icmp2*, *z*, *Tred*, *Dred*, *ierr*, *herr*, *herr_length*)

Returns reducing parameters and their derivatives associated with the mixture Helmholtz EOS; these are used to calculate the 'tau' and 'del' that are the independent variables in the EOS.

Parameters

- **ix** *[int ,in]* :: Flag specifying the order of the composition derivative to calculate, when ix = 0, compute T(red) and D(red) for icmp2=0 when ix = 1, compute 1st derivative with respect to z(icmp) or z(icmp2) when ix = 2, compute 2nd derivative with respect to z(icmp) or z(icmp2) for icmp<>0 and icmp2<>0 when ix = 11, compute cross derivative with respect to z(icmp) and z(icmp2)
- **icmp** *[int ,in]* :: Component number for which derivative will be calculated
- **icmp2** *[int ,in]* :: Second component number for which derivative will be calculated
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **Tred** *[double ,out]* :: Reducing temperature [K] or derivative
- **Dred** *[double ,out]* :: Reducing molar density [mol/L] or derivative of reducing volume [L/mol] (ix=0 - Dc; ix=1 - dVc/dxi; ix=2 - d^2Vc/dxi^2; ix=11 - d^2Vc/dxidxj)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

301 Mixing rule not found for i,j

191 Derivative not available

subroutine REDXdll (*z*, *Tred*, *Dred*)

Returns the reducing parameters associated with mixture EOS; used to calculate the 'tau' and 'del', which are the independent variables in the EOS.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)

- **Tred** *[double ,out]* :: Reducing temperature [K]
- **Dred** *[double ,out]* :: Reducing molar density [mol/L]

subroutine RESIDUALdll (*T*, *D*, *z*, *Pr*, *er*, *hr*, *sr*, *Cvr*, *Cpr*, *ar*, *gr*)

Compute the residual quantities as a function of temperature, density, and composition (where the residual is the total property minus the ideal gas portion).

This routine is the same as THERM2, except it only calculates the residual portions at any temperature and density.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Pr** *[double ,out]* :: Residual pressure [kPa] (P-D*Rxgas*T)
- **er** *[double ,out]* :: Residual internal energy [J/mol]
- **hr** *[double ,out]* :: Residual enthalpy [J/mol]
- **sr** *[double ,out]* :: Residual entropy [J/mol-K]
- **Cvr** *[double ,out]* :: Residual isochoric heat capacity [J/mol-K]
- **Cpr** *[double ,out]* :: Residual isobaric heat capacity [J/mol-K]
- **ar** *[double ,out]* :: Residual Helmholtz energy [J/mol]
- **gr** *[double ,out]* :: Residual Gibbs free energy [J/mol]

subroutine RIEMdll (*T*, *D*, *z*, *riemc*)

RIEM is the thermodynamic curvature in cubic nanometers/molecule. It has the magnitude of the correlation volume, is negative for attractive interactions, and positive for repulsive interactions, except when its magnitude gets smaller than the molecular volume.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **riemc** *[double ,out]* :: RIEM [cubic nanometers/molecule]

subroutine RMIX2dll (*z*, *Rgas*)

Mimic RMIX but return the gas constant as a parameter for use in the DLL.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Rgas** *[double]* :: XXXXXXXXXX

subroutine SATDdll (*D*, *z*, *kph*, *kr*, *T*, *P*, *Dl*, *Dv*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Iterate for temperature and pressure given density along the saturation boundary (including the sublimation and melting lines) and the composition.

Either (Dl,x) or (Dv,y) will correspond to the input state with the other pair corresponding to the other phase in equilibrium with the input state.

The flag kph is for use only with water at densities near the triple point (between 0 and 4 C).

Parameters

- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Flag specifying desired root for multi-valued inputs (typically only water)
- **kr** *[int ,out]* :: Phase flag
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of saturated liquid [mol/L]
- **Dv** *[double ,out]* :: Molar density of saturated vapor [mol/L]
- **x** (20) *[double ,out]* :: Liquid phase composition (array of mole fractions)
- **y** (20) *[double ,out]* :: Vapor phase composition (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

0,1 Return upper root

-1 Return middle root

3 Return melting line

`kr` flags

1 Input state is liquid in equilibrium with vapor.

2 Input state is vapor in equilibrium with liquid.

3 Input state is liquid in equilibrium with solid. (only for pure fluids)

4 Input state is vapor in equilibrium with solid. (only for pure fluids)

`ierr` flags

0 Successful

2 D>Dtrp of the liquid

3 D<Dtrp of the vapor

160 SATD did not converge (See subroutine LIMITX for other possible error numbers.)

`subroutine SATESTdll` (*iFlash*, *T*, *P*, *z*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Estimate temperature, pressure, and compositions to be used as initial guesses to SATTP.

Parameters

- **iFlash** *[int ,in]* :: Phase flag
- **T** *[double ,out]* :: Temperature [K] (input or output)
- **P** *[double ,out]* :: Pressure [kPa] (input or output)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions) The composition for the known x or y array should be sent in this z array, not in the output arrays shown below.
- **x** (20) *[double ,out]* :: Liquid phase composition (array of mole fractions)

- **y** (20) *[double ,out]* :: Vapor phase composition (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iflash` flags

0 Flash calculation (T and P known)

1 T and x known, P and y returned

2 T and y known, P and x returned

3 P and x known, T and y returned

4 P and y known, T and x returned If this value is negative, the retrograde point will be returned.

`ierr` flags

0 Successful

999 Unsuccessful

subroutine SATEdll (*e*, *z*, *kph*, *nroot*, *k1*, *T1*, *P1*, *D1*, *k2*, *T2*, *P2*, *D2*, *ierr*, *herr*, *herr_length*)

Iterate for temperature, pressure, and density given energy along the saturation boundary and the composition.

Parameters

- **e** *[double ,in]* :: Molar energy [J/mol]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Flag specifying desired root
- **nroot** *[int]* :: XXXXXXXXXX
- **k1** *[int]* :: XXXXXXXXXX
- **T1** *[double]* :: XXXXXXXXXX
- **P1** *[double]* :: XXXXXXXXXX
- **D1** *[double]* :: XXXXXXXXXX
- **k2** *[int]* :: XXXXXXXXXX
- **T2** *[double]* :: XXXXXXXXXX
- **P2** *[double]* :: XXXXXXXXXX
- **D2** *[double]* :: XXXXXXXXXX
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

0 Return all roots along the liquid-vapor line

1 Return only the liquid VLE root

2 Return only the vapor VLE roots

3 Return liquid SLE root (melting line)

4 Return vapor SVE root (sublimation line)

subroutine SATGUESSdll (*kph*, *iprop*, *x*, *T*, *P*, *D*, *h*, *s*, *Dy*, *y*, *ierr*, *herr*, *herr_length*)

For a pure fluid, call the ancillary equations to obtain close estimates for the saturation boundaries. The difference between these values and those from SATT, SATP, etc., depend on how well the ancillary equation was fitted, but generally they are within 0.1%, except for the saturation densities within several degrees of the critical temperature.

For a mixture, calculate approximate values from the spline curves for the saturation boundary. Subroutine SATSPLN must be called in order for this to work.

The input property should be placed in the corresponding variable for T, P, D, h, or s. Inputs of h and s only work for mixtures when SATSPLN has been called.

Parameters

- **kph** *[int ,in]* :: Input phase; 1-liquid, 2-vapor When maximum in the property does not occur near the critical point, then kph=1 returns the root at the higher density and kph=2 returns the root at the lower density.
- **iprop** *[int ,in]* :: Input property
- **x** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **h** *[double ,out]* :: Enthalpy or energy [J/mol] (not returned for a pure fluid)
- **s** *[double ,out]* :: Entropy [J/mol-K] (not returned for a pure fluid)
- **Dy** *[double ,out]* :: Equilibrium phase density [mol/L]
- **y** (20) *[double ,out]* :: Equilibrium phase composition (array of mole fractions) (h, s, and y are only returned when splines are used to calculate values.)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iprop` flags

1 Temperature

2 Pressure

3 Density

4 Enthalpy

5 Entropy

6 Energy (currently only working for pure fluids)

11 to 15 1st derivative of the property (for iprop-10) returned in Dy with respect to density (only for iFlag=0)

21 to 25 2nd derivative of the property (for iprop-20) returned in Dy with respect to density (only for iFlag=0)

101 Check if the x array is identical to those sent to SATSPLN (only for iFlag=0) (for negative values of 1 to 5, find the location of zero slope of the property with respect to D) (only for iFlag=0)

`iflag` flags

0 Use default and best methods, generally ancillary equations for pure fluids and splines (calculated from call to SATSPLN) for mixtures

11 Use Rackett technique to get density from T and P (value of kph and iprop ignored)

12 Use initial guess equations of Lemmon

`ierr` flags

0 Successful

331 Splines not available for saturation calculations

332 Initialize variable d72l first before calling SATGUESS

-311 Compositions not identical to that used in the call to SATSPLN

subroutine SATGVdll (*T*, *P*, *z*, *vf*, *b*, *ipv*, *ityp*, *isp*, *Dx*, *Dy*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Calculates the bubble or dew point state with the entropy or density method of GV. The calculation method is similar to the volume based algorithm of GERG. The cricondenbar and cricondentherm are estimated with the method in Michelsen, Saturation Point Calculations, Fluid Phase Equilibria, 23:181, 1985.

Equations to be solved simultaneously are

Pressure based:

- f(1:n) - LOG(y/x)-LOG((fxi/nxi)/(fyi/nyi))=0
- f(n+1) - SUM(y(i)-x(i))=0
- f(n+2) - b/binput-1=0, where b = P, T, D, or s

Volume based:

- f(1:n) - LOG(y/x)-LOG((fxi/nxi)/(fyi/nyi))=0
- f(n+1) - SUM(y(i)-x(i))=0
- f(n+2) - py=px
- f(n+3) - b/binput-1=0, where b = P, T, D, or s

Variables:

- 1 to nc - LOG(k(i))
- nc+1 - LOG(T)
- nc+2 - LOG(P) or LOG(Dx)
- nc+3 - LOG(Dy)

Parameters

- **T** *[double ,out]* :: Temperature [K]
- **P** *[double ,out]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Overall composition (array of mole fractions)
- **vf** *[double ,in]* :: Vapor fraction (0>=vf>=1; the input value of density can be in either state and does not affect the outputs in Dx, Dy, x, and y)

- **b** *[double ,in]* :: Input value, either entropy [J/mol-K] or density [mol/L]
- **ipv** *[int ,in]* :: Pressure or volume based algorithm
- **ityp** *[int ,in]* :: Input values
- **isp** *[int ,in]* :: Use values from Splines as initial guesses if set to 1
- **Dx** *[double ,out]* :: Density of x phase [mol/L]
- **Dy** *[double ,out]* :: Density of y phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the x array (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the y array (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `vf` flags

vf=0,ityp=0,1 Dew phase inputs, state in equilibrium returned in Dy and y

vf=1,ityp=0,1 Liquid phase inputs, state in equilibrium returned in Dx and x

vf=0,ityp=6 Inputs are returned in Dx and the x array Outputs are returned in Dy and the y array

vf=1,ityp=6 Inputs are returned in Dy and the y array Outputs are returned in Dx and the x array

`ipv` flags

1 Pressure based

2 Volume based

`ityp` flags

0 Given P, calculate T

1 Given T, calculate P

2 Cricondentherm condition, calculate T,P (ipv=1 only)

3 Cricondenbar condition, calculate T,P (ipv=1 only)

5 Given entropy, calculate T,P

6 Given density, calculate T,P

`ierr` flags

0 Successful

2 Input D<=0

151 No convergence

172 vf<0 or vf>1

191 Derivatives are not available in PR or RDXHMX

321 Trivial solution

200 Density out of range (See subroutine LIMITX for other possible error numbers.)

subroutine SATHdll (*h*, *z*, *kph*, *nroot*, *k1*, *T1*, *P1*, *D1*, *k2*, *T2*, *P2*, *D2*, *ierr*, *herr*, *herr_length*)

Iterate for temperature, pressure, and density given enthalpy along the saturation boundary and the composition.

The second root is always set as the root in the vapor at temperatures below the maximum enthalpy on the vapor saturation line. If kph is set to 2, and only one root is found in the vapor (this occurs when h<hcrit) the state point will be placed in k2,T2,P2,D2. If kph=0 and this situation occurred, the first root (k1,T1,P1,d1) would be in the liquid (k1=1, k2=2).

Parameters

- **h** *[double ,in]* :: Molar enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Flag specifying desired root
- **nroot** *[int ,out]* :: Number of roots. Value is set to one for kph=1,3,4 if ierr=0
- **k1** *[int ,out]* :: Phase of first root (1-liquid, 2-vapor, 3-melt, 4-subl) For mixtures where the saturation line goes infinite in pressure, k1=-1 if h>hmax (where hmax was found in the vapor phase)
- **T1** *[double ,out]* :: Temperature of first root [K]
- **P1** *[double ,out]* :: Pressure of first root [kPa]
- **D1** *[double ,out]* :: Molar density of first root [mol/L]
- **k2** *[int ,out]* :: Phase of second root (1-liquid, 2-vapor, 3-melt, 4-subl)
- **T2** *[double ,out]* :: Temperature of second root [K]
- **P2** *[double ,out]* :: Pressure of second root [kPa]
- **D2** *[double ,out]* :: Molar density of second root [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

0 Return all roots along the liquid-vapor line

1 Return only liquid VLE root

2 Return only vapor VLE roots

3 Return liquid SLE root (melting line)

4 Return vapor SVE root (sublimation line) kph = 3,4 presently working only for pure components

`ierr` flags

0 Successful

181 SATH did not converge for one of the roots

54 h < hmin

55 h > hmax

56 h > htrp (for sublimation inputs) (See subroutine LIMITX for other possible error numbers.)

subroutine SATPdll (*P*, *z*, *kph*, *T*, *Dl*, *Dv*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Iterate for saturated liquid and vapor states given pressure and the composition of one phase.

All variables should be initialized before calling this routine because they will be used as initial values if T<0.

Parameters

- **P** *[double ,in]* :: Pressure [kPa] If T is negative, all other variables are used as initial guesses at ABS(T).
- **z** (20) *[double ,in]* :: Composition (array of mole fractions) (phase specified by kph)
- **kph** *[int ,in]* :: Phase flag
- **T** *[double ,out]* :: Temperature [K]
- **Dl** *[double ,out]* :: Molar density of saturated liquid [mol/L]
- **Dv** *[double ,out]* :: Molar density of saturated vapor [mol/L] For a pseudo pure fluid, the density of the equilibrium phase is not returned. Call SATP twice, once with kph=1 to get Tliq and Dl, and once with kph=2 to get Tvap and Dv.
- **x** (20) *[double ,out]* :: Liquid phase composition (array of mole fractions)
- **y** (20) *[double ,out]* :: Vapor phase composition (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Input z is liquid composition (bubble point)

2 Input z is vapor composition (dew point)

3 Input z is liquid composition (freezing point)

4 Input z is vapor composition (sublimation point)

`ierr` flags

0 Successful

141 P > Pcrit

144 Pure fluid iteration did not converge (See subroutine LIMITX for other possible error numbers.)

subroutine SATSPLNdll (*z*, *ierr*, *herr*, *herr_length*)

Calculates the phase boundary of a mixture at a given composition, along with the critical point, cricondentherm, and cricondenbar.

Parameters

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

355 Saturation routine failed

subroutine SATSdll(*s*, *z*, *kph*, *nroot*, *k1*, *T1*, *P1*, *D1*, *k2*, *T2*, *P2*, *D2*, *k3*, *T3*, *P3*, *D3*, *ierr*, *herr*, *herr_length*)

Iterate for temperature, pressure, and density given entropy along the saturation boundary and the composition.

The second root is always set as the root in the vapor at temperatures below the maximum entropy on the vapor saturation line. If kph is set to 2, and only one root is found in the vapor (this occurs when s<scrit) the state point will be placed in k2,T2,P2,D2. If kph=0 and this situation occurred, the first root (k1,T1,P1,D1) would be in the liquid (k1=1, k2=2).

The third root is the root with the lowest temperature. For fluids with multiple roots, when only one root is found in the vapor phase (this happens only at very low temperatures past the region where three roots are located), the value of the root is still placed in k3,T3,P3,D3. For fluids that never have more than one root (when there is no maximum entropy along the saturated vapor line), the value of the root is always placed in k1,T1,P1,D1.

Parameters

- **s** *[double ,in]* :: Molar entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Flag specifying desired root
- **nroot** *[int ,out]* :: Number of roots. Set to one for kph=1,3,4 if ierr=0
- **k1** *[int ,out]* :: Phase of first root (1-liquid, 2-vapor, 3-melt, 4-subl)
- **T1** *[double ,out]* :: Temperature of first root [K]
- **P1** *[double ,out]* :: Pressure of first root [kPa]
- **D1** *[double ,out]* :: Molar density of first root [mol/L]
- **k2** *[int ,out]* :: Phase of second root (1-liquid, 2-vapor, 3-melt, 4-subl)
- **T2** *[double ,out]* :: Temperature of second root [K]
- **P2** *[double ,out]* :: Pressure of second root [kPa]
- **D2** *[double ,out]* :: Molar density of second root [mol/L]
- **k3** *[int ,out]* :: Phase of third root (1-liquid, 2-vapor, 3-melt, 4-subl)
- **T3** *[double ,out]* :: Temperature of third root [K]
- **P3** *[double ,out]* :: Pressure of third root [kPa]
- **D3** *[double ,out]* :: Molar density of third root [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

0 Return all roots along the liquid-vapor line

1 Return only liquid VLE root

2 Return only vapor VLE roots

3 Return liquid SLE root (melting line)

4 Return vapor SVE root (sublimation line) kph = 3,4 presently working only for pure components

`ierr` flags

0 Successful

192 SATS did not converge for one or more roots

66 s < smin

67 s > smax

68 s > strp (for sublimation inputs) (See subroutine LIMITX for other possible error numbers.)

subroutine SATTPdll (*T*, *P*, *z*, *iFlsh*, *iGuess*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *ierr*, *herr*, *herr_length*)

Calculate saturation properties for bubble, dew, or 2-phase states with the use of analytical derivatives of the Helmholtz energy with respect to composition.

Parameters

- **T** *[double ,in]* :: Temperature [K] (input or output)
- **P** *[double ,in]* :: Pressure [kPa] (input or output)
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **iFlsh** *[int ,in]* :: Phase flag
- **iGuess** *[int ,in]* :: If set to 1, the parameters Dl, Dv, x, and y are used as initial guesses for the calculation. If Dl and Dv are set to zero when iGuess=1, then the densities are obtained from the first call to TPRHO. If Dl and Dv are not zero when iGuess=1, those values are used as initial values. If set to 2 and splines have been calculated, use inputs rather than spline values.
- **D** *[double ,out]* :: Overall density [mol/L]
- **Dl** *[double ,out]* :: Molar density of saturated liquid [mol/L]
- **Dv** *[double ,out]* :: Molar density of saturated vapor [mol/L]
- **x** (20) *[double ,out]* :: Liquid phase composition (array of mole fractions)
- **y** (20) *[double ,out]* :: Vapor phase composition (array of mole fractions)
- **q** *[double ,out]* :: Quality
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `iflsh` flags

0 Flash calculation (T and P known)

1 T and x known, P and y returned

2 T and y known, P and x returned

3 P and x known, T and y returned

4 P and y known, T and x returned If this value is negative, the retrograde point will be returned.

`ierr` flags

0 Successful

121 T>Tmax (maxcondentherm)

141 P>Pmax (maxcondenbar)

151 Iteration failed

156 Probable Type III mixture with no liquid solution

159 Wrong input value for iFlsh (See subroutine LIMITX for other possible error numbers.)

subroutine SATTdll (*T*, *z*, *kph*, *P*, *Dl*, *Dv*, *x*, *y*, *ierr*, *herr*, *herr_length*)

Iterate for saturated liquid and vapor states given temperature and the composition of one phase.

All variables should be initialized before calling this routine because they will be used as initial values if T<0.

Parameters

- **T** *[double ,in]* :: Temperature [K] If T is negative, all other variables are used as initial guesses at ABS(T).
- **z** (20) *[double ,in]* :: Composition (array of mole fractions) (phase specified by kph)
- **kph** *[int ,in]* :: Phase flag
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of saturated liquid [mol/L]
- **Dv** *[double ,out]* :: Molar density of saturated vapor [mol/L] For a pseudo pure fluid, the density of the equilibrium phase is not returned. Call SATT twice, once with kph=1 to get Pliq and Dl, and once with kph=2 to get Pvap and Dv.
- **x** (20) *[double ,out]* :: Liquid phase composition (array of mole fractions)
- **y** (20) *[double ,out]* :: Vapor phase composition (array of mole fractions)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Input z is liquid composition (bubble point)

-1 Force calculation in the liquid phase even if T<Ttrp

2 Input z is vapor composition (dew point)

-2 Force calculation in the vapor phase even if T<Ttrp

3 Input z is liquid composition along the freezing line (melting line)

4 Input z is vapor composition along the sublimation line

`ierr` flags

0 Successful

121 T > Tcrit

124 Pure fluid iteration did not converge (See subroutine LIMITX for other possible error numbers.)

subroutine SETAGAdll (*ierr*, *herr*, *herr_length*)

Set up working arrays for use with AGA8 equation of state.

Parameters

- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

108 Error (e.g. fluid not found)

subroutine SETKTVdll(*icomp*, *jcomp*, *hmodij*, *fij*, *hFmix*, *ierr*, *herr*, *hmodij_length*, *hFmix_length*, *herr_length*)

Set mixture model and/or parameters.

This subroutine must be called after SETUP, but before any call to SETREF (for cases where energy, enthalpy, entropy, Gibbs energy, or the Helmholtz energy are required); it need not be called at all if the default mixture parameters (those read in by SETUP) are to be used.

The component numbers icomp and jcomp must match the order that is found in the HMX.BNC file for each binary pair, or, in the case where no interaction parameters are available in the HMX.BNC file, icomp and jcomp must be in the same order as was used in the call to SETUP. If the numbers in these two integers are backwards, an error number and message will be returned, and nothing will be changed. In this situation, switch the numbers and call this routine again.

Kunz-Wagner model (KW0)	Lemmon-Jacobsen model (LJ6)
fij(1) = betaT	fij(1) = zeta
fij(2) = gammaT	fij(2) = xi
fij(3) = betaV	fij(3) = Fij
fij(4) = gammaV	fij(4) = beta
fij(5) = Fij	fij(5) = gamma
fij(6) = 'not used'	fij(6) = 'not used'

Parameters

- **icomp** *[int ,in]* :: Component i
- **jcomp** *[int ,in]* :: Component j
- **hmodij** *[char ,in]* :: Mixing rule for the binary pair i,j (e.g. LJ6, KW0, XR0, or LIN) (character*3) If hmodij is 'RST', reset all pairs to values from the original call to SETUP (all other inputs are ignored)
- **fij** (6) *[double ,in]* :: Binary mixture parameters (array of dimension nmxpar; currently nmxpar is set to 6) The parameters will vary depending on hmodij (see above)
- **hFmix** *[char ,in]* :: No longer used. Info from previous versions: File name (character*255) containing generalized parameters for the binary mixture model; this will usually be the same as the corresponding input to SETUP (e.g.,'HMX.BNC')
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hmodij_length** *[int]* :: length of variable `hmodij` (default: 3)
- **hFmix_length** *[int]* :: length of variable `hFmix` (default: 255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

111 Error in opening mixture file

902 Routine not value for a pure fluid

903 Illegal i,j specification (i=j or i>nc or j>nc)

904 Order of fluids is backwards from that in HMX.BNC

subroutine SETMIXdll (*hMixNme*, *hFmix*, *hrf*, *ncc*, *hFiles*, *z*, *ierr*, *herr*, *hMixNme_length*, *hFmix_length*, *hrf_length*, *hFiles_length*, *herr_length*)

Open a mixture file (e.g., R410A.mix) and read constituents and mole fractions.

Parameters

- **hMixNme** *[char ,in]* :: Mixture file name (character*255)
- **hFmix** *[char ,in]* :: File containing mixture coefficients (character*255)
- **hrf** *[char ,in]* :: Reference state (character*3); See subroutine SETUP for specifics.
- **ncc** *[int ,out]* :: Number of fluids in mixture
- **hFiles** *[char ,out]* :: Array of file names specifying mixture components that were used to call setup (character*255)
- **z** (20) *[double ,out]* :: Array of mole fractions for the specified mixture
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hMixNme_length** *[int]* :: length of variable `hMixNme` (default: 255)
- **hFmix_length** *[int]* :: length of variable `hFmix` (default: 255)
- **hrf_length** *[int]* :: length of variable `hrf` (default: 3)
- **hFiles_length** *[int]* :: length of variable `hFiles` (default: 10000)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

101 Error in opening file

-102 Mixture file contains mixing parameters

805 Sum of compositions not equal to one

subroutine SETMODdll (*ncomp*, *htype*, *hmix*, *hcomp*, *ierr*, *herr*, *htype_length*, *hmix_length*, *hcomp_length*, *herr_length*)

Set model(s) other than the NIST-recommended ('NBS') ones.

This subroutine must be called before SETUP; it need not be called at all if the default (NIST-recommended) models are desired.

Parameters

- **ncomp** *[int ,in]* :: Number of components (1 for pure fluid) (integer)
- **htype** *[char ,in]* :: Flag indicating which model to set (character*3)
- **hmix** *[char ,in]* :: Mixture model for the property specified in htype (character*3) (ignored if number of components = 1)

- **hcomp** *[char ,in]* :: Component model(s) for property specified in htype [array (1..ncomp) of character*3]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **htype_length** *[int]* :: length of variable `htype` (default: 3)
- **hmix_length** *[int]* :: length of variable `hmix` (default: 3)
- **hcomp_length** *[int]* :: length of variable `hcomp` (default: 60)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `htype` flags

'NBS' Reset all of the above model types to 'NBS' (values of hmix and hcomp are ignored)

'EOS' Equation of state

'ETA' Viscosity

'TCX' Thermal conductivity

'STN' Surface tension

`hmix` flags

'NBS' Use NIST recommendation for specified fluid/mixture

'HMX' Mixture Helmholtz model for thermodynamic properties

'ECS' Extended corresponding states for viscosity or therm. cond.

'STX' Surface tension mixture model

`hcomp` flags

'NBS' NIST recommendation for specified fluid/mixture

'FEQ' Helmholtz energy model

'BWR' Pure fluid modified Benedict-Webb-Rubin (MBWR)

'ECS' Extended corresponding states (all fluids)

'PRT' Peng-Robinson (PRT model from fluid file)

'VS1' The 'composite' model for R134a, R152a, NH3, etc.

'VS2' Younglove-Ely model for hydrocarbons

'VS4' Generalized friction theory of Quinones-Cisneros and Deiters

'VS5' Chung et al. (1988) predictive model

'VS6' Vesovic form of VS1 model

'VS7' Polynomial/exponential model

'TC1' The 'composite' model for R134a, R152a, etc.

'TC2' Younglove-Ely model for hydrocarbons

'TC5' Chung et al. (1988) predictive model

'ST1' Surface tension as f(tau); tau = 1 - T/Tc

`ierr` flags

0 Successful

113 ncomp outside of bounds

subroutine SETNCdll (*ncomp*)

Allow the user to modify the value of nc (the number of components in a mixture) so that a subset of the loaded mixtures can be used. For example, a 4 component mixture could be set up, but nc could be set to 3 to calculate properties of the mixture of the first three components. The last component can then be used as a pure fluid. For example, SETUP could be called to load four fluids, R32.fld, R125.fld, R134a.fld, and R407C.ppf, followed by a call to SETNC(3) so that the 4th component is not used in mixture calculations. The pseudo-pure fluid equation in R407C.ppf can be accessed by calling PUREFLD(4) to get single-phase thermodynamic properties, and VLE states or transport properties that require the first three components can be obtained by calling PUREFLD(0) and setting the composition array z with the appropriate values.

Parameters **ncomp** *[int ,in]* :: number of components in the mixture

subroutine SETREFDIRdll (*hpth*, *hpth_length*)

Set a path to the location of original fluid files so that a user can specify where their fluid file is located, but the reference files needed for transport properties, etc., such as nitrogen.fld, can be found.

Parameters

- **hpth** *[char ,in]* :: Location of the fluid files (character*255) The path does not need to contain the ending “”. For example: hpth=’C:Program FilesRefpropfluids’
- **hpth_length** *[int]* :: length of variable `hpth` (default: 255)

subroutine SETREFdll (*hrf*, *ixflag*, *x0*, *h0*, *s0*, *T0*, *P0*, *ierr*, *herr*, *hrf_length*, *herr_length*)

Set reference state enthalpy and entropy.

This subroutine must be called after SETUP; it need not be called at all if the reference state specified in the call to SETUP is to be used.

Parameters

- **hrf** *[char ,in]* :: Reference state for thermodynamic calculations (character*3)
- **ixflag** *[int ,in]* :: Composition flag
- **x0** (20) *[double ,in]* :: Composition for which h0 and s0 apply; array(1:nc) (array of mole fractions) This is useful for mixtures of a predefined composition, e.g., refrigerant blends such as R410A. Only has meaning if ixflag = 2
- **h0** *[double ,in]* :: Reference state enthalpy at T0, P0, and x0 [J/mol] (only has meaning if hrf = ‘OTH’ or ‘OT0’)
- **s0** *[double ,in]* :: Reference state entropy at T0, P0, and x0 [J/mol-K] (only has meaning if hrf = ‘OTH’ or ‘OT0’)
- **T0** *[double ,in]* :: Reference state temperature [K] (only has meaning if hrf = ‘OTH’ or ‘OT0’) T0 = -1 indicates saturated liquid at normal boiling point (bubble point for a mixture)
- **P0** *[double ,in]* :: Reference state pressure [kPa] (only has meaning if hrf = ‘OTH’ or ‘OT0’) P0 = -1 indicates saturated liquid at T0 (and x0) P0 = -2 indicates saturated vapor at T0 (and x0)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **hrf_length** *[int]* :: length of variable `hrf` (default: 3)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `hrf` flags

'NBP' h,s = 0 at normal boiling point(s)

'ASH' h,s = 0 for saturated liquid at -40 C (ASHRAE convention)

'IIR' h = 200 kJ/kg, s = 1 kJ/kg-K for sat. liquid at 0 C (IIR convention)

'DEF' Default reference state as specified in fluid file

'OTH' Other, as specified by h0, s0, T0, P0 (real gas state)

'OT0' Other, as specified by h0, s0, T0, P0 (ideal-gas state)

'TD0' h,s = 0 at zero temperature and density

'NA' Not applicable, do not set up the reference state. The values of e, h, and s will have a random reference state. Do not use except for EOS testing.

'???' Set hrf to the value of the current reference state and exit

`ixflag` flags

1 Reference state applied to pure components

2 Reference state applied to mixture x0 In order for option 2 to work, each fluid in the mixture must use the same reference state.

`ierr` flags

0 Successful

22 Tmin > Tref for IIR reference state

23 Tcrit < Tref for IIR reference state

24 Tmin > Tref for ASHRAE reference state

25 Tcrit < Tref for ASHRAE reference state

26 Tmin > Tnbp for NBP reference state

-28 Can't apply 'DEF' to mixture; will apply to pure components

-29 Unknown reference state specified; will use 'DEF'

119 Convergence failure in calculating reference state

subroutine SETUPdll (*ncomp*, *hFiles*, *hFmix*, *hrf*, *ierr*, *herr*, *hFiles_length*, *hFmix_length*, *hrf_length*, *herr_length*)

Define models and initialize arrays.

Parameters

- **ncomp** *[int ,in]* :: Number of components (1 for pure fluid) (integer) If called with ncomp=-1, the version number*10000 will be returned in ierr.
- **hFiles** *[char ,in]* :: Array of file names specifying the fluid or the mixture components (character*255); e.g., 'fluidsr134a.fld' (DOS) ':fluids:r134a.fld' (Mac) '[full_path]/fluids/r134a.fld' (UNIX)
- **hFmix** *[char ,in]* :: Name of file containing mixture coefficients (character*255); e.g., 'fluidsHMX.BNC'
- **hrf** *[char ,in]* :: Reference state for thermodynamic calculations (character*3). Other choices are possible, see SETREF
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)

- **hFiles_length** *[int]* :: length of variable `hFiles` (default: 10000)
- **hFmix_length** *[int]* :: length of variable `hFmix` (default: 255)
- **hrf_length** *[int]* :: length of variable `hrf` (default: 3)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `hrf` flags

'DEF' default reference state as specified in fluid file

'NBP' h,s = 0 at pure component normal boiling point

'ASH' h,s = 0 for sat. liquid at -40 C (ASHRAE convention)

'IIR' h = 200 kJ/kg and s = 1 kJ/kg-K for sat. liquid at 0 C (IIR convention)

`ierr` flags

0 Successful

101 Error in opening file

102 Error in file or premature end of file

-107 Unknown model encountered in file

105 Specified model not found

-105 Must use routine SETREF for (OTH) reference state choice

111 Error in opening mixture file

112 Mixture file of wrong type

114 nc not equal to the nc sent to SETMOD

-117 Binary pair not found, all parameters will be estimated

117 Mixture parameters not available, mixture is outside the range of the model and calculations will not be made

subroutine SPLNROOTdll (*isp*, *iderv*, *f*, *a*, *ierr*, *herr*, *herr_length*)

Calculates the root of a given value of a spline function

Parameters

- **isp** *[int ,in]* :: Indicator for which spline to use 1 to nc - Composition nc+1 - Temperature nc+2 - Pressure nc+3 - Density nc+4 - Enthalpy or Energy (depending on the value of ieflag). nc+5 - Entropy
- **iderv** *[int ,in]* :: Values of -1 and -2 return lower and upper root values, a value of 0 returns the spline root, a value of 1 returns the root where the derivative of the spline with respect to D is equal to the the value of f (set f=0 to find maximum or minimum).
- **f** *[double ,in]* :: Value of spline function
- **a** *[double ,out]* :: Root value (initial value required since some splines can be doubled valued)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

151 Routine did not converge.'

subroutine SPLNVALdll (*isp*, *iderv*, *a*, *f*, *ierr*, *herr*, *herr_length*)
Calculates the value of a spline or the derivative of the spline at the specified value.

Parameters

- **isp** *[int ,in]* :: Indicator for which spline to use
- **iderv** *[int ,in]* :: Values of -1 and -2 return lowest and highest density values of the splines, a value of 0 returns spline function value, a value of 1 returns the derivative of the spline with respect to the input value, and a value of 2 returns the 2nd derivative.
- **a** *[double ,in]* :: Input value (molar density of the known phase)
- **f** *[double ,out]* :: Desired output value
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `isp` flags

0 Molar density of the known phase

1 to nc Composition of the incipient phase

nc+1 Temperature

nc+2 Pressure

nc+3 Molar density of the equilibrium phase

nc+4 Enthalpy

nc+5 Entropy

subroutine STNdll (*T*, *Dl*, *Dv*, *x*, *y*, *sigma*, *ierr*, *herr*, *herr_length*)
Compute surface tension with appropriate core model. For mixtures, this routine requires that the saturation densities and vapor compositions be sent as inputs. If these are not available, call SURFT.

The critical temperature used is that of the current equation of state. This may differ slightly from that used in the original correlation of the surface tension; this change is necessary to give proper behavior of surface tension near the critical point and to avoid possible numerical crashes.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **Dl** *[double ,in]* :: Molar density of liquid phase [mol/L]
- **Dv** *[double ,in]* :: Molar density of vapor phase [mol/L]
- **x** (20) *[double ,in]* :: Composition of liquid phase (array of mole fractions)
- **y** (20) *[double ,in]* :: Composition of vapor phase (array of mole fractions)
- **sigma** *[double ,out]* :: Surface tension [N/m]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

1 T > Tcrit

502 Unknown model

151 Failed to converge

subroutine SUBLPdll (*P*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Compute the sublimation line temperature as a function of pressure and composition.

Parameters

- **P** *[double ,in]* :: Sublimation line pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

-4 Pressure above triple point pressure

501 No equation available

subroutine SUBLTdll (*T*, *z*, *P*, *ierr*, *herr*, *herr_length*)

Compute the sublimation line pressure as a function of temperature and composition.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P** *[double ,out]* :: Sublimation line pressure [kPa]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

501 No equation available

subroutine SURFTdll (*T*, *Dl*, *z*, *sigma*, *ierr*, *herr*, *herr_length*)

Compute surface tension as a function of T. SATT is called to obtain the liquid density. If this is already known then your calling routines should use subroutine STN to greatly reduce the time needed in the calculation of the surface tension.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **Dl** *[double ,out]* :: Molar density of liquid phase [mol/L] (only returned for mixtures)
- **z** (20) *[double ,in]* :: Composition of liquid phase (array of mole fractions)

- **sigma** *[double ,out]* :: Surface tension [N/m]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful Other error messages returned from SATT or STN

subroutine SURTENdll (*T*, *Dl*, *Dv*, *x*, *y*, *sigma*, *ierr*, *herr*, *herr_length*)
With version 10 of Refprop, this routine should no longer be used, and STN or SURFT should be used instead.

(See subroutine STN for the description of all variables.)

Parameters

- **T** *[double]* :: XXXXXXXXXX
- **Dl** *[double]* :: XXXXXXXXXX
- **Dv** *[double]* :: XXXXXXXXXX
- **x** (20) *[double]* :: XXXXXXXXXX
- **y** (20) *[double]* :: XXXXXXXXXX
- **sigma** *[double]* :: XXXXXXXXXX
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TDFLSHdll (*T*, *D*, *z*, *P*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)
Flash calculation given temperature, bulk density, and bulk composition. This routine accepts both single-phase and two-phase states as inputs; for single-phase calculations, the subroutine THERM is much faster. (See subroutines ABFLSH or TPDFLSH for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Density [mol/L]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **P** *[double ,out]* :: Pressure [kPa]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]

- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TEFL1dll (*T*, *e*, *z*, *Dmin*, *Dmax*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase density as a function of temperature, energy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **Dmin** *[double ,in]* :: Lower bound on density [mol/L]
- **Dmax** *[double ,in]* :: Upper bound on density [mol/L]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TEFLSHdll (*T*, *e*, *z*, *kr*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, bulk energy, and bulk composition. (See subroutines ABFLSH or TBFLSH for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **e** *[double ,in]* :: Internal energy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kr** *[int]* :: XXXXXXXXXX
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]

- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine THERM0dll(*T*, *D*, *z*, *P0*, *e0*, *h0*, *s0*, *Cv0*, *Cp00*, *w0*, *a0*, *g0*)

Compute ideal-gas thermal quantities as a function of temperature, density, and composition from core functions.

This routine is the same as THERM, except it only calculates ideal gas properties (Z=1) at any temperature and density.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P0** *[double ,out]* :: Pressure [kPa]
- **e0** *[double ,out]* :: Internal energy [J/mol]
- **h0** *[double ,out]* :: Enthalpy [J/mol]
- **s0** *[double ,out]* :: Entropy [J/mol-K]
- **Cv0** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp00** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w0** *[double ,out]* :: Speed of sound [m/s]
- **a0** *[double ,out]* :: Helmholtz energy [J/mol]
- **g0** *[double ,out]* :: Gibbs free energy [J/mol]

subroutine THERM2dll(*T*, *D*, *z*, *P*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *zz*, *hjt*, *a*, *g*, *xkappa*, *beta*, *dPdD*, *d2PdD2*, *dPdT*, *dDdT*, *dDdP*, *d2PdT2*, *d2PdTD*, *spare3*, *spare4*)

Compute thermal quantities as a function of temperature, density, and composition. This routine is the simply the combination of several others. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P** *[double]* :: XXXXXXXXXX
- **e** *[double]* :: XXXXXXXXXX
- **h** *[double]* :: XXXXXXXXXX
- **s** *[double]* :: XXXXXXXXXX
- **Cv** *[double]* :: XXXXXXXXXX
- **Cp** *[double]* :: XXXXXXXXXX
- **w** *[double]* :: XXXXXXXXXX
- **zz** *[double ,out]* :: Compressibility factor (= PV/RT) [-] (See subroutines THERM, THERM3, AG, and DERVPVT for the description of all variables.)

- **hjt** *[double]* :: XXXXXXXXXX
- **a** *[double ,out]* :: Helmholtz energy [J/mol]
- **g** *[double ,out]* :: Gibbs free energy [J/mol]
- **xkappa** *[double]* :: XXXXXXXXXX
- **beta** *[double]* :: XXXXXXXXXX
- **dPdD** *[double]* :: XXXXXXXXXX
- **d2PdD2** *[double]* :: XXXXXXXXXX
- **dPdT** *[double]* :: XXXXXXXXXX
- **dDdT** *[double]* :: XXXXXXXXXX
- **dDdP** *[double]* :: XXXXXXXXXX
- **d2PdT2** *[double]* :: XXXXXXXXXX
- **d2PdTD** *[double]* :: XXXXXXXXXX
- **spare3** *[double]* :: XXXXXXXXXX
- **spare4** *[double]* :: XXXXXXXXXX

subroutine THERM3dll (*T*, *D*, *z*, *xkappa*, *beta*, *xisenk*, *xkt*, *betas*, *bs*, *xkkt*, *throtl*, *pi*, *spht*)

Compute miscellaneous thermodynamic properties. See warning in subroutines THERM or ALLPROPS.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **xkappa** *[double ,out]* :: Isothermal compressibility [1/kPa]
- **beta** *[double ,out]* :: Volume expansivity [1/K]
- **xisenk** *[double ,out]* :: Isentropic expansion coefficient [-]
- **xkt** *[double ,out]* :: Isothermal expansion coefficient [-]
- **betas** *[double ,out]* :: Adiabatic compressibility [1/kPa]
- **bs** *[double ,out]* :: Adiabatic bulk modulus [kPa]
- **xkkt** *[double ,out]* :: Isothermal bulk modulus [kPa]
- **throtl** *[double ,out]* :: Isothermal throttling coefficient [L/mol]
- **pi** *[double ,out]* :: Internal pressure [kPa]
- **spht** *[double ,out]* :: Specific heat input [J/mol]

subroutine THERMdll (*T*, *D*, *z*, *P*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *hjt*)

Compute thermal quantities as a function of temperature, density, and composition from core functions (Helmholtz energy, ideal gas heat capacity, and various derivatives and integrals).

Warning: Do NOT call this routine for two-phase states, otherwise it will return a metastable state if near the phase boundary or complete nonsense at other conditions. The value of q that is returned from the flash routines will indicate a two phase state by returning a value between 0 and 1. In such a situation, properties can only be calculated for the saturated liquid and vapor states. For example, when calling PHFLSH:

```
call PHFLSH (P,h,z,T,D,Dl,Dv,x,y,q,e,s,Cv,Cp,w,ierr,herr)
```

If q>0 and q<1, then values of the liquid compositions will be returned in the x and y arrays, and the properties of the liquid and vapor states can be calculated, for example, as:

```
call ENTRO (T,Dl,x,sliq)
call ENTRO (T,Dv,x,svap)
```

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **P** *[double ,out]* :: Pressure [kPa]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **hjt** *[double ,out]* :: Isenthalpic Joule-Thomson coefficient [K/kPa]

subroutine THFL1dll (*T*, *h*, *z*, *Dmin*, *Dmax*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase density as a function of temperature, enthalpy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **Dmin** *[double ,in]* :: Lower bound on density [mol/L]
- **Dmax** *[double ,in]* :: Upper bound on density [mol/L]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine THFLSHdll (*T*, *h*, *z*, *kr*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, bulk enthalpy, and bulk composition. Often in the liquid, two solutions exist, one of them in the two phase. If this is the case, call THFLSH with kr=2 to get the single-phase state. (See subroutines ABFLSH or TBFLSH for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]

- **h** *[double ,in]* :: Enthalpy [J/mol]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kr** *[int]* :: XXXXXXXXXX
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TPFL2dll (*T*, *P*, *z*, *Dl*, *Dv*, *x*, *y*, *q*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, pressure, and bulk composition. This routine accepts only two-phase states as inputs; if the phase is not known use TPFLSH. Use TPRHO for single-phase states.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Overall composition (array of mole fractions)
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality on a MOLAR basis (moles vapor/total moles)
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

213 TPRHO did not converge

226 Iteration did not converge

subroutine TPFLSHdll (*T*, *P*, *z*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, pressure, and bulk composition. This routine accepts both single-phase and two-phase states as inputs; for single-phase calculations, the subroutine TPRHO is much faster.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Overall composition (array of mole fractions)
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole or mass fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole or mass fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255) (See subroutine ABFLSH for the description of all other output variables.)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

213 Density calculation did not converge

215 Supercritical density calculation did not converge

223 Bubble point did not converge

224 Dew point did not converge

subroutine TPRHOPRdll (*T*, *P*, *z*, *D1*, *D2*)

Compute density with a volume-translated modification of the Peng-Robinson equation of state:

```
P=RT/(v+c+b)-a/((v+c)*(v+c+b)+b(v+c+b))
```

c is a translation constant, as given in Peneloux and Rauzy, Fluid Phase Equilib. 8:7-23, 1982.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)

- **D1** *[double ,out]* :: Largest density root [mol/L]
- **D2** *[double ,out]* :: Smallest density root [mol/L]

subroutine TPRHOdll (*T*, *P*, *z*, *kph*, *kguess*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for density as a function of temperature, pressure, and composition of a specified phase.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **P** *[double ,in]* :: Pressure [kPa]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **kph** *[int ,in]* :: Phase flag
- **kguess** *[int ,in]* :: Guess flag
- **D** *[double ,out]* :: Molar density [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `kph` flags

1 Liquid phase

2 Vapor phase

0 Any single phase state when T>Tc or P>Pc. For T<Tc or P<Pc, this option is not allowed and TPFLSH should be used instead. TPFLSH is required in order to find the saturation state, which is then used to determine if the state is liquid or vapor. However, if an initial guess is supplied for D then kph=0 can be used and the routine will determine the phase by checking if D<Dc or D>Dc.

-1 Force the search in the liquid phase (for metastable points)

-2 Force the search in the vapor phase (for metastable points)

`kguess` flags

0 No first guess for D provided

1 First guess for D provided

`ierr` flags

0 Successful

201 Illegal input (kph <= 0)

202 Liquid-phase iteration did not converge

203 Vapor-phase iteration did not converge

subroutine TQFLSHdll (*T*, *q*, *z*, *kq*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *e*, *h*, *s*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, quality, and bulk composition. This routine accepts saturation or two-phase states as inputs. (See subroutines ABFLSH or AQFLSH for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **q** *[double ,in]* :: Vapor quality [mol/mol]

- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kq** *[int]* :: XXXXXXXXXX
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **s** *[double ,out]* :: Entropy [J/mol-K]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]
- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TRNPRPdll (*T*, *D*, *z*, *eta*, *tcx*, *ierr*, *herr*, *herr_length*)

Compute the transport properties thermal conductivity and viscosity as functions of temperature, density, and composition.

Warning: Do NOT call this routine for two-phase states, otherwise it will return a metastable state if near the phase boundary or complete nonsense at other conditions. The value of q that is returned from the flash routines will indicate a two phase state by returning a value between 0 and 1. In such a situation, the transport properties can only be calculated for the saturated liquid and vapor states. For example, when calling PHFLSH:

```
call PHFLSH (P,h,z,T,D,Dl,Dv,x,y,q,e,s,Cv,Cp,w,ierr,herr)
```

If q>0 and q<1, then values of the liquid compositions will be returned in the x and y arrays, and the transport properties of the liquid and vapor states can be calculated as follows:

```
call TRNPRP (T,Dl,x,etaliq,tcxliq,ierr,herr)
call TRNPRP (T,Dv,y,etavap,tcxvap,ierr,herr)
```

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **D** *[double ,in]* :: Molar density [mol/L]
- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **eta** *[double ,out]* :: Viscosity [uPa-s]
- **tcx** *[double ,out]* :: Thermal conductivity [W/m-K]
- **ierr** *[int ,out]* :: Error flag

- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

502 Unknown viscosity or thermal conductivity model specified

73 One or more inputs are out of bounds for the ECS model

74 Inputs to the vis. and th. cond. correlations are out 2of range

540 Transport equations are not available for one or more of the fluids

541 Transport equations are not available for mixtures with water at molar concentrations greater than 5%

542 Transport equations are not available for mixtures with alcohols

543 Transport equations are not available for the ammonia/water mixture

-508 Invalid region for viscosity of reference fluid 1

-558 2-D Newton-Raphson method for conformal temperature and density did not converge

-560 Pure fluid is exactly at the critical point; thermal conductivity is infinite

561 Pure fluid correlation produced an erroneous value for either viscosity or thermal conductivity

5## Pure fluid correlation out of range; attempted to use ECS method

subroutine TSATDdll (*D*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Compute the saturated temperature as a function of saturated density and composition.

Parameters

- **D** *[double ,in]* :: Saturated density [mol/L]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

2 Density greater than triple point density

501 No equation available

subroutine TSATPdll (*P*, *z*, *T*, *ierr*, *herr*, *herr_length*)

Compute the vapor temperature as a function of pressure and composition.

If the input pressure is negative, compute the liquid temperature as a function of liquid pressure and composition (used only for pseudo-pure fluids).

Parameters

- **P** *[double ,in]* :: Vapor pressure [kPa] If negative, liquid pressure [kPa] (the negative sign is only used as a flag).
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **T** *[double ,out]* :: Temperature [K] ierr, herr; See error codes in the ANCERR2 routine.
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TSFL1dll (*T*, *s*, *z*, *Dmin*, *Dmax*, *D*, *ierr*, *herr*, *herr_length*)

Iterate for single-phase density as a function of temperature, entropy, and composition. (See subroutine ABFL1 for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **Dmin** *[double ,in]* :: Lower bound on density [mol/L]
- **Dmax** *[double ,in]* :: Upper bound on density [mol/L]
- **D** *[double ,out]* :: Density [mol/L]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine TSFLSHdll (*T*, *s*, *z*, *kr*, *P*, *D*, *Dl*, *Dv*, *x*, *y*, *q*, *e*, *h*, *Cv*, *Cp*, *w*, *ierr*, *herr*, *herr_length*)

Flash calculation given temperature, bulk entropy, and bulk composition. (See subroutines ABFLSH or TBFLSH for the description of all variables.)

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **s** *[double ,in]* :: Entropy [J/mol-K]
- **z** (20) *[double ,in]* :: Bulk Composition (array of mole fractions)
- **kr** *[int]* :: XXXXXXXXXX
- **P** *[double ,out]* :: Pressure [kPa]
- **D** *[double ,out]* :: Density [mol/L]
- **Dl** *[double ,out]* :: Molar density of the liquid phase [mol/L]
- **Dv** *[double ,out]* :: Molar density of the vapor phase [mol/L]
- **x** (20) *[double ,out]* :: Composition of the liquid phase (array of mole fractions)
- **y** (20) *[double ,out]* :: Composition of the vapor phase (array of mole fractions)
- **q** *[double ,out]* :: Vapor quality [mol/mol]
- **e** *[double ,out]* :: Internal energy [J/mol]
- **h** *[double ,out]* :: Enthalpy [J/mol]
- **Cv** *[double ,out]* :: Isochoric heat capacity [J/mol-K]

- **Cp** *[double ,out]* :: Isobaric heat capacity [J/mol-K]
- **w** *[double ,out]* :: Speed of sound [m/s]
- **ierr** *[int ,out]* :: Error code (no error if ierr==0)
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine UNSETAGAdll()
Load original values into arrays changed in the call to SETAGA. This routine resets the values back to those loaded when SETUP was called.

subroutine VAPSPNDLdll(*T*, *z*, *D*, *ierr*, *herr*, *herr_length*)
Find the vapor spinodal density for a given temperature. If no spinodal exists, return the point of zero curvature. This only happens with a few of the older equations, these being D5, methanol, and nitrogen.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **D** *[double ,out]* :: Density at vapor spinodal [mol/L]
- **ierr** *[int ,out]* :: Error flag
- **herr** *[char ,out]* :: Error string (character*255)
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

Flags `ierr` flags

0 Successful

121 T>Tc

633 Failed to converge

-638 Spinodal not found, point of zero curvature returned

subroutine VIRBAdll(*T*, *z*, *Ba*)
Compute the second acoustic virial coefficient Ba (L/mol) as a function of temperature (K) and composition x (array of mole fractions). For further information, see Trusler and Zarari, J. Chem. Thermodyn., 28:329-335, 1996. Gillis and Moldover, Int. J. Theromphys., 17(6):1305-1324, 1996. This routine approximates Ba. For pure fluids, the routine VIRBCD is exact.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Ba** *[double ,out]* :: Second acoustic virial coefficient [L/mol]

subroutine VIRBCD12dll(*T*, *z*, *iFlag*, *B*, *C*, *D*, *E*)
Compute virial coefficients as a function of temperature and composition. The routine currently works only for pure fluids and for the Helmholtz equation. All values are computed exactly based on the terms in the EOS, not as was done in VIRB by calculating properties at a density of 1d-8.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)

- **iFlag** *[int ,in]* :: Flag to specify the outputs (or for use in future applications). 0 - Return virial coefficients. 1 - Return acoustic virial coefficients; see: Trusler and Zarari, J. Chem. Thermodyn., 28:329-335, 1996. Gillis and Moldover, Int. J. Theromphys., 17(6):1305-1324, 1996.
- **B** (6) *[double]* :: XXXXXXXXXX
- **C** (6) *[double]* :: XXXXXXXXXX
- **D** (6) *[double]* :: XXXXXXXXXX
- **E** (6) *[double]* :: XXXXXXXXXX

subroutine VIRBCDdll (*T*, *z*, *B*, *C*, *D*, *E*)

Compute virial coefficients as a function of temperature and composition. The routine currently works only for pure fluids and for the Helmholtz equation. All values are computed exactly based on the terms in the EOS, not as was done in VIRB by calculating properties at a density of 1d-8.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **B** *[double ,out]* :: Second virial coefficient [L/mol] = a01
- **C** *[double ,out]* :: Third virial coefficient [(L/mol)^2] = a02
- **D** *[double ,out]* :: Fourth virial coefficient [(L/mol)^3] = a03/2d0
- **E** *[double ,out]* :: Fifth virial coefficient [(L/mol)^4] = a04/6d0 assume nomenclature of a02=[partial^2(alphar)/partial(del)^2] above

subroutine VIRBdll (*T*, *z*, *B*)

Compute the second virial coefficient B (L/mol) as a function of temperature T (K) and composition x (array of mole fractions). This routine approximates B. For pure fluids, the routine VIRBCD is exact.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **B** *[double ,out]* :: Second virial coefficient [L/mol]

subroutine VIRCAdll (*T*, *z*, *Ca*)

Compute the third acoustic virial coefficient Ca (L/mol)^2 as a function of temperature (K) and composition x (array of mole fractions). For further information, see Estela-Uribe and Trusler, Int. J. Theromphys., 21(5):1033, 2000. Gillis and Moldover, Int. J. Theromphys., 17(6):1305-1324, 1996. This routine approximates Ca. For pure fluids, the routine VIRBCD is exact.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **Ca** *[double ,out]* :: Third acoustic virial coefficient [(L/mol)^2]

subroutine VIRCdll (*T*, *z*, *C*)

Compute the third virial coefficient C (L/mol)^2 as a function of temperature T (K) and composition x (array of mole fractions). This routine approximates C. For pure fluids, the routine VIRBCD is exact.

Parameters

- **T** *[double ,in]* :: Temperature [K]

- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **C** *[double ,out]* :: Third virial coefficient [(L/mol)^2] = a02

subroutine VIRTAUdll (*T*, *z*, *iFlag*, *BTau*, *CTau*, *DTau*, *ETau*, *ierr*, *herr*, *herr_length*)
Compute virial coefficients as a function of tau and composition. The routine currently works only for pure fluids and for the Helmholtz equation. All values are computed exactly based on the terms in the EOS, not as was done in VIRB by calculating properties at a density of 1d-8.

Parameters

- **T** *[double ,in]* :: Temperature [K]
- **z** (20) *[double ,in]* :: Composition (array of mole fractions)
- **iFlag** *[int]* :: XXXXXXXXXX
- **BTau** (6) *[double]* :: XXXXXXXXXX
- **CTau** (6) *[double]* :: XXXXXXXXXX
- **DTau** (6) *[double]* :: XXXXXXXXXX
- **ETau** (6) *[double]* :: XXXXXXXXXX
- **ierr** *[int]* :: XXXXXXXXXX
- **herr** *[char]* :: XXXXXXXXXX
- **herr_length** *[int]* :: length of variable `herr` (default: 255)

subroutine WMOLIdll (*icomp*, *wmm*)
Return the molar mass (molecular weight) of a component in a mixture.

Parameters

- **icomp** *[int ,in]* :: Component number in the mixture
- **wmm** *[double]* :: XXXXXXXXXX

subroutine WMOLdll (*z*, *wmm*)
Return the molar mass (molecular weight) for a mixture of a specified composition.

Parameters

- **z** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **wmm** *[double]* :: XXXXXXXXXX

subroutine XMASSdll (*xmol*, *xkg*, *wmix*)
Converts composition on a mole fraction basis to mass fractions.

Parameters

- **xmol** (20) *[double ,in]* :: Composition array (array of mole fractions)
- **xkg** (20) *[double ,out]* :: Composition array (array of mass fractions)
- **wmix** *[double ,out]* :: Molar mass of the mixture [g/mol], a.k.a. molecular weight

subroutine XMOLEdll (*xkg*, *xmol*, *wmix*)
Converts composition on a mass fraction basis to mole fraction.

Parameters

- **xkg** (20) *[double ,in]* :: Composition array (array of mass fractions)
- **xmol** (20) *[double ,out]* :: Composition array (array of mole fractions)

- **wmix** *[double ,out]* :: Molar mass of the mixture [g/mol], a.k.a. molecular weight

Index

A

ABFL1dll() (*fortran subroutine*), 61
ABFL2dll() (*fortran subroutine*), 63
ABFLSHdll() (*fortran subroutine*), 29
AGdll() (*fortran subroutine*), 64
ALLPROPS0dll() (*fortran subroutine*), 32
ALLPROPS1dll() (*fortran subroutine*), 32
ALLPROPSdll() (*fortran subroutine*), 33

B

B12dll() (*fortran subroutine*), 64
BLCRVdll() (*fortran subroutine*), 64

C

CHEMPOTdll() (*fortran subroutine*), 65
CP0dll() (*fortran subroutine*), 65
CRITPdll() (*fortran subroutine*), 66
CRTPNTdll() (*fortran subroutine*), 66
CSATKdll() (*fortran subroutine*), 66
CSTARdll() (*fortran subroutine*), 65, 67
CV2PKdll() (*fortran subroutine*), 67
CVCPdll() (*fortran subroutine*), 68

D

DBDTdll() (*fortran subroutine*), 68
DBFL1dll() (*fortran subroutine*), 68
DEFL1dll() (*fortran subroutine*), 69
DEFLSHdll() (*fortran subroutine*), 69
DERVPVTdll() (*fortran subroutine*), 70
DHD1dll() (*fortran subroutine*), 70
DHFL1dll() (*fortran subroutine*), 70
DHFLSHdll() (*fortran subroutine*), 71
DIELECdll() (*fortran subroutine*), 71
DLSATKdll() (*fortran subroutine*), 72
DPDD2dll() (*fortran subroutine*), 72
DPTSATKdll() (*fortran subroutine*), 72
DQFL2dll() (*fortran subroutine*), 73
DSD1dll() (*fortran subroutine*), 73
DSFL1dll() (*fortran subroutine*), 73
DSFLSHdll() (*fortran subroutine*), 74
DVSATKdll() (*fortran subroutine*), 74

E

ENTHALdll() (*fortran subroutine*), 75
ENTROdll() (*fortran subroutine*), 75
ERRMSGdll() (*fortran subroutine*), 40
ESFLSHdll() (*fortran subroutine*), 75
EXCESSdll() (*fortran subroutine*), 76

F

FGCTY2dll() (*fortran subroutine*), 76
FGCTYdll() (*fortran subroutine*), 76
FLAGSdll() (*fortran subroutine*), 41
FPVdll() (*fortran subroutine*), 77
FUGCOFdll() (*fortran subroutine*), 77

G

GERG04dll() (*fortran subroutine*), 77
GERG08dll() (*fortran subroutine*), 77
GETENUMdll() (*fortran subroutine*), 45
GETFIJdll() (*fortran subroutine*), 78
GETKTVdll() (*fortran subroutine*), 78
GETMODdll() (*fortran subroutine*), 79
GETREFDIRdll() (*fortran subroutine*), 80
GIBBSdll() (*fortran subroutine*), 80

H

HEATdll() (*fortran subroutine*), 81
HEATFRMdll() (*fortran subroutine*), 80
HMXORDERdll() (*fortran subroutine*), 81
HSFL1dll() (*fortran subroutine*), 82
HSFLSHdll() (*fortran subroutine*), 82

I

IDCRVdll() (*fortran subroutine*), 83
INFOdll() (*fortran subroutine*), 83

J

JICRVdll() (*fortran subroutine*), 83

JTCRVdll() (*fortran subroutine*), 84

L

LIMITKdll() (*fortran subroutine*), 84
LIMITSdll() (*fortran subroutine*), 85
LIMITXdll() (*fortran subroutine*), 85
LIQSPNDLdll() (*fortran subroutine*), 86

M

MASSFLUXdll() (*fortran subroutine*), 87
MAXPdll() (*fortran subroutine*), 87
MAXTdll() (*fortran subroutine*), 88
MELTKdll() (*fortran subroutine*), 88
MELTPdll() (*fortran subroutine*), 89
MELTTdll() (*fortran subroutine*), 89
MLTH2Odll() (*fortran subroutine*), 89

N

NAMEdll() (*fortran subroutine*), 90

P

PASSCMNdll() (*fortran subroutine*), 90
PDFL1dll() (*fortran subroutine*), 91
PDFLSHdll() (*fortran subroutine*), 91
PEFL1dll() (*fortran subroutine*), 92
PEFLSHdll() (*fortran subroutine*), 92
PHFL1dll() (*fortran subroutine*), 93
PHFLSHdll() (*fortran subroutine*), 93
PHI0dll() (*fortran subroutine*), 94
PHIDERVdll() (*fortran subroutine*), 94
PHIHMXdll() (*fortran subroutine*), 95
PHIKdll() (*fortran subroutine*), 95
PHIMIXdll() (*fortran subroutine*), 96
PHIXdll() (*fortran subroutine*), 96
PQFLSHdll() (*fortran subroutine*), 96
PREOSdll() (*fortran subroutine*), 97
PRESSdll() (*fortran subroutine*), 97
PSATKdll() (*fortran subroutine*), 97
PSFL1dll() (*fortran subroutine*), 98
PSFLSHdll() (*fortran subroutine*), 98
PUREFLDdll() (*fortran subroutine*), 99

Q

QMASSdll() (*fortran subroutine*), 99
QMOLEdll() (*fortran subroutine*), 99

R

RDXHMXdll() (*fortran subroutine*), 100
REDXdll() (*fortran subroutine*), 100
REFPROP1dll() (*fortran subroutine*), 46
REFPROP2dll() (*fortran subroutine*), 47
REFPROPdll() (*fortran subroutine*), 47
RESIDUALdll() (*fortran subroutine*), 101
RIEMdll() (*fortran subroutine*), 101
RMIX2dll() (*fortran subroutine*), 101

S

SATDdll() (*fortran subroutine*), 101
SATEdll() (*fortran subroutine*), 103
SATESTdll() (*fortran subroutine*), 102
SATGUESSdll() (*fortran subroutine*), 104
SATGVdll() (*fortran subroutine*), 105
SATHdll() (*fortran subroutine*), 106
SATPdll() (*fortran subroutine*), 107
SATSdll() (*fortran subroutine*), 109
SATSPLNdll() (*fortran subroutine*), 108
SATTdll() (*fortran subroutine*), 111
SATTPdll() (*fortran subroutine*), 110
SETAGAdll() (*fortran subroutine*), 111
SETFLUIDSdll() (*fortran subroutine*), 56
SETKTVdll() (*fortran subroutine*), 112
SETMIXdll() (*fortran subroutine*), 113
SETMIXTUREdll() (*fortran subroutine*), 56
SETMODdll() (*fortran subroutine*), 113
SETNCdll() (*fortran subroutine*), 115
SETPATHdll() (*fortran subroutine*), 57
SETREFDIRdll() (*fortran subroutine*), 115
SETREFdll() (*fortran subroutine*), 115
SETUPdll() (*fortran subroutine*), 116
SPLNROOTdll() (*fortran subroutine*), 117
SPLNVALdll() (*fortran subroutine*), 118
STNdll() (*fortran subroutine*), 118
SUBLPdll() (*fortran subroutine*), 119
SUBLTdll() (*fortran subroutine*), 119
SURFTdll() (*fortran subroutine*), 119
SURTENdll() (*fortran subroutine*), 120

T

TDFLSHdll() (*fortran subroutine*), 120
TEFL1dll() (*fortran subroutine*), 121
TEFLSHdll() (*fortran subroutine*), 121
THERM0dll() (*fortran subroutine*), 122
THERM2dll() (*fortran subroutine*), 122
THERM3dll() (*fortran subroutine*), 123
THERMdll() (*fortran subroutine*), 123
THFL1dll() (*fortran subroutine*), 124
THFLSHdll() (*fortran subroutine*), 124
TPFL2dll() (*fortran subroutine*), 125
TPFLSHdll() (*fortran subroutine*), 125
TPRHOdll() (*fortran subroutine*), 127
TPRHOPRdll() (*fortran subroutine*), 126
TQFLSHdll() (*fortran subroutine*), 127
TRNPRPdll() (*fortran subroutine*), 128
TSATDdll() (*fortran subroutine*), 129
TSATPdll() (*fortran subroutine*), 129
TSFL1dll() (*fortran subroutine*), 130
TSFLSHdll() (*fortran subroutine*), 130

U

UNSETAGAdll() (*fortran subroutine*), 131

V

VAPSPNDLdll() (*fortran subroutine*), 131
VIRBAdll() (*fortran subroutine*), 131
VIRBCD12dll() (*fortran subroutine*), 131
VIRBCDdll() (*fortran subroutine*), 132
VIRBdll() (*fortran subroutine*), 132
VIRCAdll() (*fortran subroutine*), 132
VIRCdll() (*fortran subroutine*), 132
VIRTAUdll() (*fortran subroutine*), 133

W

WMOLdll() (*fortran subroutine*), 133
WMOLIdll() (*fortran subroutine*), 133

X

XMASSdll() (*fortran subroutine*), 133
XMOLEdll() (*fortran subroutine*), 133