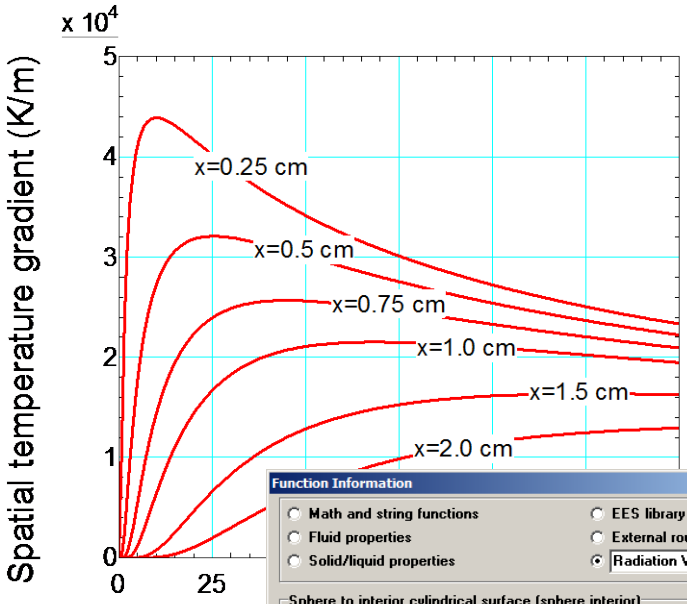
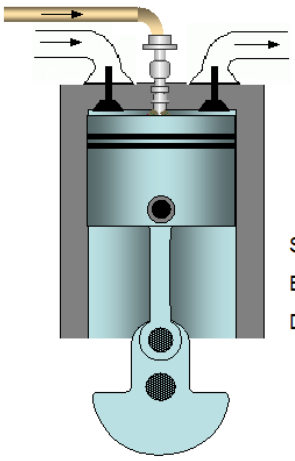
F-Chart Software

Mastering EES

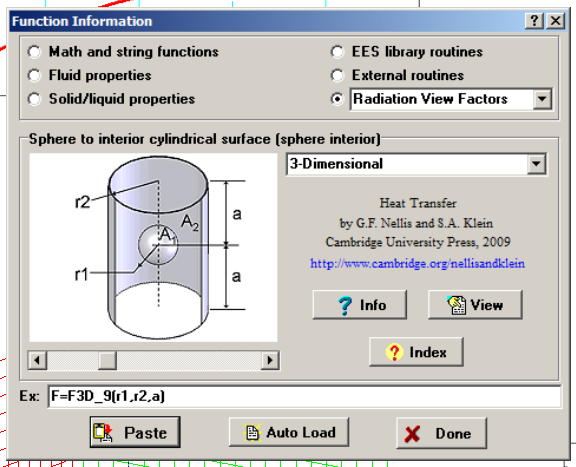
Sanford Klein and Gregory Nellis

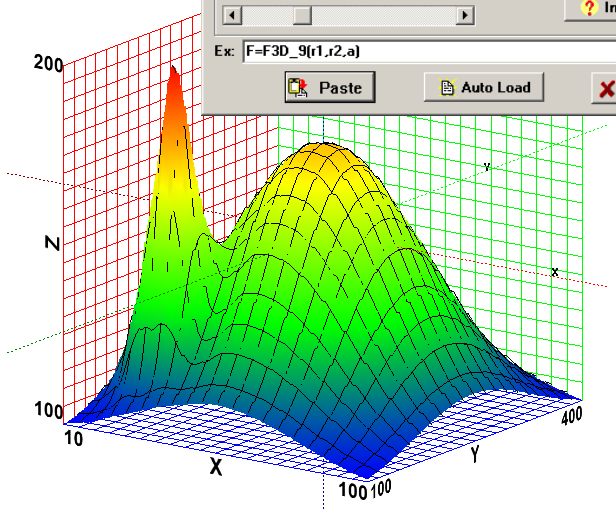


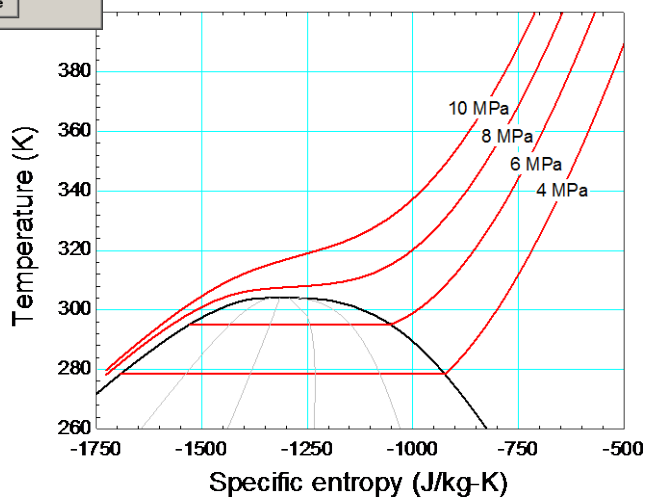


Stroke = 75 [mm]
Bore = 130 [mm]
Displacement = 0.9955 [l]

CompressionRatio = 18 Efficiency = 0
RPM = 3000 [1/min] Power = 0 [kW]
CutoffRatio = 2 T_{max} = 0 [K]







F-Chart Software

Box 44042

Madison, WI 53744
http://fchart.com

July 30, 2012

TABLE OF CONTENTS

1 INTRODUCTION TO EES

EES (pronounced 'ease') is an acronym for Engineering Equation Solver. The basic function provided by EES is the numerical solution of linear and non-linear algebraic and differential equations. It is important to recognize that EES is an *equation*-solver. EES utilizes equations rather than the assignments that are used in a formal programming language. EES solves systems of equations (i.e., relationships between variables) automatically, which frees the user from having to develop their own iterative technique for solving a set of non-linear equations. There are many additional features associated with EES; for example such as unit checking, optimization, numerical integration, high quality property data, plotting and uncertainty analyses. These features make EES a powerful tool for developing mathematical models of many types of engineering systems. This chapter discusses the basic features of EES.

1.1 Acquiring and Installing EES

This section provides information about obtaining and installing the EES software. If EES is already installed on your computer, you can skip this section.

Acquiring EES

EES is licensed and distributed solely by F-Chart Software LLC. A license for EES can be purchased directly from the company website, http://fchart.com. Single and multi-user licenses are available. In addition, the company offers academic licenses for educational institutions involved in formal classroom instruction and academic research leading to a degree. Details and costs for the different license options can be found at http://fchart.com/ees/. If you do not have access to EES, you can download a demonstration copy of the program from http://fchart.com/ees/demo.php. The demonstration copy will allow you to examine most of the features and examples that are provided in this book.

Commercial and Professional versions are available. The Commercial version provides all of the basic functionality of EES, includes the ability to solve 6000 simultaneous equations and allows access high accuracy property information. It can be used to conduct parametric studies, do unit conversion and check unit consistency, provide publication quality plots, perform uncertainty analyses, and many other things.

The Professional version provides many additional features, as detailed at http://fchart.com/ees/pro-comm-versions.php. The most important of these features are the ability to solve larger systems of equations and the ability to create special purpose executable EES programs that can be freely distributed to others. This book will provide information for both the Commercial and Professional versions. Features that are only available in the Professional version will be designated using the side bar adjacent to this paragraph.

Installing EES

Installing EES is relatively straightforward, but there are a few options that merit discussion. To install the program, you will need both the Setup_EES.exe and EES.dft files. These files should have been provided to you when you purchased the program or by a system administrator if you obtained the program from your company or institution. There are a few institutions where the EES.dft file has already been bundled with the setup file. Execute the Setup_EES.exe program and click the Next button from the Welcome screen. Read the license agreement and, if you agree, click the I Agree button and then click the Next button. Continue to click the Next button until you see the Select Destination Directory dialog, shown in Figure 1-1.

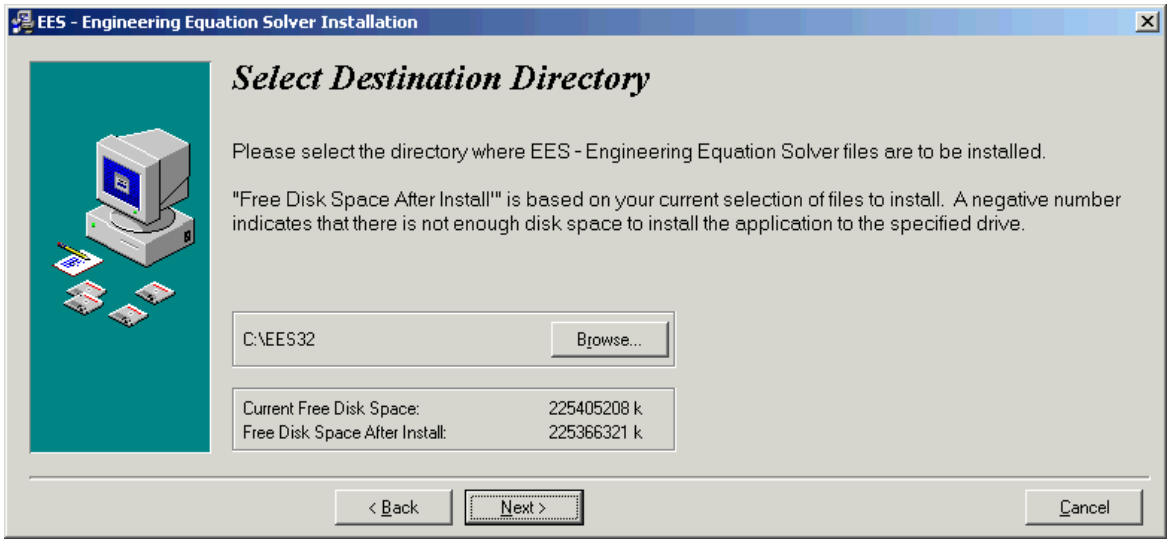


Figure 1-1: Select Destination Directory dialog.

For historical reasons, the default directory that EES will be installed into is named EES32. EES is a 32-bit program, but it will operate on all modern 32-bit and 64-bit Windows operating systems. You can change the directory name, if you wish, by clicking the Browse button. Depending on the particular installation program, you will need to copy the EES.dft file to the directory that you choose to install EES. Click the Next button after you have selected the directory.

The next window, shown in Figure 1-2, provides several installation options. If selected, the Install Heat Transfer Library option will install an extensive set of library functions that facilitate heat transfer calculations. These functions have been developed to accompany the text book *Heat Transfer*, by G. F. Nellis and S.A. Klein (2009). Selection of this option is recommended unless you are sure that you will never be doing any heat transfer calculations. A discussion of these library functions is provided in Chapter 12.

Two versions of EES are provided in the installation program. One version is specifically developed to allow 3-dimensional plotting. Checking the Install 3D plotting capability option in the Select Components dialog will install this version. The only reason for not selecting this option is that it may be incompatible with the graphic display capabilities of some computer systems. If EES fails to start after installation then this is likely the problem. In this case,

reinstall, and unselect this option, as shown in Figure 1-2. Note that it is recommended that you not install the 3D version if you are developing distributable programs with the Professional version, as discussed in Chapter 17. Distributable programs should not use the 3D version because it may not operate properly on your users' computers.

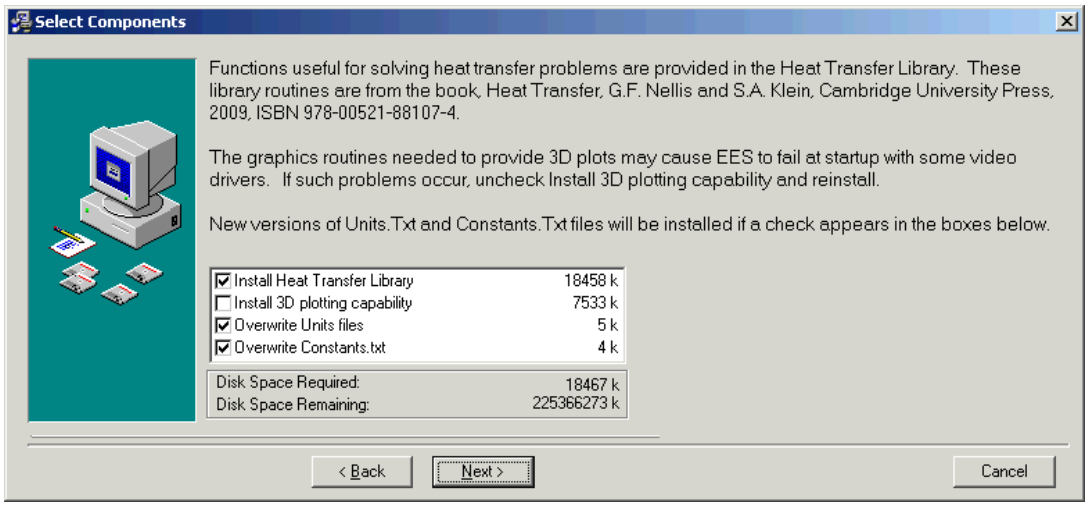


Figure 1-2: Select Components dialog.

EES allows you to save custom Units and Constants files as discussed in Sections 1.5 and 1.9. If you have done this, unselect the Overwrite Units files and Overwrite Constants.txt file in Figure 1-2. Otherwise these files will be replaced by the files in the installation program.

Silent Installation

It is possible to install EES silently, so that it installs the program with the default options and without displaying any confirmation windows. To do a silent install, enter:

```
C:\myDir\Setup_EES.exe /s
```

into the Windows Run dialog after replacing myDir with the directory name where the Setup_EES.exe program is located.

Clicking the Next button will install the program. If EES fails to start, the most likely problem is that the EES.dft file is not contained in the directory that the program was installed into. In this case, copy the EES.dft into the install directory and then restart EES by double-clicking on the EES.exe icon or by entering C:\EES32\EES.exe in the Windows run dialog.

1.2 Entering and Solving Equations

A mathematical model of an engineering system is based on a system of equations that result from the application of appropriate theory and simplifications. These equations describe mass, energy, and entropy balances, rate relations, properties, etc. Modern computer tools, such as EES, facilitate the solution of the large set of coupled equations that result from the analysis of a typical engineering system. When you open EES for the first time you will encounter the Equations Window where the equations that are to be solved are entered. EES allows the user to enter *equations* rather than *assignments*, as are required by most formal programming languages. This is an important distinction. In an assignment statement, the value of each variable on the right side of an expression must have been previously determined. An equation is simply a relationship between variables. Assignments are explicit and can be solved sequentially. A set of equations may be implicit and nonlinear and must be solved simultaneously and iteratively.

Entering Equations

Consider the following set of equations:

$$x + y = 3 \tag{1-1}$$

$$y = z - 4 \tag{1-2}$$

$$z = x^2 - 3 \tag{1-3}$$

Equations (1-1) through (1-3) are three non-linear equations in the three unknowns *x*, *y*, and *z*. However, they are not directly solvable using most formal programming languages (e.g., MATLAB or Fortran) because they are equations rather than assignments. In order to solve this system of equations using a formal programming language, it would be necessary to either employ an iterative solution technique (e.g., successive substitution) or carry out sufficient algebra to convert the equations into assignments. Let's take the latter approach here. Substituting Eq. (1-3) into Eq. (1-2) provides:

$$y = x^2 - 3 - 4 \tag{1-4}$$

Substituting Eq. (1-4) into Eq. (1-1) provides:

$$x + x^2 - 7 = 3 \tag{1-5}$$

which can be rearranged:

$$x^2 + x - 10 = 0 \tag{1-6}$$

Equation (1-6) is a quadratic equation that can be solved using the quadratic formula:

$$x = \frac{-1 \pm \sqrt{1^2 - 4(-10)}}{2} \qquad \sqrt{} \tag{1-7}$$

Equation (1-7) has two solutions:

$$x = -\frac{1}{2} + \frac{\sqrt{41}}{2} = 2.702 \tag{1-8}$$

and

$$x = -\frac{1}{2} - \frac{\sqrt{41}}{2} = -3.702 \tag{1-9}$$

Substituting Eqs. (1-8) and (1-9) into Eqs. (1-3) and (1-4) provides the two solutions to the original set of equations: x = 2.702 or -3.702, y = 0.2984 or 6.702, and z = 4.298 or 10.70.

At its most fundamental level, EES is an equation solver that solves sets of nonlinear equations directly. For example, the three equations (1-1) through (1-3) can be entered directly into the Equations Window, as shown in Figure 1-3.

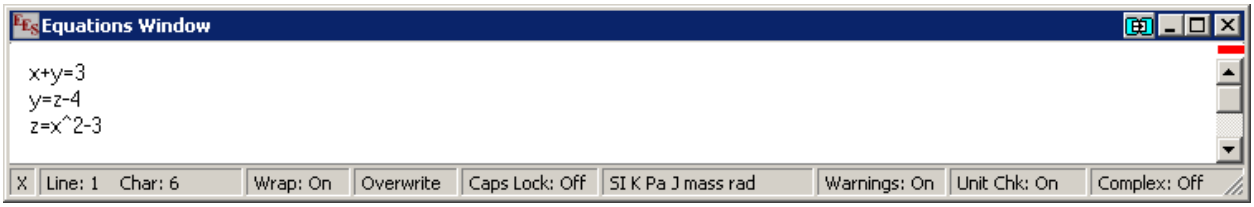


Figure 1-3: Equations Window with Eqs. (1-1) through (1-3) entered.

In this book, text that is entered in the Equations Window will be shaded, as shown below.

```
x+y=3
y=z-4
z=x^2-3
```

The Solutions Window

Select Solve from the Calculate menu (or use the shortcut F2) in order to initiate the iterative process that EES uses internally to solve the system of equations. EES will re-order and block the equations in a logical manner (see Chapter 5) and then, starting from a guessed solution, it will iteratively search for an actual solution to the equations. The result should be the dialog shown in Figure 1-4, which shows that the calculations were successfully completed and provides some of the details of the process (which will be discussed more completely in Chapter 5).

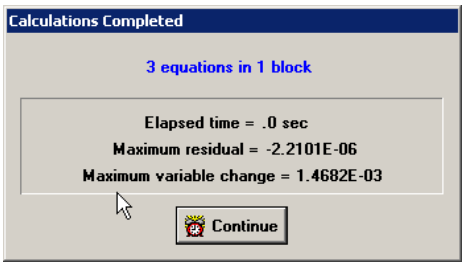


Figure 1-4: Dialog indicating that the calculations are complete.

Select Continue to proceed to the Solutions Window, shown in Figure 1-5.

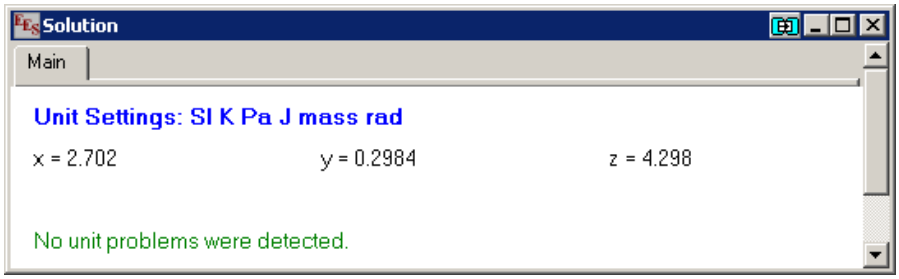


Figure 1-5: Solutions Window.

EES has identified one of the two solutions to the equation set. The other solution can be found by changing guess values, as explained below. The process of solving the equations was done internally; the user is not required to carry out any algebra or iteration.

Decimal vs Comma Separator

The solution is shown in Figure 1-5 using the decimal point as the decimal separator. However, EES operates just as well using a European numerical formatting style. Close EES and open the Regional and Language Options dialog in the Windows Control Panel. Under the Regional Options tab, select a European language (e.g., French). Open EES and solve the problem again to obtain the Solutions Window shown in Figure 1-6; note that the decimal separator is now a comma rather than a decimal point.

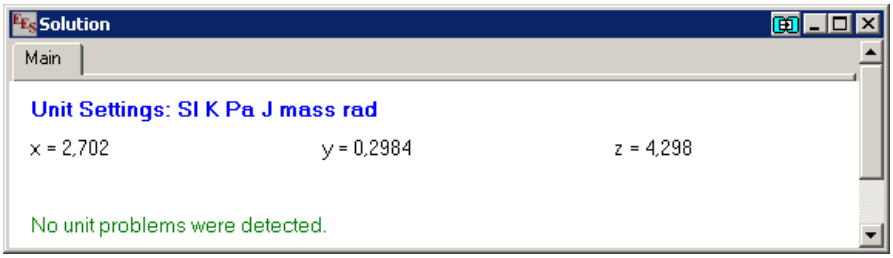


Figure 1-6: Solutions Window using a European format.

The Variable Information Window

Why did EES identify one of the solutions (x = 2.702, y = 0.2984, and z = 4.298) as opposed to the other one (x = -3.702, y = 6.702, and z = 10.70)? EES uses a variation of Newton's method to solve systems of equations, as will be discussed in Chapter 5. This technique begins with an assumed or guessed value of each variable and then iteratively adjusts these values until the equations are satisfied. In situations where multiple solutions exist, EES will likely converge to the solution that is closest to the guessed solution that is used to start the process. Select Variable Info (or press F9) from the Options menu in order to access the Variable Information Window, shown in Figure 1-7.

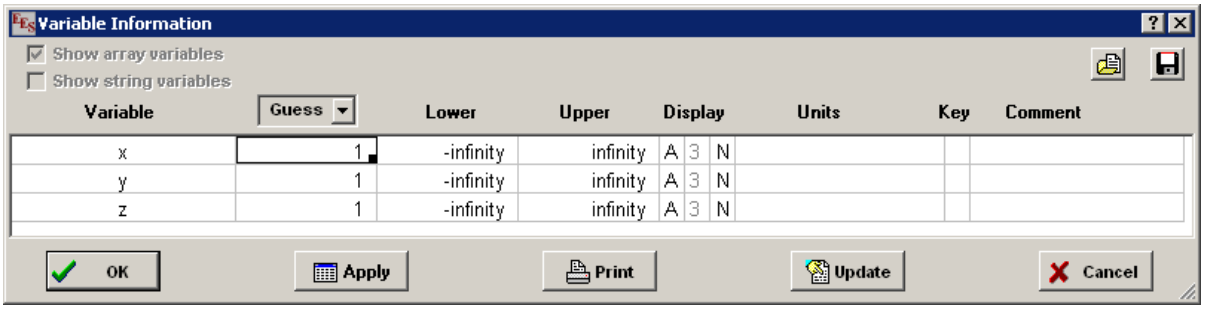


Figure 1-7: Variable Information Window.

Guess Values

There is a row corresponding to each of the three variables that make up the problem. Columns allow the user to change various characteristics of these variables. The first column corresponds to the guess values for each variable; these are the values used to start the iterative solution process. By default, the guess value for each variable is 1. Note that x = 1, y = 1, and z = 1 is not the correct solution, but it is closer to the solution x = 2.702, y = 0.2984, and z = 4.298 than it is to the alternative solution x = -3.702, y = 6.702, and z = 10.7. In order to converge to the other solution, it is necessary to change the guess values. For example, change the guess value of x to something closer to x = -3.702 (e.g., -5), select OK and press F2 (the shortcut for the Solve command). The solution identified by EES is shown in Figure 1-8. Guess values can also be provided using other EES variables or equations, as described in Section 5.3.

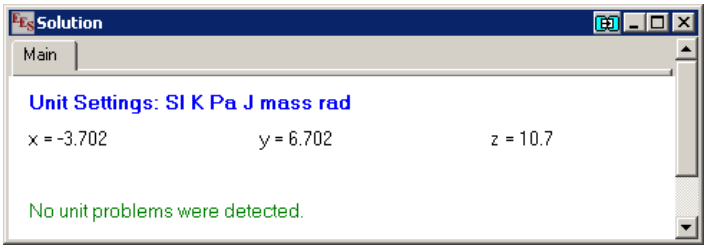


Figure 1-8: Solution Window with modified guess value.

Limits

Lower and upper limits for each variable can also be set in the Variable Information Window. For example, in order to identify only a solution for which the value of *x* is negative, the Variable Information Window could have been set up as shown in Figure 1-9.

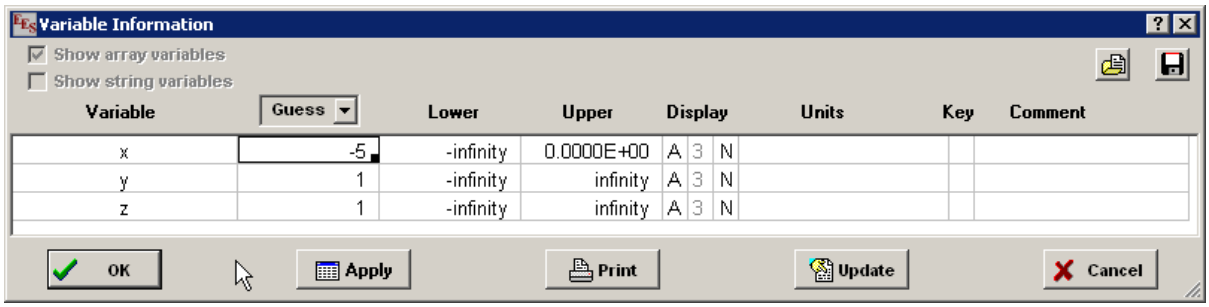


Figure 1-9: Variable Information Window with limits set for the variable x.

Display Format

All EES numerical values use extended precision that internally provides 20 significant figures of precision. However, the display format can be separately specified for each variable in the Variable Information Window. The A in the first of the three columns under the word Display indicates Auto format. Click on the A under display for the variable *y* and select Fixed decimal (F) with 9 significant figures. Click on the N (normal) in the third display column on the same row and select boxed (X). Hit OK and then solve the equations to obtain the solutions window shown in Figure 1-10. Notice that the solution for *y* is shown in fixed decimal format to 9 significant figures and the result is boxed.

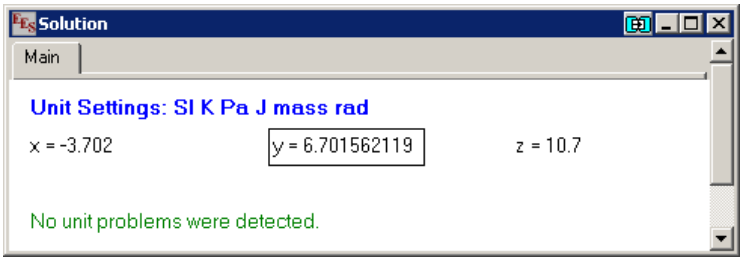


Figure 1-10: Solutions Window with an altered format for the variable y.

The display format can also be set directly in the Solution Window by right-clicking on a variable and using the resulting Format Selected Variables dialog.

Rules for Entering Equations

Let's return to the Equations Window. Notice that the equations can be entered in any order and rearranged algebraically in any way. For example, we can switch the order of the first two equations and switch the right and left sides of the third equation and EES will identify the same solution.

```
y=z-4
x+y=3
x^2-3=z
```

The Equations Window functions a lot like a word processor. You can use the cut (Ctrl+X), copy (Ctrl+C), paste (Ctrl+V), and undo (Ctrl-Z) commands just as you would in Microsoft Word® or other programs that manipulate text. Equations are typically entered one per line; however, multiple equations can be entered on the same line provided that they are separated by a semicolon (or a colon if you are using European format):

```
y=z-4; x+y=3; x^2-3=z
```

The mathematical operators and order of operation used in the equations are consistent with those used by most any programming language. For example, the equation:

```
a=2^3+3*4+2
```

will result in $a = 22$. Variable names in EES must start with a letter but they can include any U.S. keyboard character except ( ) * / + - { } or ;. EES is case-insensitive; that is, the symbols x and X are interpreted to be the same variable in the Equations Window. Very long equations can be broken into two lines for ease of reading or improved printing by using the ampersand (&). Our example doesn't include any long equations, but we could still break the first equation into two lines according to:

```
x+y&
=3
y=z-4
z=x^2-3
```

Comments

It is good practice to annotate your EES code in order to clarify the meaning behind the equation set. Typically each equation is followed by a comment that is ignored by the equation solver itself but is visible to the user. Comments in the Equations Window should be enclosed either in curly braces or quotes. Any line that begins with two forward slashes is also considered a comment. The comments are displayed in blue (by default) in the Equations Window to indicate that they are not part of the equation set that will be solved. Comments that begin with the ! character (referred to as comment type 2) are displayed in red by default. The default colors for comments can be changed in the Preferences dialog (Options menu).

```
//this is a set of equations

x+y=3                                   {comment for line 1}
y=z-4                                   "comment for line 2"
z=x^2-3                                 "! this is a type 2 comment"
//this is a comment
```

Occasionally it will be useful to temporarily remove an equation or a group of equations from the Equations Window. Of course, you could just delete the equation(s) and then type them back in if and when they are needed again. However, a better solution is to "comment them out". Highlight the equation(s) to be removed and then right-click to bring up a pop-up menu, as

shown in Figure 1-11. Select Comment {} in order to place curly brackets around the entire section of highlighted code, temporarily removing it from the equation set. To restore the equation(s), highlight the commented code, right-click and select Undo Comment {}.

Figure 1-11: Commenting out the last two equations.

Active hypertext can also be entered in comments in the Equations Window. EES will automatically recognize hypertext links that begin with http:\\, https:\\, or file:. For example, enter:

```
"http:\\fchart.com"
```

and notice that it becomes a link to that web page. A summary of the recognized hypertext links is provided in Table 1-1.

Table 1-1: Summary of recognized hypertext links.

Hypertext link	Description
http:\\fchart.com	Open the default browser and point it at the web page that follows http:
https:\\fchart.com	Open the default browser and point it at the web page that follows https:
file:c:\ees32\ees_manual.pdf	Open the file that follows file: and start the appropriate application.
\\EES_Solution	Open the Solution Window and bring it to the front.
\\EES_Format	Open the Formatted Equations Window and bring it to the front.
\\EES_Plot	Open the Plot Window and bring it to the front.
\\EES_PlotN	Open Plot Window N where N is an integer and bring it to the front.
\\EES_Parametric	Open the Parametric Table and bring it to the front
\\EES_Lookup	Open the Lookup Table and bring it to the front.
\\EES_Array	Open the Array Table and bring it to the front.
\\EES_Integral	Open the Integral Table and bring it to the front.
\\EES_Report	Open the Report Window and bring it to the front.
\\EES_Diagram	Open the Diagram Window and bring it to the front.
\\EES_Residual	Open the Residuals Window and bring it to the front.
\\EES_Calculator	Open the Calculator Window and bring it to the front.
\\EES_Solve	Solve the current set of equations.
\\EES_SolveTable	Apply the Solve Table command from the Calculate menu.
\\EES_MinMax	Apply the Min/Max command from the Calculate menu.
\\EES_MinMaxTable	Apply the Min/Max Table command from the Calculate menu.

Built-In Math Functions

EES provides many built-in functions. The built-in math functions provide basic mathematical, trigonometric, statistical capability as well as commands to operate on data files and tables. A complete list and description of all built-in math functions is provided in Appendix A.

The built-in math functions can be conveniently accessed by selecting Function Information from the Options menu, which displays the Function Information dialog appearing in Figure 1-12. By default, all the built-in math and string functions are initially shown in the list on the left side of the dialog. The list of functions can be reduced by selecting a function classification from the list on the right side. An example of the proper use of the selected function is shown in the Ex: edit control at the bottom of the dialog. The text in the Ex: edit control can be edited in the conventional manner. Clicking the Paste button will paste the contents of this control into the Equations Window at the current cursor position.

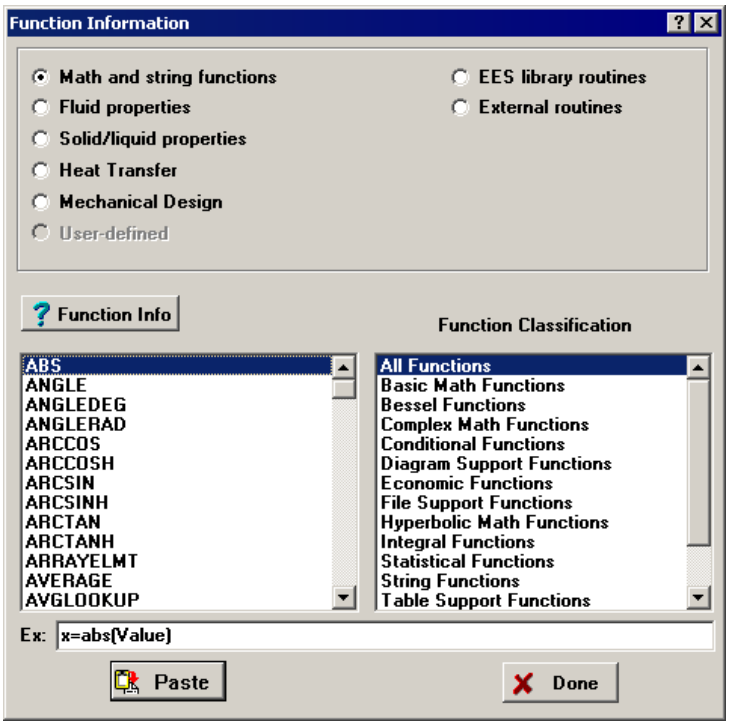


Figure 1-12: Function Information dialog showing math and string functions.

String Variables

EES provides both numerical and string variables. A string variable is identified with a variable name that ends with the $ character. The variable name must begin with a letter and consist of 30 or fewer characters, including the $ character. String variables can be set to string constants, which are strings that enclosed with single quote marks.

```
A$ = 'carbon dioxide'
```

String variables can be used in EES equations anywhere in which character information is provided. For example the name of a fluid provided to a thermophysical property function may

be a string variable, as shown in Section 4.3. A string variable may be used to supply the units of other variables, as shown in Section 1.5. String variables may be passed as arguments to internal functions, procedures, modules, and subprograms or to external functions and procedures as described in Chapters 3 and 10. String variables are very useful in the Diagram window, as shown in Chapter 15. String functions are provided to manipulate string variables and to convert them to or from numerical values. A complete list of string functions is provided in Appendix B. String functions can be accessed from the Function Information dialog, as shown in Figure 1-12.

The $TabStops Directive

Directives are instructions to EES that are usually executed before the equations are compiled and solved. Directive instructions are preceded by the dollar sign ($) and provide a powerful method for controlling an EES solution. Chapter 14 discusses the use of directives in detail and a complete list of directives is provided in Appendix C. However, we will make use of various directives throughout the text. The $TabStops directive provides a mechanism for controlling the location of the tab stops used in the Equations Window. The protocol for using the $TabStops directive is shown below:

```
$TabStops TabStop1  TabStop 2 ... TabStop5  Units
```

where TabStop1, TabStop2, etc. are numerical values corresponding to the desired positions of the tabs and Units indicates the units used for the tabs. The value of Units must either be in (for inch) or cm (for centimeter). The code below sets two tab stops at 1 inch and 2 inch; notice that the locations of the comments have shifted to 1 inch, which is consistent with the first tab stop set in the $TabStops directive.

```
$TabStops 1 2 in

//this is a set of equations

x+y=3          {comment for line 1}
y=z-4          "comment for line 2"
z=x^2-3        "! this is a type 2 comment"
```

Showing Values and Setting Variable Units in the Equations Window

Selecting a variable in the Equations Window will cause a small hint window to appear just below the variable. The hint window will show the value of the variable, if calculations have been completed, and variable units. The variable can be easily selected by double-clicking the left mouse button when the cursor is positioned over the variable. This capability makes it unnecessary to refer to the Solutions window to determine values of EES variables. After a variable has been selected, right-clicking will bring up a pop-up menu that has Variable Info as one of the menu items. Selecting Variable Info will provide a small dialog window in which the format and units of the variable can be entered. This is perhaps the easiest way to enter the units of a variable.

The Status Bar

The status bar is located at the bottom of the Equations Window, as shown in Figure 1-13.

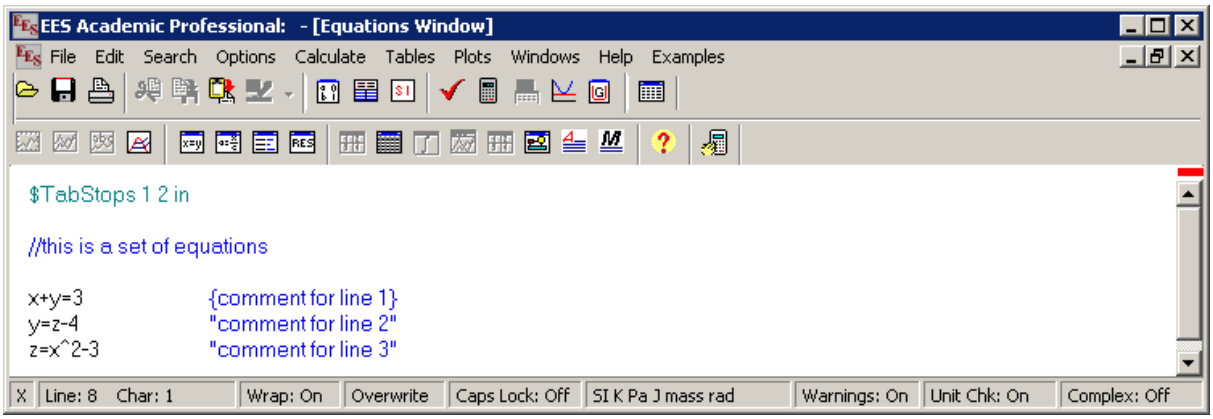
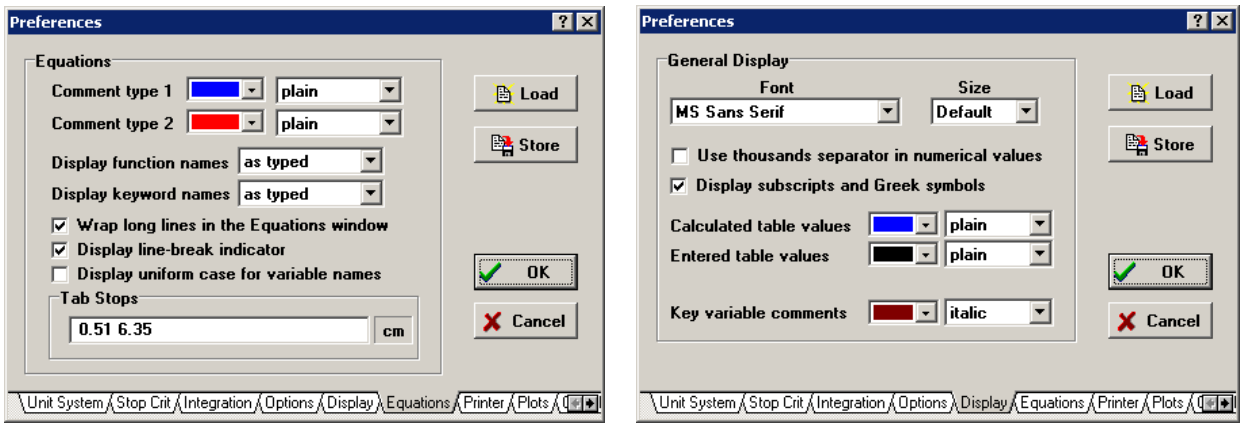


Figure 1-13: Status bar.

Selecting the X at the left side of the status bar deactivates it. The next panel displays the line and character on that line at which the cursor is positioned. The various values of the current settings are displayed in subsequent panels on the status bar, including the word wrap, overwrite, and caps lock settings. Clicking the mouse button with the cursor positioned in any of these panels allows the settings to be changed. The current unit system (discussed in Section 1.5) is displayed. The last three panels indicate whether warnings, unit checking, and complex algebra are currently activated.

Equations and Display Preferences

Many of the characteristics of the Equations Window can be altered using the Preferences dialog accessed by selecting Preferences from the Options menu. Select the Equations tab to bring up the dialog shown in Figure 1-14(a), which allows you to change the colors used to indicate the two comment types, the format used to display functions, and other display options. Select the Display tab to bring up the dialog shown in Figure 1-14(b), which allows you to change the font and size as well as other aspects of the format used for display. The Preferences dialog has many such tabs that can be used control various features of your EES programs.



(a) **(b)**

Figure 1-14: (a) Equations and (b) Display pages of the Preferences dialog.

The Formatted Equations Window

The equations can also be viewed in an easy to read, mathematical notation by selecting Formatted Equations from the Windows menu. The result is shown in Figure 1-15. Notice that, by default, the comments enclosed by quotations are shown in the Formatted Equations Window while those enclosed by curly brackets are not shown. All comments can be turned off by right clicking in the Formatted Equations Window and de-selecting the Display Comments option from the pop-up menu.

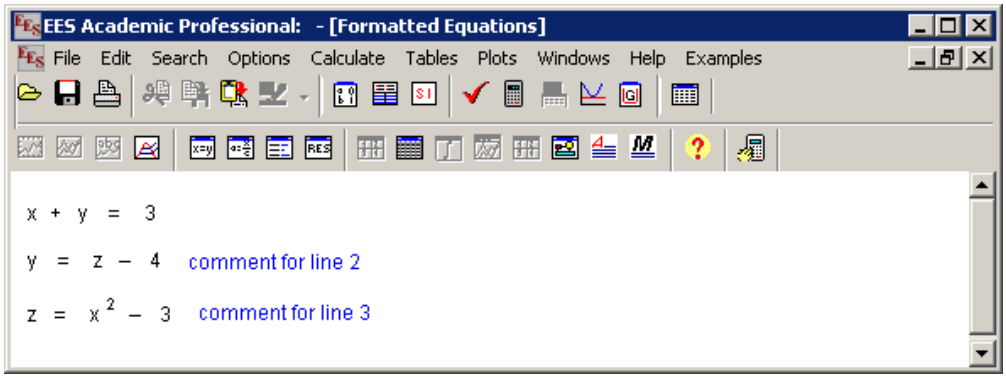


Figure 1-15: Formatted Equations Window.

The Formatted Equations window is particularly useful for viewing large equations with multiple parentheses. For example, consider the equation:

$$\overline{f} = \frac{4}{Re_D}\left[\frac{3.44}{\sqrt{L^+}} + \frac{\frac{1.25}{4L^+} + \frac{64}{4} - \frac{3.44}{\sqrt{L^+}}}{1 + \frac{0.00021}{\left(L^+\right)^2}}\right] \tag{1-10}$$

Equation (1-10) is the correlation provided by Shah and London (1978) for the average friction factor associated with developing flow in a round tube. The symbol Re_D is the Reynolds number and L^+ is the dimensionless length of the tube. This correlation can be entered in the Equations Window as shown below. Note that Chapter 12 discusses the EES heat transfer library which implements this and other useful correlations automatically.

```
Re_D=1000                                   "Reynolds number"
L|plus=10                                   "dimensionless length"
f_bar=(4/Re_D)*(3.44/sqrt(L|plus)+(1.25/(4*L|plus)+64/4-3.44/sqrt(L|plus))/(1+0.00021/L|plus^2))
    "average friction factor"
sigma=1                                     "a Greek symbol"
```

The equation for $\overline{f}$ is difficult to read or debug in the word processing format used by the Equations Window. It is much easier to view in the Formatted Equations Window, as shown in Figure 1-16.

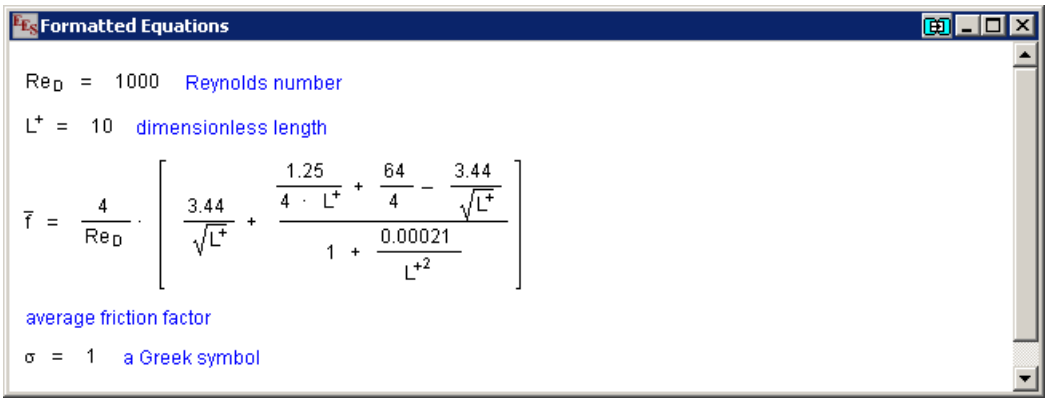


Figure 1-16: Formatted Equations Window.

Notice that the symbol f_bar is interpreted as f with an overbar in the Formatted Equations Window. Similarly Re_D is interpreted as Re with a subscript D and L|plus is interpreted as L with a superscript +. Greek symbols are also interpreted correctly; notice that the variable sigma is interpreted as σ. The formatting options that are available are summarized in Table 1-2. Section 1.6 shows how equations in the Formatted Equations Window can be copied in various formats in order to facilitate writing reports or papers.

Table 1-2: Variable display options for Formatted Equations Window.

Description	Method	Example of variable in Equations Window	Result in the Formatted Equations Window
Overbar	append _bar to the variable	X_bar	$\overline{X}$
Tilde	append _tilde to the variable	X_tilde	$\tilde{X}$
Hat	append _hat to the variable	X_hat	$\hat{X}$
Dot	append _dot to the variable	X_dot	$\dot{X}$
Double dot	append _ddot to the variable	X_ddot	$\ddot{X}$
Single prime	append _prime to the variable	X_prime	X'
Double prime	append _dprime to the variable	X_dprime	X''
Subscript	use the _ to start the subscript	X_a	X_a
Superscript	use the | to start the superscript	X|a	X^a
Infinity symbol	spell out infinity	X_infinity	X_∞
Greek symbol	spell out the name of the symbol	delta	δ
		Delta	Δ

The Splitter Bar

In a large EES program, you may also find it convenient to be working on one section of the code while viewing another. Professional versions of EES provide this capability using the splitter bar. The splitter bar divides the vertical scroll bar into two sections. The position of the splitter bar is indicated by the red rectangle on the vertical scroll bar. Initially, the splitter bar is at its uppermost position, indicating that the vertical scroll bar has only one section. To see how the splitter bar works, select EES Example Problems from the Examples menu and then select Psychrometric functions. Select the EES example named Using psychrometric functions in a supermarket model. This example is a relatively large EES program and it is possible that you

may want to use the splitter bar to work on it. Select the red rectangle and adjust its position to create two sections, as shown in Figure 1-17.

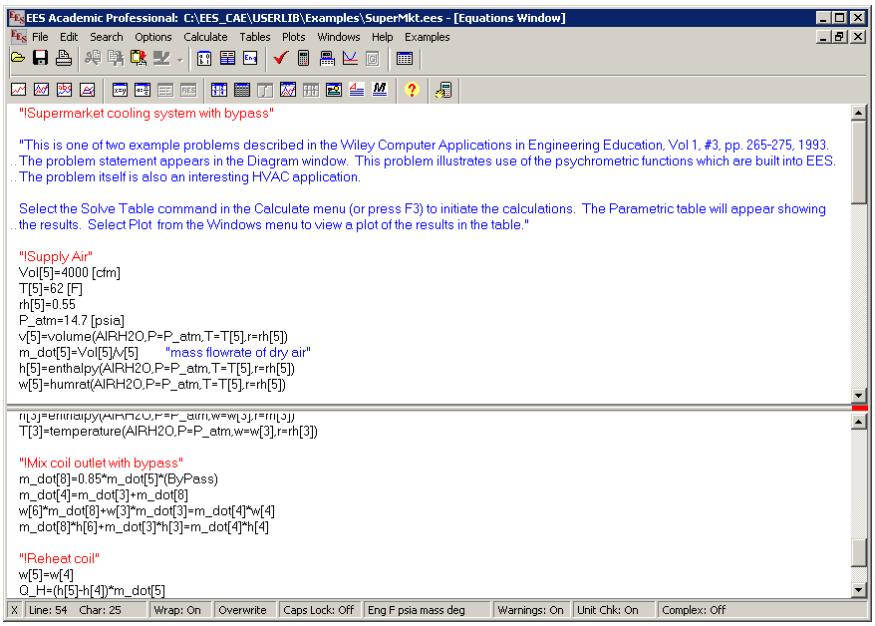


Figure 1-17: Splitter bar used to create two sections.

Password Protection

In the Professional version of EES, it is possible to password protect the Equations Window so that only you can edit it. Right click in the Equations Window and select Protected from the resulting pop-up menu. You will be prompted to enter and then re-enter a password, after which the background color will change to indicate that the text is now read only. Any subsequent attempt to edit the protected regions of the Equations Window will not be allowed until you unprotect the text. The text can be unprotected by right clicking on it and selecting Protected, at which time you must enter the password.

Key Variables

The Solution Window in a large EES program will contain many variables. Variables are listed in alphabetical order; however, it may be difficult to quickly and easily identify the small subset of variables that are of particular interest. To focus attention on a particular variable, it is possible to highlight its appearance. For example, select the variable savings in the Solution Window and right-click on it. The Format Selected Variables dialog that appears is shown in Figure 1-18.

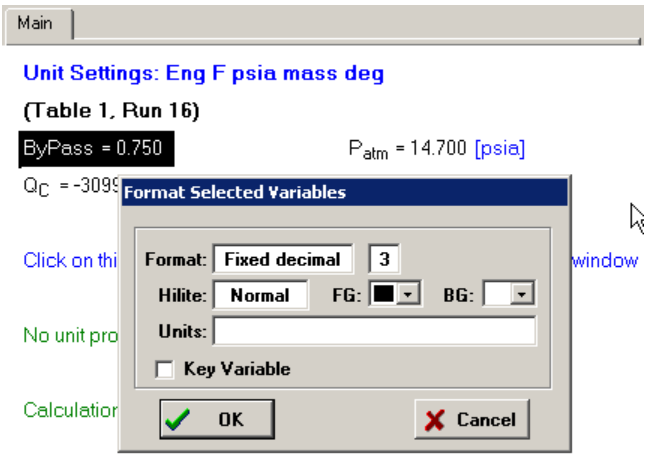


Figure 1-18: Format Selected Variables dialog.

The Format Selected Variables dialog allows you to change the formatting that is used to display the variable in the Solutions Window. For example, you can box the variable and change its background color. You can also select a subset of all of the variables in your Solution Window and make them key variables. Highlight the variable of interest and select Key Variable in the resulting Format Selected Variable dialog in order to cause a box to appear in which you can write some descriptive comments about the variable, as shown in Figure 1-19. Select OK and you will see that a new tab will appear in the Solutions Window allowing access to the Key Variables Window. All of the variables that have been identified as key variables are placed in the Key Variable Window together with their associated descriptive comments, as shown in Figure 1-20. The position of each line in the Key Variable Window can be changed by pressing and holding the mouse button down on the desired line, while moving the mouse to the new location.

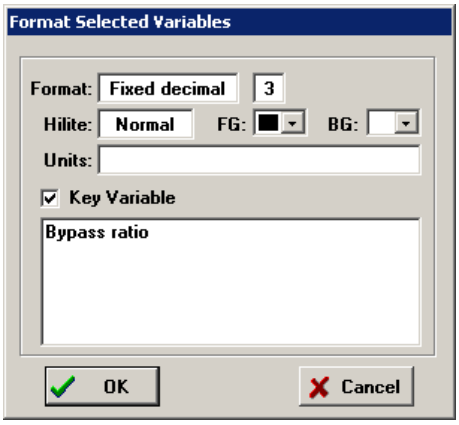


Figure 1-19: Key Variable.

Figure 1-20: Key Variables Window.

1.3 Parametric Tables

Section 1.2 described how to use EES to obtain a single solution to a set of equations. It is often interesting to run a parametric study in which the effect of one variable (the independent variable) on another variable (the dependent variable) is studied. For example, let's estimate the pressure loss incurred by the flow of water through a 45° elbow as a function of the flow rate. The inner diameter of the elbow is D = 2 cm and the density of water is ρ = 1000 kg/m^3. The dimensionless resistance coefficient for this type of elbow is estimated to be K = 0.3. The pressure loss can be estimated according to:

$$\Delta P = K \frac{\rho u^2}{2} \tag{1-11}$$

where u is the velocity of the water. The inputs are entered in the Equations Window; note that we'll start with $\dot{V}$ = 100 liter/min and subsequently vary this value.

```
$TabStops 2.5 in

"Inputs"
K=0.3                    "K factor for a 45 degree elbow, dimensionless"
V_dot=100                "volumetric flow rate, in liter per min"
D=2                      "inner diameter of elbow, in cm"
rho=1000                 "density of water, in kg/m^3"
```

In Section 1.5, we will discuss how EES can help you deal with unit assignments and conversions. For now we'll have to keep track of units by manually. Notice that the units assigned to each variable are indicated in the associated comment. The cross sectional area of the elbow is given by:

$$A_c = \pi \frac{D^2}{4} \tag{1-12}$$

```
A_c=pi#*D^2/4            "cross-sectional area for flow, in cm^2"
```

The units of the variable A_c must be cm^2 (because the units of the variable D is cm). Also, the variable pi# is a built-in constant in EES corresponding to the value of π (other built-in constants are discussed in Section 1.9). The velocity of the water is:

$$u = \frac{\dot{V}}{A_c} \tag{1-13}$$

```
u=V_dot/A_c              "velocity, in liter/min-cm^2"
```

As written, the units for the variable u must be liter/min (the units for the variable V_dot) divided by cm^2 (the units for the variable A_c). Therefore, the units for u must liter/min-cm^2. Note that

in this book, the hyphen sign will be used to indicate multiplication on one side of the divisor in the context of units; therefore liter/min-cm^2 should be read as $\frac{\text{liter}}{\text{min cm}^2}$. EES uses the same convention for unit specifications, although the hyphen can be replaced with a space or a dot (Alt-250 on a U.S. keyboard).

The units for the variable u, as calculated, don't make a lot of sense; let's convert the units to cm/min, which are more reasonable:

$$\frac{\text{liter}}{\text{min-cm}^2}\left\|\frac{1000\ \text{cm}^3}{1\ \text{liter}}=\frac{\text{cm}}{\text{min}}\right. \tag{1-14}$$

The EES code, with the unit conversion, is:

```
u=(V_dot/A_c)*1000                    "velocity, in cm/min"
```

Finally, Eq. (1-11) is used to compute the pressure loss.

```
DeltaP=K*rho*u^2/2                    "pressure loss, in kPa"
```

Again, the units for the variable DeltaP don't make a lot of sense; as calculated, the units are kg-cm^2/m^3-min^2. Nobody will understand your result when expressed in these units. Let's convert to kPa, which are more commonly used:

$$\frac{\text{kg-cm}^2}{\text{m}^3\text{-min}^2}\left\|\frac{\text{kPa-m}^2}{1000\ \text{N}}\right|\frac{\text{N-s}^2}{\text{kg-m}}\left|\frac{\text{m}^2}{100^2\ \text{cm}^2}\right|\frac{\text{min}^2}{60^2\ \text{s}^2}=\text{kPa} \tag{1-15}$$

Notice that we had to look up four separate unit conversion factors in order to accomplish the unit conversion indicated by Eq. (1-15). Section 1.5 shows that EES greatly reduces the hassle and potential errors that are associated with working with units.

```
DeltaP=(K*rho*u^2/2)/(1000*100^2*60^2)    "pressure loss, in kPa"
```

Solving leads to ΔP = 4.22 kPa for $\dot{V}$ = 100 liter/min.

Creating a Parametric Table

In order to parametrically vary the volumetric flow rate (the independent variable) and study its effect on the pressure loss (the dependent variable) we need to create a Parametric Table. Select New Parametric Table from the Tables menu. The New Parametric Table dialog will appear, as shown in Figure 1-21.

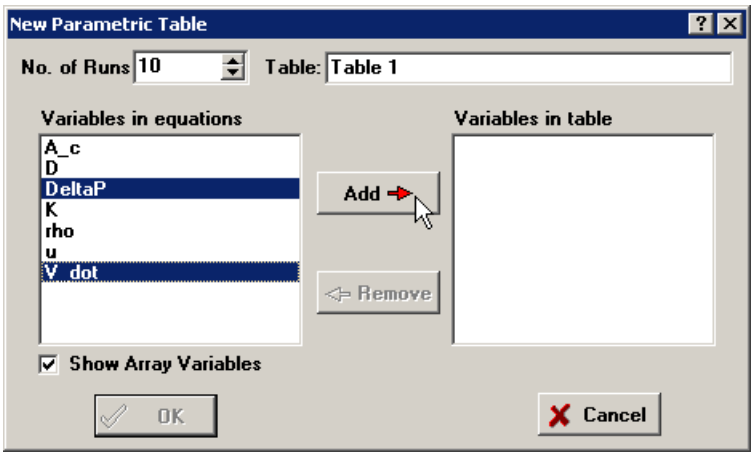


Figure 1-21: New Parametric Table dialog.

The window on the left side of the dialog provides a list of all of the variables that are included in the Equations Window. Highlight the independent and dependent variables of interest (in this case, the variables V_dot and DeltaP) by clicking on them and then select Add to add the variables to the window on the right side, which lists variables to be included in the table. Any other variables that you wish to include in the table can also be added. You can select the variables one at a time or all at once. Select OK in order to create the Parametric Table, shown in Figure 1-22. By default, there are 10 runs (rows) in the table. Runs can be added or removed by selecting Insert/Delete Runs from the Tables menu or by right-clicking on any run number and selecting Insert Runs or Delete Runs. There is a column for each of the variables included in the table. Columns can be added or deleted by selecting Insert/Delete Vars from the Tables menu or right clicking on any variable and selecting Insert Column or Delete.

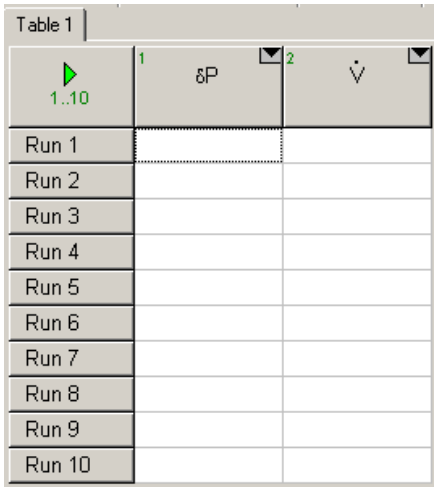


Figure 1-22: Empty Parametric Table.

Alter Values

To carry out a parametric study in which the volumetric flow rate is changed, it is necessary to fill in the column for the variable V_dot with flow rate values that are of interest. This process can be done manually, one run at a time. More typically, you will right-click on the column of interest and select Alter Values or click on the triangular icon in the column header to bring up the dialog shown in Figure 1-23.

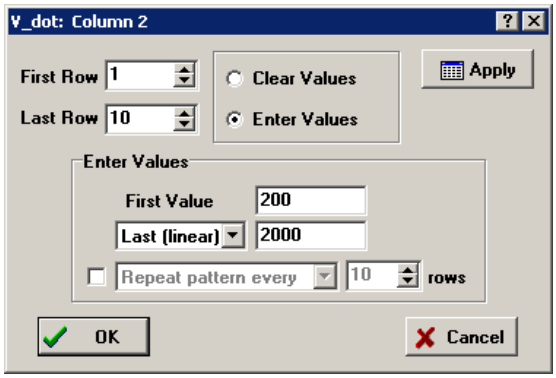


Figure 1-23: Alter values dialog.

Select the rows to be filled in (rows 1 to 10, for this example) and the pattern to be used to enter the values. In Figure 1-23, the dialog is filled in so that the volumetric flow rate varies from 200 liter/min (in row 1) to 2000 liter/min (in row 10) in equally spaced intervals, as shown in Figure 1-24. Other options can also be specified by clicking on the drop down menu under First Value. For plots that will use a logarithmic scale, the selection Last (Log) is particularly useful as it logarithmically spaces the values in the table.

Table 1

1..10	1 δP	2 V̇
Run 1		200
Run 2		400
Run 3		600
Run 4		800
Run 5		1000
Run 6		1200
Run 7		1400
Run 8		1600
Run 9		1800
Run 10		2000

Figure 1-24: Parametric Table with values of V_dot set.

Solving a Parametric Table

The Parametric Table is solved and filled in using the Solve Table command from the Calculate menu (or by pressing F3). Solving the table prompts EES to solve one or more of the runs in the table. EES will begin with the first run that is specified and look in the table to see which columns have specified values. It will then specify the value of these variables in the Equations Window, solve the resulting system of equations, and fill in the values of the remaining columns based on the solution. Select Solve Table to bring up the Solve Table dialog shown in Figure 1-25.

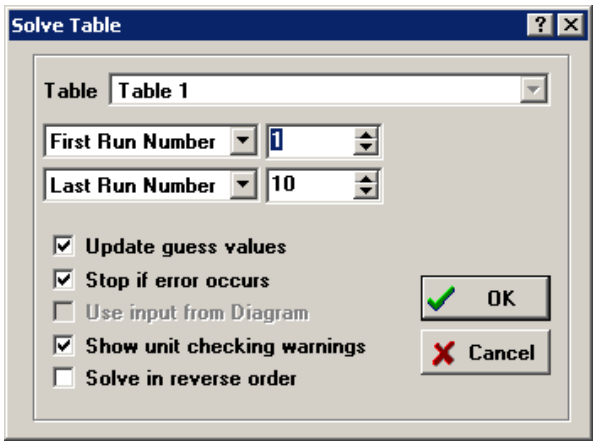


Figure 1-25: Solve Table dialog.

The dialog is initially set so that EES will start with run 1 and end with run 10. By pressing OK, EES will go to run 1 of the Parametric Table shown in Figure 1-24 and find that the value of the variable V_dot is set to 200. Therefore, you can imagine that EES places the following in the Equations Window.

```
V_dot=100
```

Of course, this presents a problem because the statement:

```
V_dot=100                    "volumetric flow rate, in liter per min"
```

already exists in the Equations Window. The variable V_dot cannot have multiple values as it causes the equation set to be over-constrained (8 equations and 7 variables). Solving the table will result in the error dialog shown in Figure 1-26.

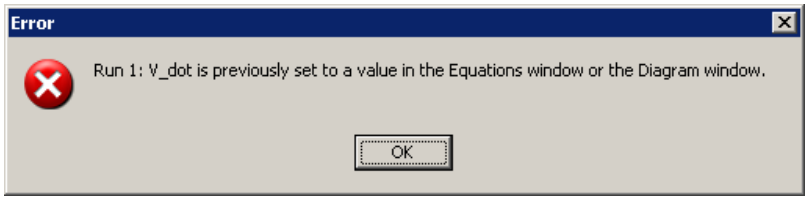


Figure 1-26: Error dialog.

This problem can be alleviated by commenting out the equation that specifies the volumetric flow rate in the Equations Window. Highlight the statement, right-click, and select Comment {} in order to temporarily remove it from the equation set.

```
{V_dot=100                    "volumetric flow rate, in liter per min"}
```

Now solve the table to obtain the solved Parametric Table shown in Figure 1-27.

Table 1

1..10	1 δP	2 V̇
Run 1	16.89	200
Run 2	67.55	400
Run 3	152	600
Run 4	270.2	800
Run 5	422.2	1000
Run 6	607.9	1200
Run 7	827.5	1400
Run 8	1081	1600
Run 9	1368	1800
Run 10	1689	2000

Figure 1-27: Solved Parametric Table.

It will sometimes be useful to solve only a portion of the table. This is easily accomplished by changing the first and last run numbers in the Solve Table dialog.

In the Professional version of EES, the start and stop values can be set to variables. For example, add the lines:

```
startrow=3                    "first run"
endrow=7                      "last run"
```

Open the Parametric Table, right click on the DeltaP variable column and select Alter Values. Select Clear Values to clear the column. Solve the table again. This time, click on the arrow next to First Run Number and change it to First Run Variable and then select startrow. Change Last Run Number to Last Run Variable and select endrow. The result is shown in Figure 1-28. Select OK and you will see that only runs 3 through 7 have been filled in.

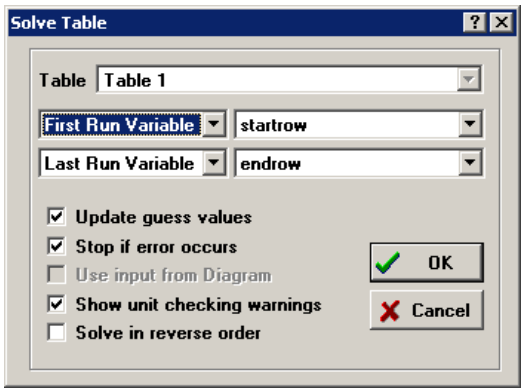


Figure 1-28: Solve Table dialog with first and last runs set with variables.

Formatting Columns

The Parametric Table shows the value of pressure drop for each corresponding value of volumetric flow rate. In Section 1.4 we will see how to use this information to prepare a plot. Before we do that, let's look at some other features of Parametric Tables. The position of the columns can be adjusted by dragging (i.e., pressing the left mouse button and then moving mouse while holding down the mouse button) the column headers. The format of the columns can be adjusted by right-clicking on the column and selecting Properties to bring up the Format Properties Table dialog shown in Figure 1-29 (for the column corresponding to the variable DeltaP). The numerical format used to display the results, the background color, and the column width can also be adjusted. The column number can also be set in this dialog. At the bottom of the dialog, various statistics are provided for the entries in the column.

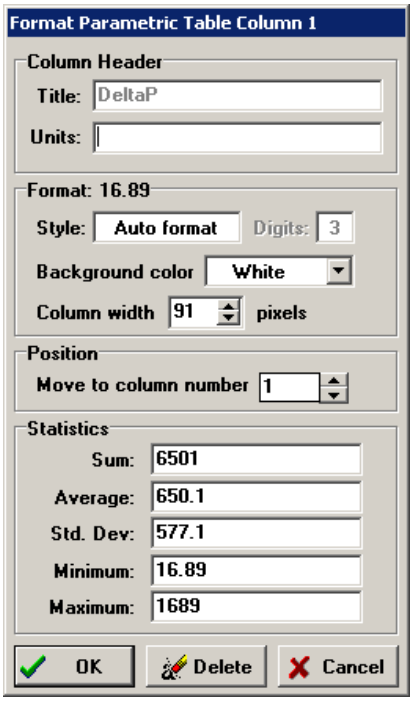


Figure 1-29: Format Properties Table dialog.

The $If, $IfNot, $Else, and $EndIf Directives

In order to run the Parametric Table we had to comment out the line in the Equations Window that specified by the value of the variable V_dot. This change temporarily removes the specification; it can be returned by highlighting the text, right clicking, and selecting Undo Comment {} from the pop-up menu.

```
V_dot=100                                   "volumetric flow rate, in liter per min"
```

A more elegant method for removing one or more lines of code when a Parametric Table is being solved is to use the $If directive. The $If directive is used according to:

```
$If Condition
    line(s) of code to be executed if Condition is true
$Else
    line(s) of code to be executed if Condition is false
$EndIf
```

where Condition is a keyword that indicates an execution condition. There are many such keywords recognized by EES and these are discussed in more detail in Chapter 18. One keyword is ParametricTable, which evaluates to true when the equations are being solved from a Parametric Table. For our problem, we want to specify the value of the variable V_dot in the Equations Window only if the value of ParametricTable is false:

```
$If ParametricTable
$Else
    V_dot=100                               "volumetric flow rate, in liter per min"
$EndIf
```

Alternatively, we could use the $IfNot directive according to:

```
$IfNot ParametricTable
    V_dot=100                               "volumetric flow rate, in liter per min"
$EndIf
```

You should find that your equation set now runs if you select either Solve or Solve Table from the Calculate menu (or if you press either F2 or F3, respectively).

Naming and Documenting Parametric Tables

The Parametric Table can be renamed and annotated. Right click on the tab corresponding to the table in order to bring up the Information for Parametric Table dialog, shown in Figure 1-30. Provide a descriptive name in the Title bar (DP) and record the details of the calculation in the Description panel. The description can optionally be printed with the table.

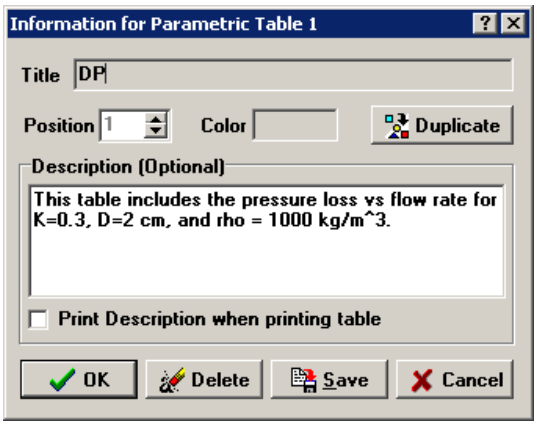


Figure 1-30: Information for Parametric Table dialog.

Saving and Loading Parametric Tables

The Parametric Table is automatically saved with the EES program and reloaded when the program is reopened. However, you can also save your Parametric Table independently by selecting Save in the Information for Parametric Table dialog, which is accessed by right-clicking on the tab name for the table. The table can be saved as a text (.txt) or comma separated values (.csv) file; either of these file formats can be opened by most any other software or spreadsheet. The table can also be saved in a format that can be reopened by a different EES program as a Lookup Table (.lkt). Lookup Tables are discussed in Section 1.8.

Copying Information from a Parametric Table

A range of cells from the table can be copied to a temporary memory location called the "clipboard" and pasted into other tables in EES or other applications, such as a word processor or a spreadsheet. For example, highlight the cells between rows 3 and 7 by clicking in upper left cell, and the pressing and holding the Shift key down while clicking in the lower right cell. Next, right click to bring up the pop-up menu shown in Figure 1-31.

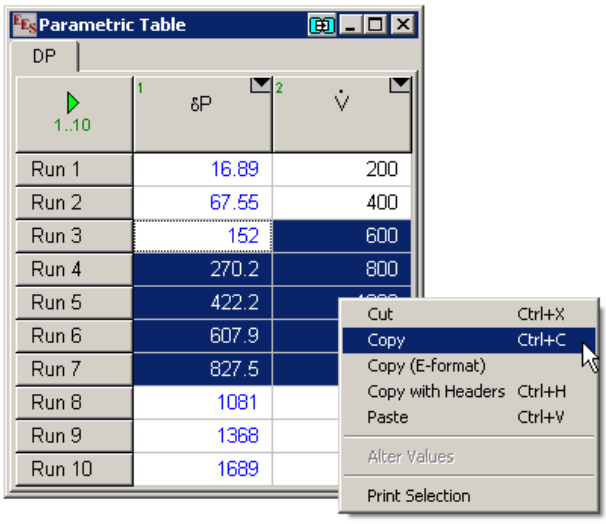
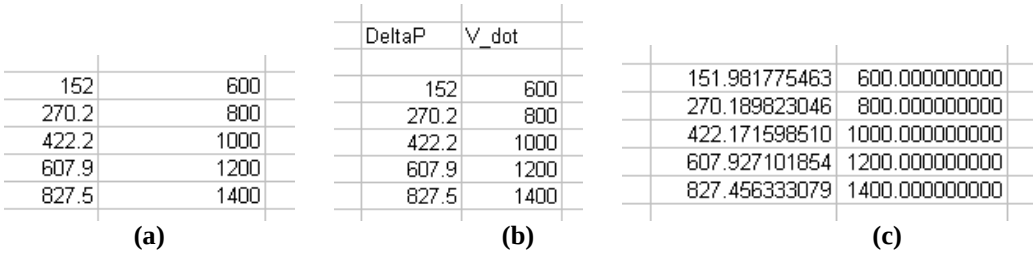


Figure 1-31: Pop-up menu.

Select Copy to place the information on the clipboard. Open another spreadsheet or word processor and then paste the information. The result is shown in Figure 1-32(a) using Excel. If the Copy with Headers option is selected then the header for each column is also copied; the result is shown in Figure 1-32(b). Notice in both cases that the number of significant figures associated with the data that is copied corresponds to the number of significant figures used to display the information in the Parametric Table. If the Copy (E-format) option is selected then the data are copied with 12 significant figures so that there is no loss of accuracy associated with the copy/paste operation, as shown in Figure 1-32(c).



152	600
270.2	800
422.2	1000
607.9	1200
827.5	1400

(a)

DeltaP	V_dot
152	600
270.2	800
422.2	1000
607.9	1200
827.5	1400

(b)

151.981775463	600.000000000
270.189823046	800.000000000
422.171598510	1000.000000000
607.927101854	1200.000000000
827.456333079	1400.000000000

(c)

Figure 1-32: Information pasted into Excel using (a) Copy, (b) Copy with Headers, and (c) Copy (E-format) commands.

1.4 Basic Plotting

Section 1.3 described how to create a Parametric Table that includes the value of a dependent variable at each of several values of an independent variable. This information is viewed most conveniently in the form of a plot. EES allows the generation of several types of plots, as discussed in Chapter 9. In this section, only the most basic X-Y type plot is discussed. Data contained in the Parametric Table created in Section 1.3 will be plotted in this section. Data contained in Lookup Tables (see Section 1.8), Arrays (see Section 1.7), and Integral Tables (see Chapter 7) can also be used for this purpose.

Generating a Plot

To generate a plot, select New Plot Window from the Plots menu in order to access the New Plot Setup dialog shown in Figure 1-33. At the top of the dialog you can provide a title (in the Tab Name field) and some descriptive text. The upper right portion of the dialog is used to select the source of the data; here it is a Parametric Table and the title of the table is DP. As shown in Figure 1-33, the New Plot dialog is set up to plot all 10 rows in the table but it is also possible to plot only a subset of the rows. The two windows in the dialog allow you to specify the independent (X-Axis) and dependent (Y-Axis) data. Figure 1-33 shows the New Plot dialog set up to plot the pressure loss as a function of the flow rate. Select OK to create the plot, which is shown in Figure 1-34.

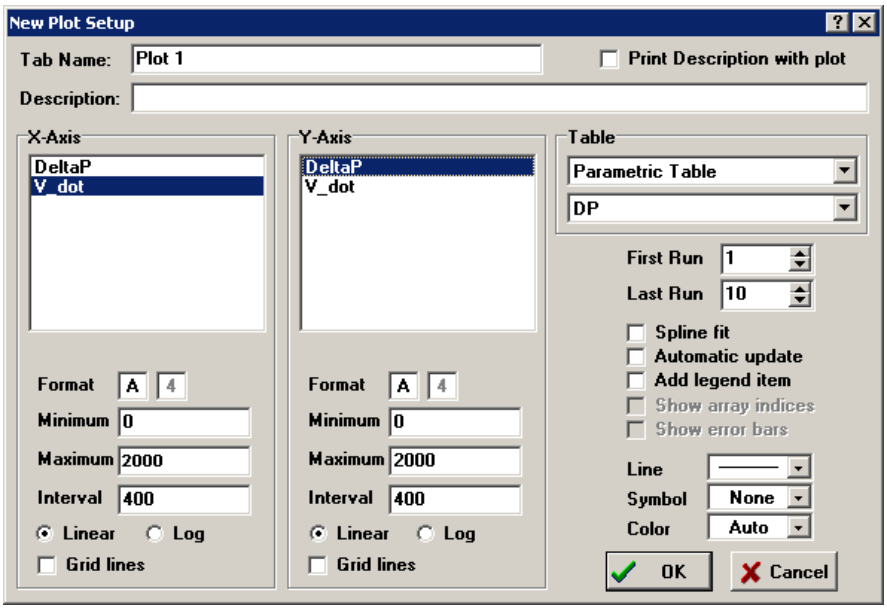


Figure 1-33: New Plot Setup dialog.

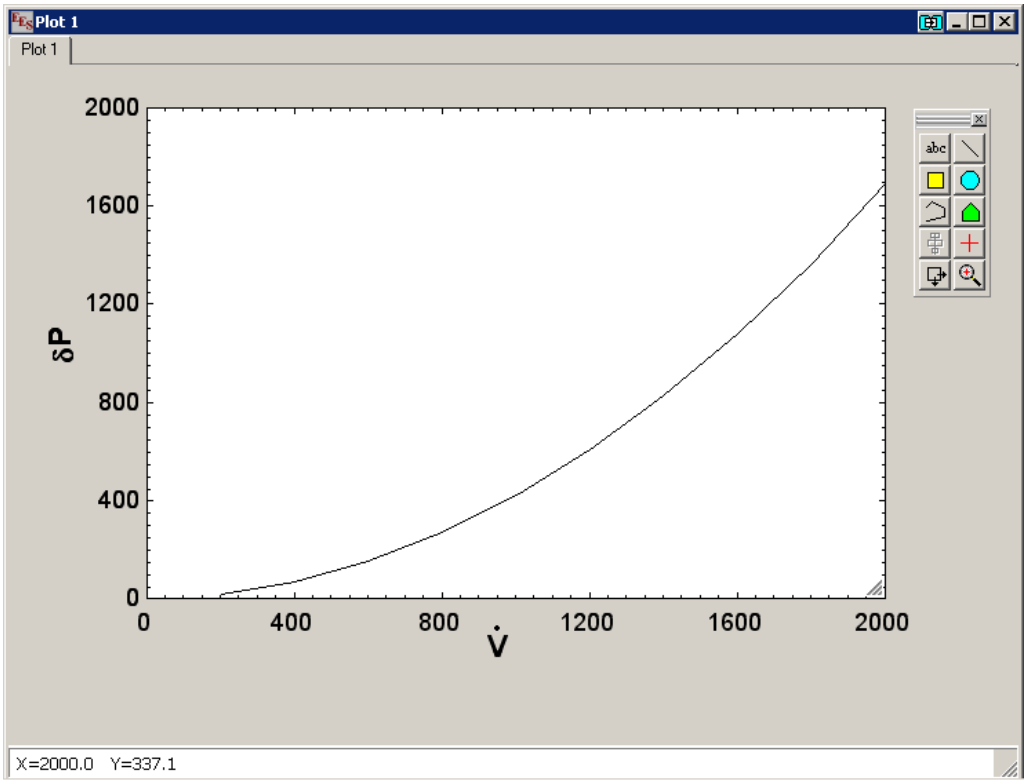


Figure 1-34: Pressure loss as a function of flow rate.

Modifying Axes

Almost every aspect of the plot can be modified in order to customize or improve it. A detailed discussion of plotting options is provided in Chapter 9. Some of the common options are described in this section.

Double-click (or right-click) on either axis label to bring up the Format Text Item dialog. You can change the axis label to a descriptive text with units and change the font. A gray triangular icon is visible at the bottom right of the plot when the plot toolbar is visible. You can resize the plot by selecting the gray triangle in the bottom right-corner and holding the mouse button down on the icon while moving the mouse to an alternative location. Note that the size of all text and graphic items in the plot changes in proportion to the size of the plot unless the Ctrl key is held down during this process. The axis scale can be adjusted by placing the mouse over any of the axes (left, right, bottom, or top) and clicking the right mouse button. This action will bring up the Modify Axis dialog shown in Figure 1-35, which allows you to make adjustments in the axis scale, add grid lines, etc. The axis starting and stopping values can also be specified by holding the left mouse button down on an axis (which will cause a double-headed arrow to appear) and then moving the mouse.

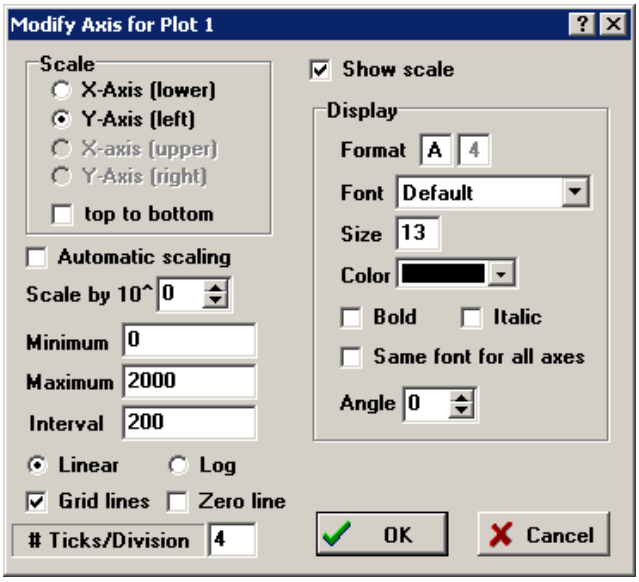


Figure 1-35: Modify Axis dialog.

An improved plot of pressure loss as a function of flow rate is shown in Figure 1-36(a). By selecting the Automatic scaling option in the Modify Axis dialog, the axis scale will be automatically adjusted to encompass the data being presented. The axis labels can be scaled by a factor of 10, 100, 1000, etc. by adjusting the Scale by 10^ option; Figure 1-36(b) shows the result of scaling the labels by 100 (10^2). Select the top to bottom option in the Modify Axis dialog in order to invert the scale, as shown in Figure 1-36(c).

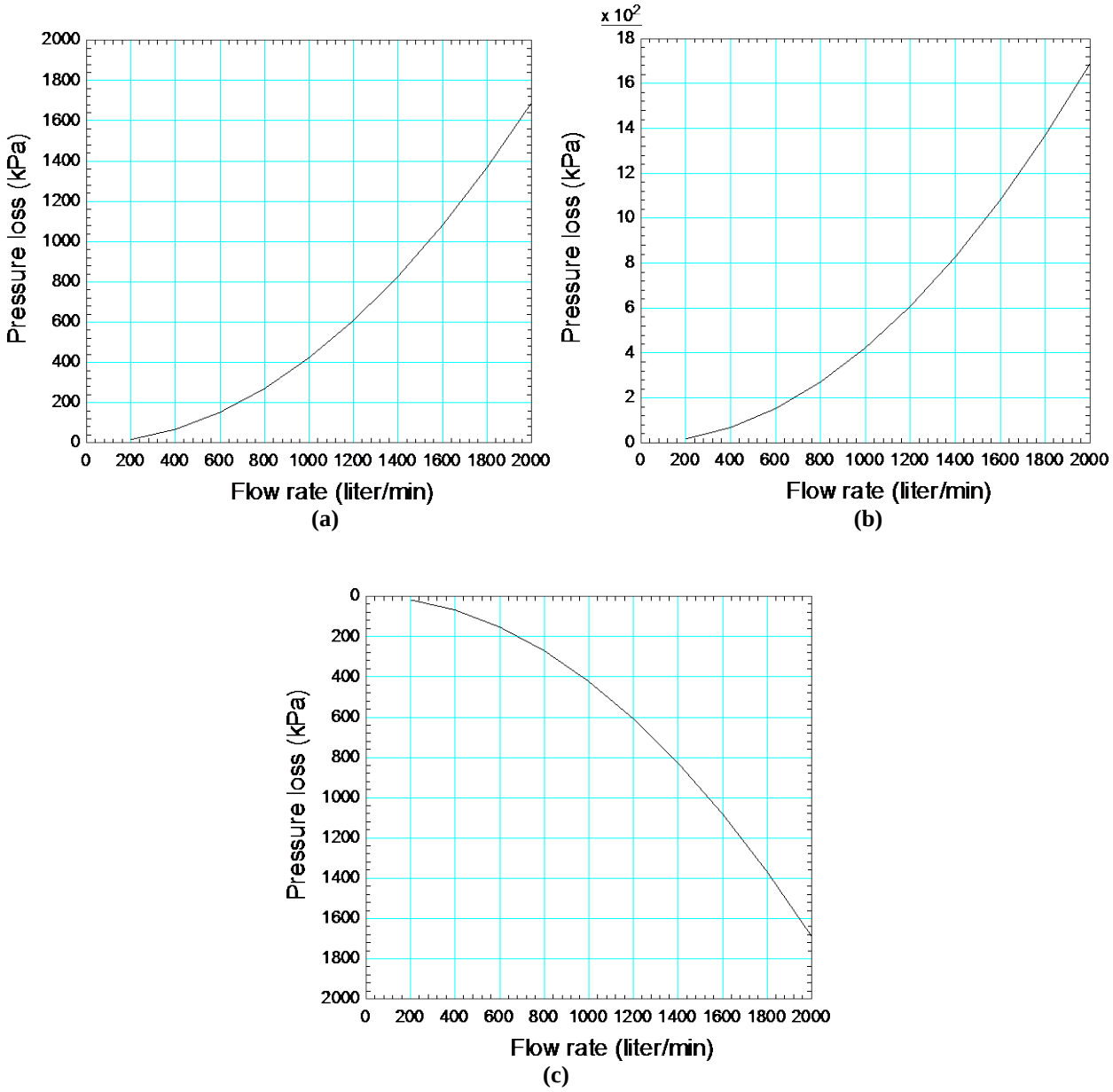


Figure 1-36: Improved plot of pressure loss as a function of flow rate (a) with default y-axis format, (b) with y-axis scaled by a factor of 10^2, and (c) with y-axis scaled from top-to-bottom.

Overlaying Plots

Multiple data series can be overlayed onto the same plot. Change the resistance coefficient to K = 0.4 and run the Parametric Table again. Select Overlay Plot from the Plots menu in order to plot pressure loss as a function of flow rate for the adjusted value of K. Your plot should now have two sets of data, as shown in Figure 1-37.

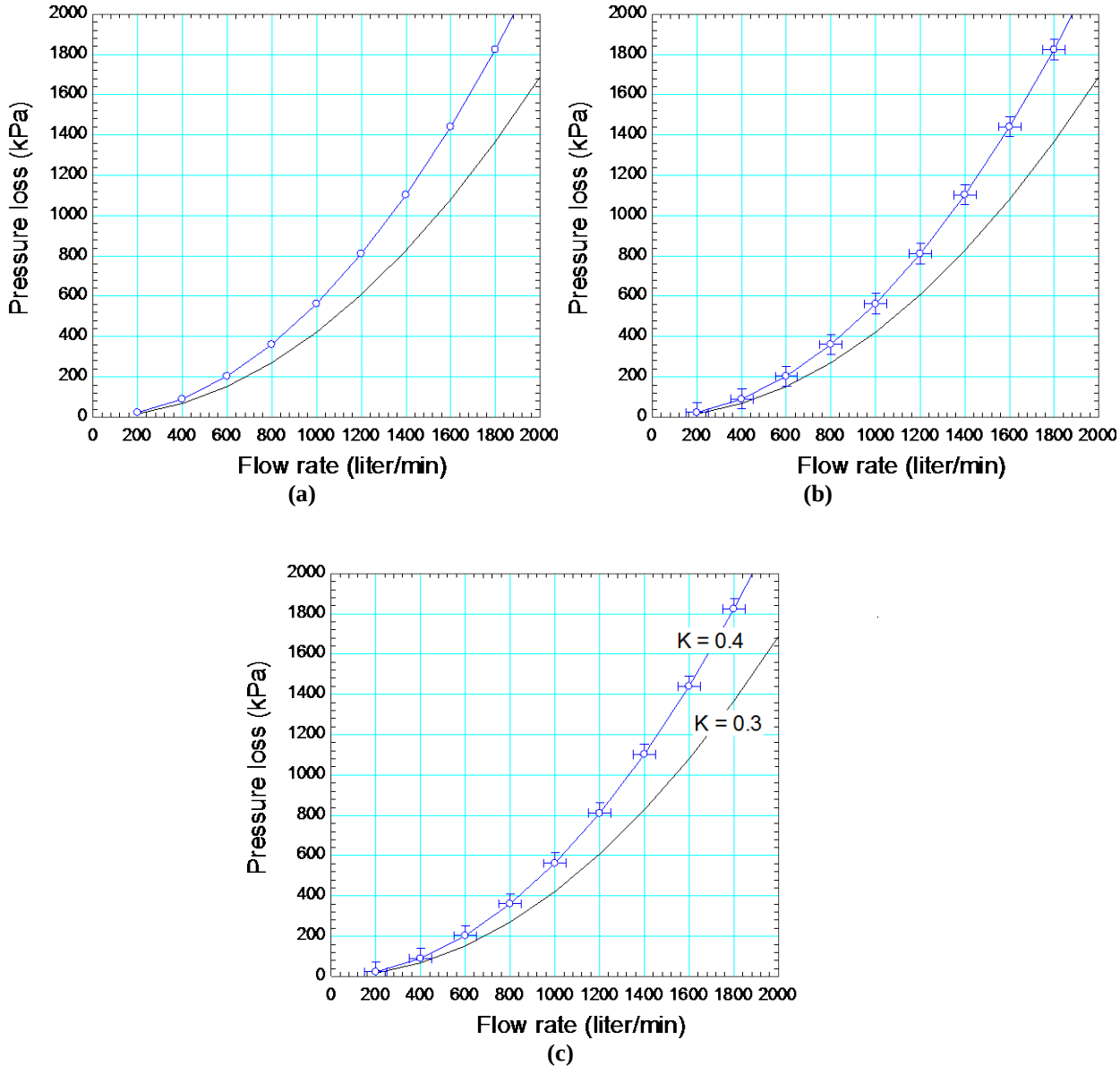


Figure 1-37: Plot with two data series (a) unmodified, (b) with error bars, and (c) with text annotations.

Modifying Plots

Double click (or right click) anywhere on the plot (other than on a text or graphic item) in order to access the Modify Plot dialog shown in Figure 1-38. This dialog allows you to add and adjust grid lines and modify the border and size of the plot. The upper window lists all of the data series that appear in the plot. You can delete one or more of these series by selecting Delete button. The underlying data set that is used in the plot to a can be extracted to a Lookup Table by selecting Get Data button. The characteristics of each data series (e.g., the line thickness, color, symbols, etc.) can be adjusted. The Smoothing option allows you to adjust the line passing through the data using either a cubic spline, polynomial fit, or moving average.

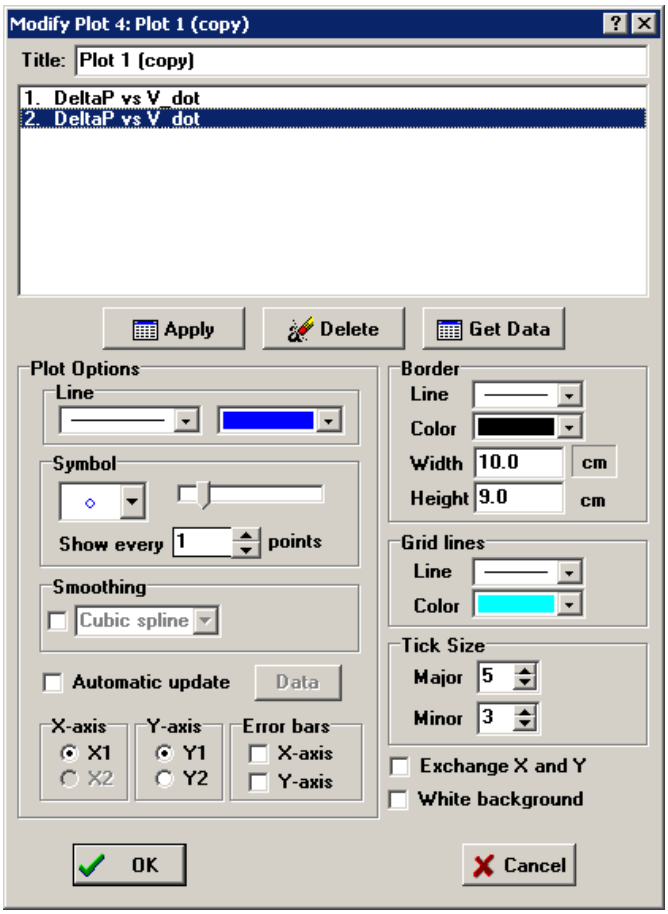


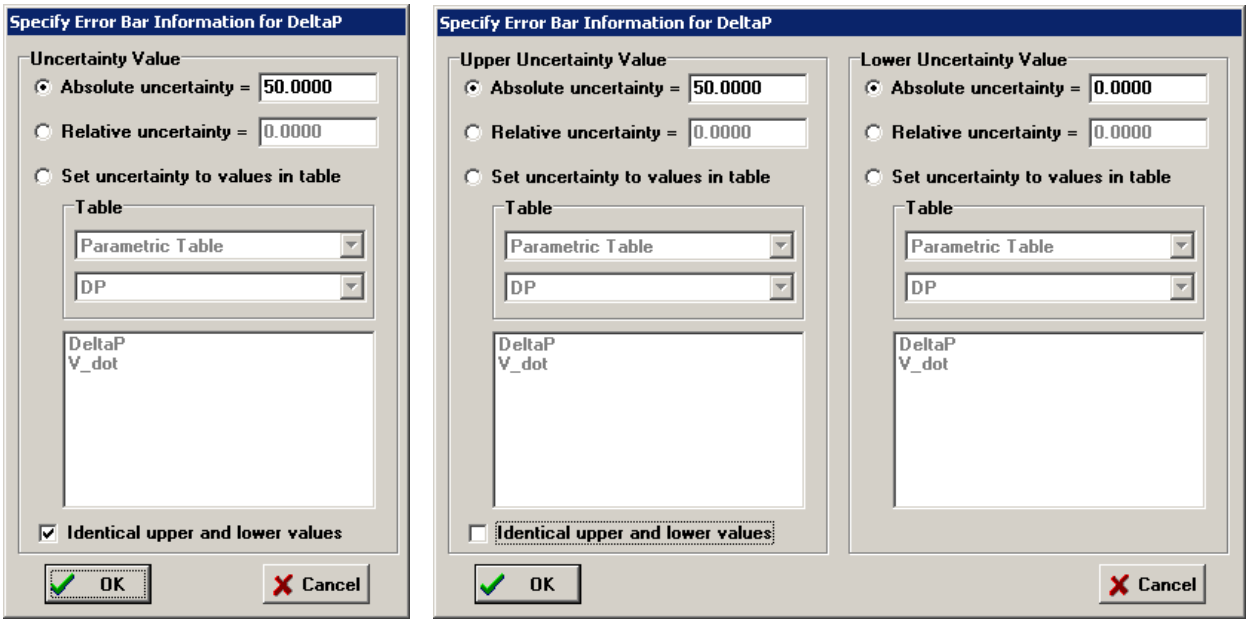
Figure 1-38: Modify Plot dialog.

Automatic Update

The Automatic update option in the Modify Plot dialog causes the data series to be re-plotted each time the data source changes (e.g., the data in the Parametric Table are adjusted). This option is useful if you want to adjust parameters in your model and immediately see how they affect a plotted result. Once the Automatic update option is selected, the Data button becomes live allowing you to adjust the range of data that is plotted. At the bottom of the dialog it is possible to place data series on primary or secondary axes (top and bottom for *x* and left and right for *y*), as desired.

Error Bars

Select Y-axis in the Modify Plot dialog to bring up the Specify Error Bar Information dialog shown in Figure 1-39(a). The error bars can be set according to either an absolute or relative uncertainty. Select an Absolute uncertainty of 50 kPa for the *y* error bar. Repeat the process for the *x* error bar using an uncertainty of 50 liter/min in order to generate the plot shown in Figure 1-37(b). Alternatively, the uncertainty values used to generate the error bars can be obtained from a table. It is also possible to generate asymmetric error bars by de-selecting the Identical upper and lower values option in the Specify Error Information dialog, which brings up a separate window for selecting the upper and lower uncertainty values, as shown in Figure 1-39(b).



(a) **(b)**

Figure 1-39: Specify Error Bar Information dialog (a) with identical upper and lower values, and (b) with different upper and lower values.

The Plot Tool Bar

The plot tool bar (Figure 1-40) should be visible in the plot window; if not, select Show Tool Bar from the Plots menu.

Figure 1-40: Plot tool bar.

Selecting the text tool opens the Format Text dialog, shown in Figure 1-41(a). You can type text into the window in order to annotate your figure, as was done in Figure 1-37(c).

In the Professional version, you can also link the position of the text to a certain data point using the Link Text to Data Point in Plot option at the bottom of Figure 1-41(a). This capability is particularly useful if you have a plot that is automatically updated as you change parameters and you would like the annotations to remain in their correct locations with respect to the data series.

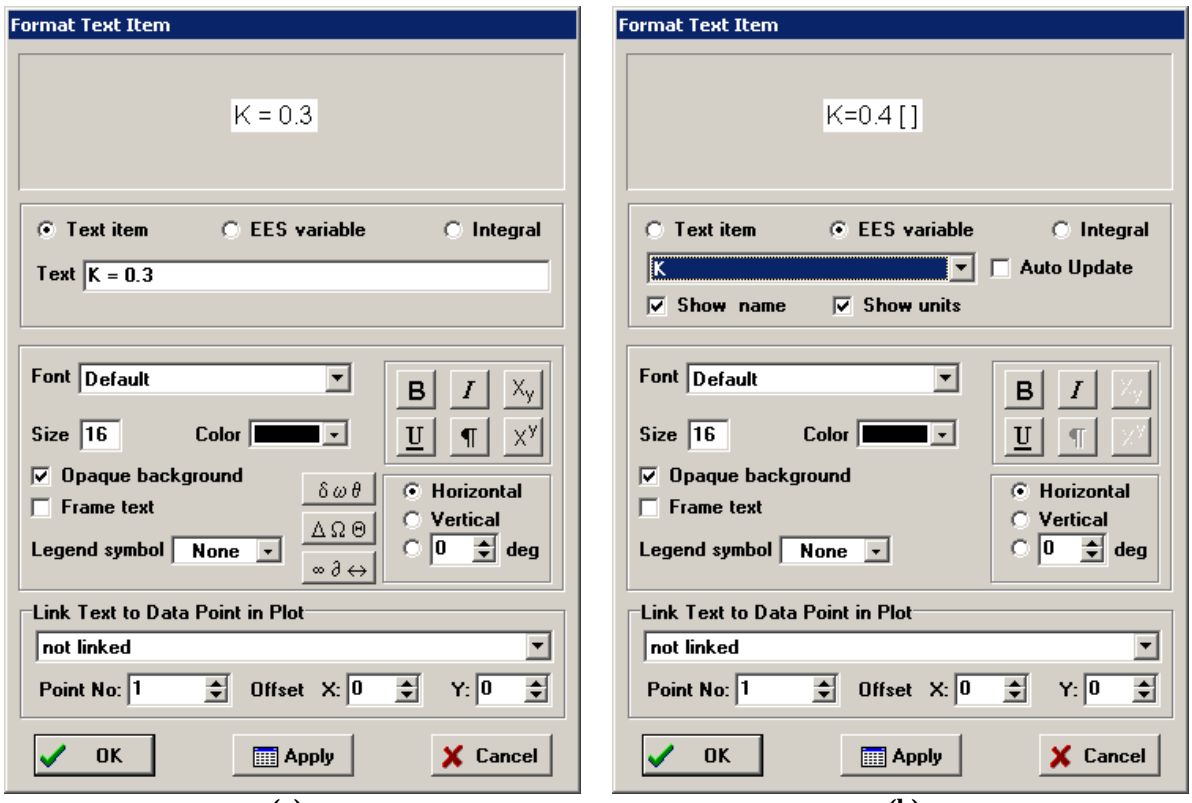


Figure 1-41: Format Text dialog with (a) Text item and (b) EES variable selected.

By default, the text tool is set to place a text item onto the plot; therefore the Text item radio button is filled in at the top of the dialog. However, you can also annotate the plot using the value of an EES variable by selecting the EES variable radio button. The resulting dialog is shown in Figure 1-41(b). Use the pulldown menu to select the appropriate variable (in our case K). If the radio button Show name is selected, then both the value and the name of the variable will be shown. If the Show units radio button is selected, then the units of the variable will also be shown provided that they are set, as discussed in Section 1.5. The Integral radio button automatically integrates one of the data series shown in the plots (selected by the user from a dropdown menu) and places the value on the plot.

There are several other useful tools in the Plot tool bar shown in Figure 1-40. The line tool allows you to draw lines on your plot. By holding the shift key down during the drawing process, the lines are constrained to horizontal, vertical, or 45°. The polyline tool allows you to draw a series of connected lines. The square , circle , and polygon tools allow

rectangular, circular, and polygon regions to be outlined on the plot (note that the polygon tool is only available in the Professional version). Holding the Shift key down while drawing these objects forces them to have a constant aspect ratio.

When the plot tool bar is visible you can also copy graphic objects from other applications (e.g., PowerPoint®) and paste them into the plot window. The elbow shown at the bottom right of Figure 1-42 was inserted in this manner. Right click on any of the objects on a plot in order to change its properties (e.g., to add arrows to lines or to shade regions in opaque or semi-transparent colors) or move objects forward (in front of) or backwards (behind each other) each other. Select multiple objects at once by clicking on them one by one while holding the Shift key down. Alternatively, you can select a rectangular region on the plot by pressing and holding the left mouse button down at the upper left corner of the region and moving the mouse to the lower right corner of the region. All objects within the region will be selected. The properties of all selected objects can be changed at once.

The alignment tool  will be activated when multiple objects are selected. The alignment tool can be used to align the positions of a group of selected objects vertically and/or horizontally. Figure 1-42 shows a carefully formatted plot of pressure loss as a function of flow rate through a 45º elbow calculated using various values of the resistance coefficient. The alignment tool is also activated when the X or Y-axis label is selected. In this case, the alignment tool can be used to move the label to the center the axis.

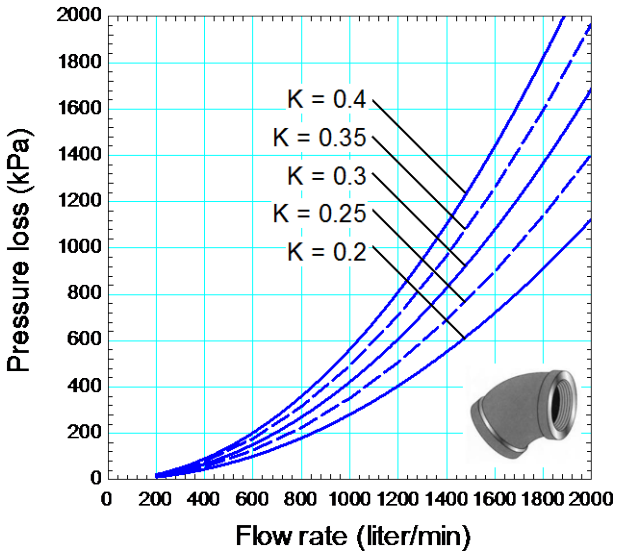


Figure 1-42: Formatted plot of pressure loss as a function of flow rate through a 45º elbow for various values of the resistance coefficient.

The Cross-Hairs, Move, and Zoom Tools

The cross-hairs tool brings up a status bar at the bottom of the plot showing the *x* and *y* coordinates of the cursor, as shown in Figure 1-43. This tool can also be made to appear by holding the Ctrl and Shift keys down, even if the plot tool bar is not visible. The plot is normally locked in position in the plot window. However, it can be moved using the Move plot tool (it can also be moved if the Ctrl key is held down).

The Zoom tool allows a rectangular region of the plot to be selected, as shown in Figure 1-44(a). A new plot is subsequently created, as shown in Figure 1-44(b), which includes the selected region and therefore provides a zoomed in view of a certain region of the original plot.

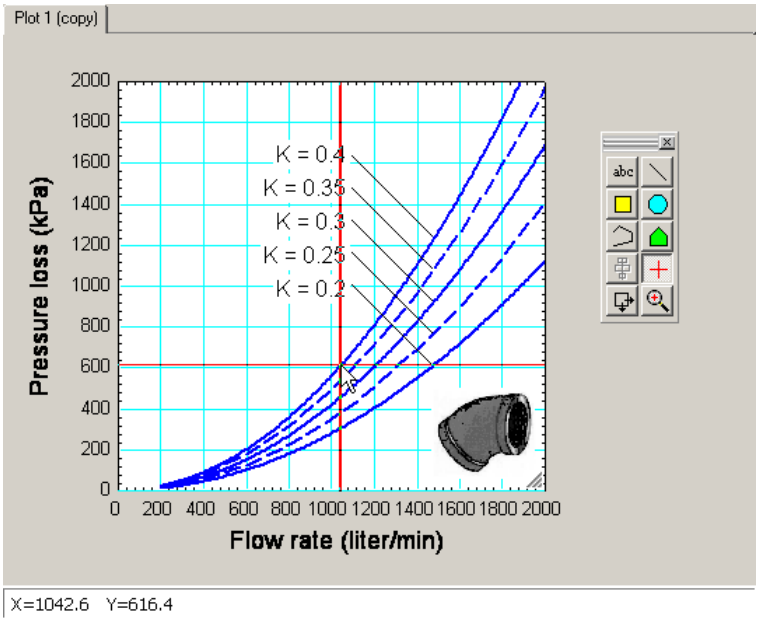
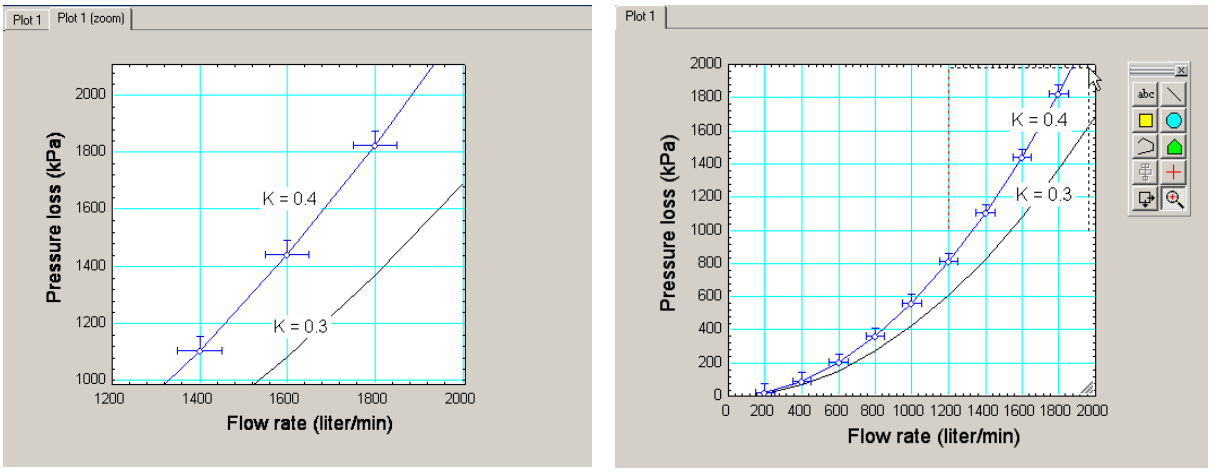


Figure 1-43: Plot window with cross hairs and status bar (at bottom).



(a) **(b)**

Figure 1-44: (a) Plot with the zoom feature activated and (b) new plot with zoomed in region.

1.5 Units

In Section 1.3 we calculated the pressure loss through a 45º elbow. During the process of solving the problem we found that it was tedious keeping track of the units of each variable and converting between different units. Variables in engineering problems typically represent physical quantities and therefore they will have units. EES allows you to assign not only a value to each variable, but also its units. This is an incredibly useful feature of EES because the unit consistency of the equations will be examined by EES and any unit conversion errors will be flagged. Further, EES facilitates the process of converting between units by including a large number of built-in unit conversion factors.

Unit System

Units are commonly categorized as belonging to the English or SI (Systems International) unit system. If you are accessing any of the built-in thermodynamic or transport property functions or any of the trigonometric functions in EES, then it is important that you specify the units system that you will be using. This is done by selecting Unit System from the Options menu to access the Unit System tab of the Preferences dialog, shown in Figure 1-45. You can select which unit system (English or SI) you will be using. Most of the settings in this page become important only when accessing property databases, as discussed in Chapter 4. In the section labeled Trig functions you must specify whether you want to provide arguments to trigonometric functions in radians or degrees.

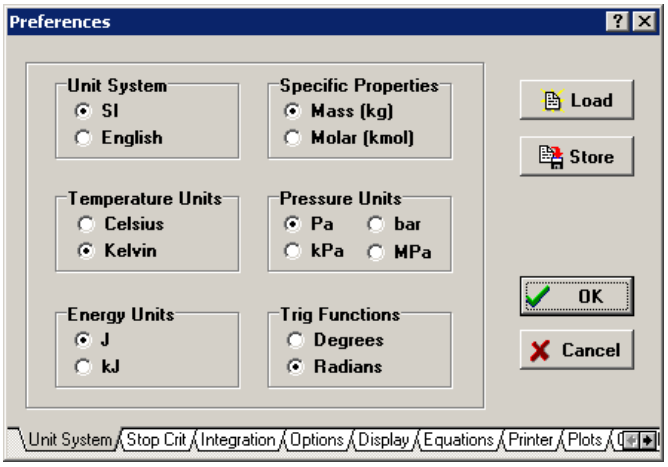


Figure 1-45: Unit System dialog.

Entering Units

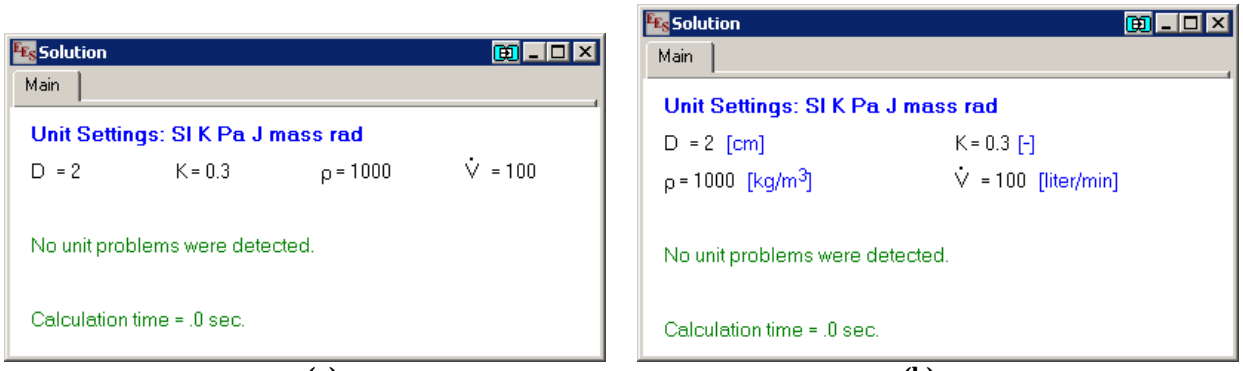
Let's return to the example from Section 1.3. The inner diameter of the elbow is D = 2 cm and the density of water is ρ = 1000 kg/m^3. The dimensionless resistance coefficient is estimated to be K = 0.3, allowing the pressure loss can be estimated according to:

$$\Delta P = K \frac{\rho u^2}{2} \tag{1-16}$$

where u is the velocity of the water. We will compute the pressure loss when the volumetric flow rate is $\dot{V}$ = 100 liter/min. The inputs are entered in EES as before:

```
$TabStops 0.2 2.5 in

"Inputs"
K=0.3                    "K factor for a 45 degree elbow, dimensionless"
D=2                      "diameter"
rho=1000                 "density"
V_dot=100                "flow rate"
```

Solve the equations and examine the Solution Window, shown in Figure 1-46(a). The value of each variable is indicated, but no units have been set.



(a) **(b)**

Figure 1-46: Solutions Window showing input parameters (a) with no units set and (b) with units set.

There are several ways to set the units of each variable. The units of a numerical constant (e.g., 2 or 1000) can be set directly in the Equations Window by placing the unit within square brackets directly after the value. The units of each input variable can therefore be set as they are entered.

```
$TabStops 0.2 2.5 in

"Inputs"
K=0.3 [-]                "K factor for a 45 degree elbow, dimensionless"
D=2 [cm]                 "diameter"
rho=1000 [kg/m^3]        "density"
V_dot=100 [liter/min]    "flow rate"
```

The resulting Solutions Window is shown in Figure 1-46(b). Both the value and units for each variable are indicated. Notice that using the hyphen alone as a unit specification indicates that the variable K is dimensionless (i.e., it has no units). A space could be used in place of the hyphen. The units for each variable can also be set by right-clicking on the variable in the Solution Window in order to access the Format Selected Variables dialog, shown in Figure 1-47 for the variable D.

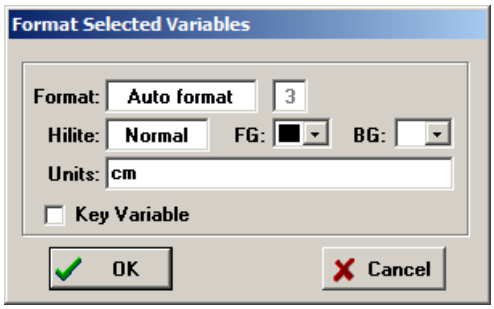


Figure 1-47: Format Selected Variables dialog.

Finally, the units for each variable can be set in the Variables Information Window which is accessed by selecting Variable Info from the Options menu.

Units Recognized by EES

The units that EES recognizes can be examined by selecting Unit Conversion Info from the Options menu in order to bring up the Unit Conversion Information dialog, shown in Figure 1-48. The left window lists each dimension (e.g., length) while the right window lists the built in units that are recognized for the selected dimension (e.g., m and ft). If you select two units in the right window, then EES will display the unit conversion in red at the bottom of the dialog, as shown in Figure 1-48 for m and ft.

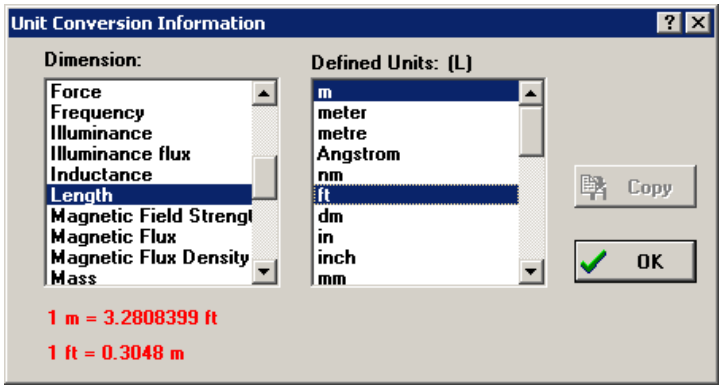


Figure 1-48: Unit Conversion Information dialog.

Checking Units

The cross sectional area of the elbow is given by:

$$A_c = \pi \frac{D^2}{4} \tag{1-17}$$

```
A_c=pi*D^2/4                    "cross-sectional area for flow, in cm^2"
```

By default, EES will not automatically set the units of A_c. Solve the problem and access the Solution Window, shown in Figure 1-49(a). Notice that there is no unit listed after the variable

A_c and also that a red warning message has appeared at the bottom of the Solution Window indicating that EES has detected a potential problem with the unit consistency of the equations.

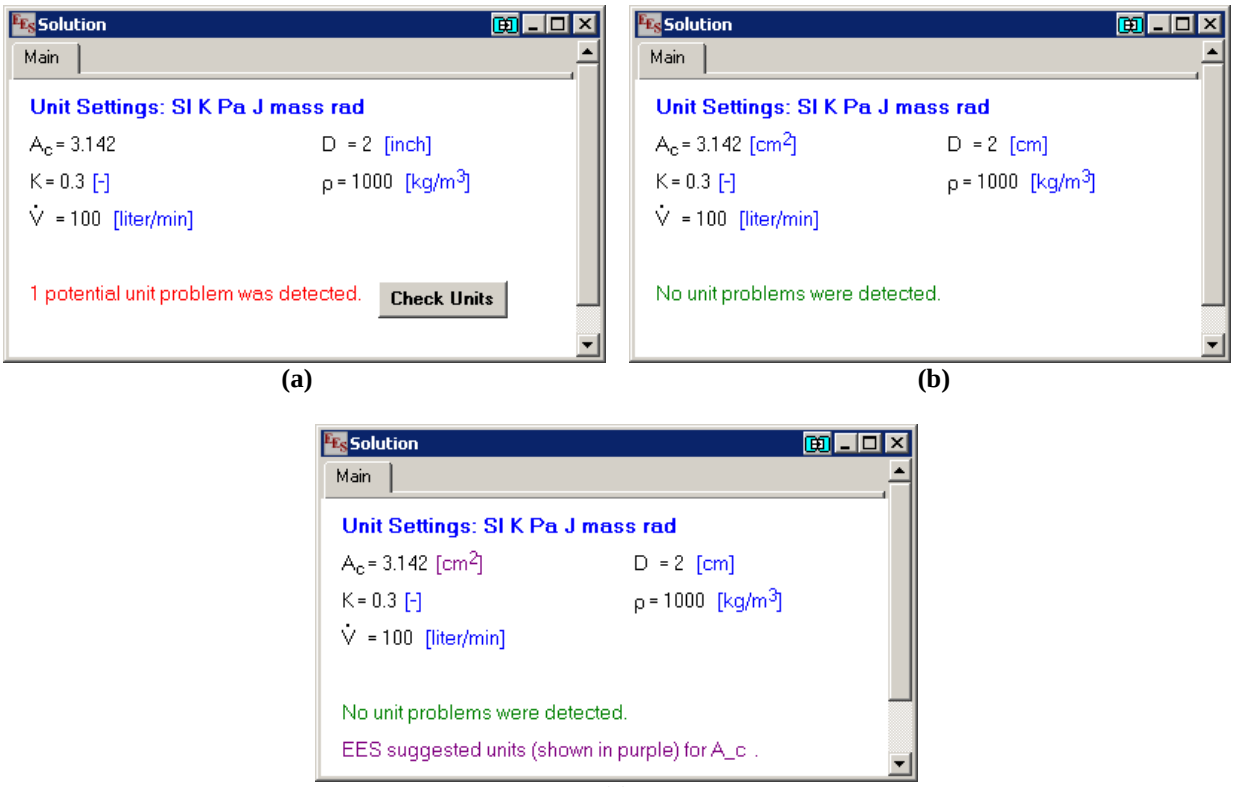


Figure 1-49: Solution Window with (a) the units of A_c not set, (b) the units of A_c set by the user, and (c) the units of A_c automatically set by EES.

Select Check Units in order to bring up the Check Units window shown in Figure 1-50. EES has identified that Eq. (1-17) is not dimensionally consistent if A_c is dimensionless and D has units cm. It is evident that the units of the variable A_c must be cm^2.

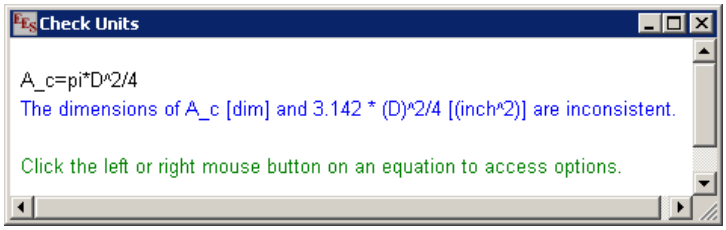


Figure 1-50: Check Units Window.

Clicking the left or right mouse button on an equation that has unit issues will bring up the pop-up menu in Figure 1-51.

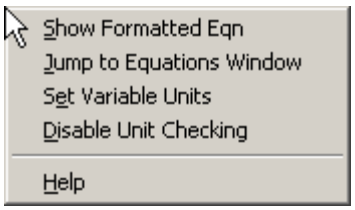


Figure 1-51: Options in the Unit Checking Pop-up Menu.

The first menu option will display the formatted equation showing the units of all variables and constants. This display makes it easy to identify unit problems. The second option will move the focus to the equation in the Equations window. The third option opens the Variable Information dialog where the units of one or more variables in the equation can be specified. The last option disables unit checking for the selected equation.

By default, EES automatically performs this type of unit check each time the system of equations is solved. Although not recommended, this option can be deactivated by selecting Preferences from the Options menu and selecting the Options tab to access the dialog shown in Figure 1-52. Automatic unit checking can also be controlled with the $CheckUnits directive. Placing $CheckUnits AutoOn at the top of the Equations Window activates automatic unit checking and $CheckUnits AutoOff deactivates it. The $CheckUnits On and $CheckUnits Off directives can be placed around a set of equations in order to locally deactivate the unit checking for these equations, as discussed in Section 14.2. The unit consistency of the equation set can also be checked at any time by selecting Check Units from the Calculate menu (or by selecting F8).

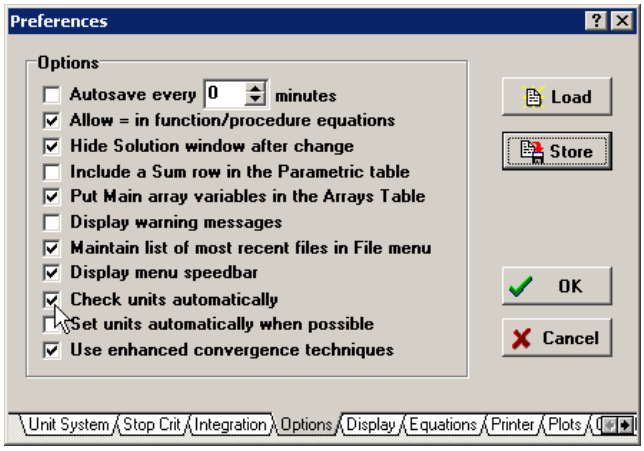


Figure 1-52: Options dialog.

It is not possible to set the units for the variable A_c in the Equations Window using square brackets because the right side of Eq. (1-17) is not a numerical constant. For example, typing:

```
A_c=pi*D^2/4 [cm^2]          "cross-sectional area for flow"
```

will result in EES interpreting the constant 4 in the denominator as having units cm^2. Alternatively, typing:

```
A_c [cm^2] =pi*D^2/4         "cross-sectional area for flow"
```

will result in EES interpreting the square bracket as the subscript in an array (discussed in Section 1.7), leading to an error message. Rather, the units of the variable A_c must be set either by right-clicking on the variable in the Solutions Window and entering the units or by using the Variable Information window. The result is shown in Figure 1-49(b). Note that the red unit warning is now gone.

Automatically Setting Units

It is also possible to have EES attempt to discern the appropriate units for each variable based on the equation in which it is used and then set these units automatically. This is not recommended as EES is no longer really checking the unit consistency of your equations and variables but rather forcing unit consistency, if possible, by adjusting the units of the variables. To activate this option, remove the unit specification for the variable A_c and select Set units automatically when possible from the Options tab of the Preferences dialog, shown in Figure 1-52. Solve the equation set to obtain the Solution Window shown in Figure 1-49(c). The units for the variable A_c have been automatically set to cm^2. The units suggested by EES are automatically indicated in purple to highlight the fact that these are not user-specified units. All units shown in purple should be confirmed by the user by right-clicking on one (or more) variables in the Solutions window and then making any necessary changes in the dialog window that appears.

Automatic unit setting can also be activated by placing the $AutoSetUnits On directive at the top of the EES Equations Window:

```
$AutoSetUnits On
```

The Convert Function

Let's continue to solve the example problem. The velocity of the water is calculated according to:

$$u = \frac{\dot{V}}{A_c} \tag{1-18}$$

```
u=V_dot/A_c                    "velocity"
```

As written, the units for the variable u must be liter/min (the units for the variable V_dot) divided by cm^2 (the units for the variable A_c). Therefore, the units for u must liter/min-cm^2. (Again, note that in this book and also in EES, the hyphen sign will be used to indicate multiplication on one side of the divisor in the context of units). If we set the units for u to liter/min-cm^2 in the Solutions Window, then EES will find no unit inconsistencies. Of course, the units liter/min-cm^2 don't make a lot of sense for a velocity and it would be better to convert the value to a more reasonable set of units, like cm/min. We could do this manually, looking up each of the various unit conversion factors that are required to arrive at the conclusion that we need to multiply by 1000:

$$\frac{\text{liter}}{\text{min-cm}^2}\left\|\frac{1000\ \text{cm}^3}{1\ \text{liter}}=\frac{\text{cm}}{\text{min}}\right. \tag{1-19}$$

Fortunately, EES has built-in unit conversion information that is easily accessible using the convert function. The convert function simply retrieves the conversion factor between two units and therefore the function requires two arguments; the first argument is the unit to convert *from* while the second is the unit to convert *to*. By adding the convert function to the equation calculating the velocity:

```
u=V_dot/A_c*convert(liter/min-cm^2,cm/min)      "velocity"
```

the unit conversion is easy and transparent to the user. The convert function returned the correct unit conversion multiplier (1000) for the unit conversion that is specified by the two arguments. This is particularly evident in the Formatted Equations Window, shown in Figure 1-53.

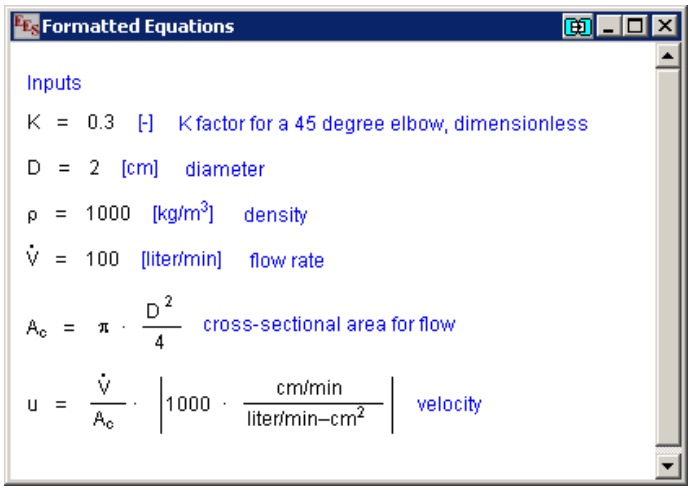


Figure 1-53: Formatted Equations Window.

The units of the variable u should now correctly be set as cm/min in the Solutions Window. Finally, Eq. (1-11) is used to compute the pressure loss and the result is converted to kPa:

```
DeltaP=K*rho*u^2/2*convert(kg-cm^2/m^3-min^2,kPa)    "pressure loss"
```

The units of the variable DeltaP are set to kPa in the Solutions Window. The unit assignments of each variable will automatically be stored and the units will be indicated whenever you use the variables, such in a Parametric Table or plot.

The Units List

The Units List provides a convenient shortcut for entering units that are used often. The units list is activated by right-clicking in any of the areas where you could otherwise manually enter units; this includes the Equations Window, the Units field in the Format Selected Variables dialog in the Solution Window, or the Units column of the Variable Information Window. Select Units List to access a list of commonly used units, as shown in Figure 1-54. The basic set of units in the list will depend on whether your unit system is set to English or SI (see Figure 1-45). You can simply click on the appropriate unit and select Paste or double click on the unit in order to enter it.

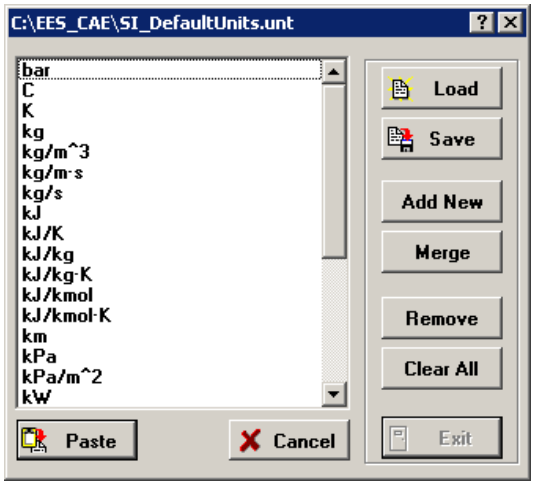
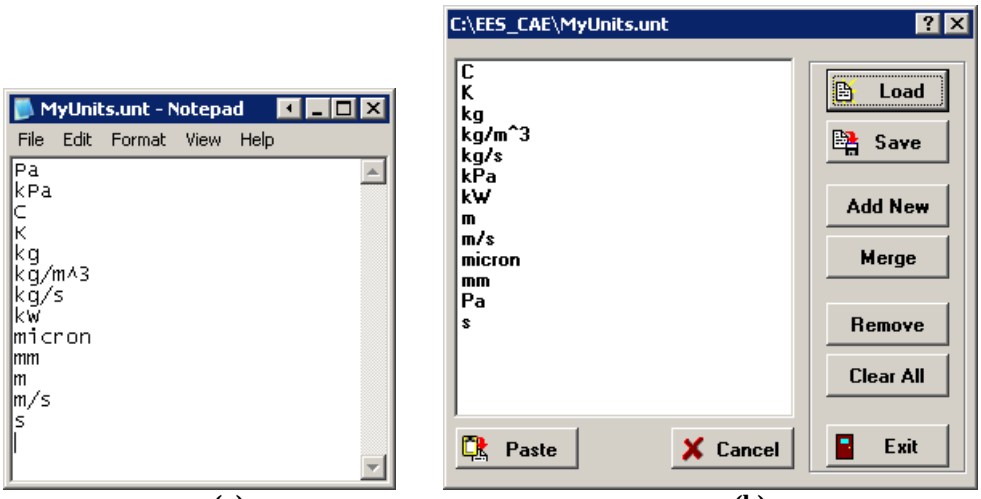


Figure 1-54: The default SI Units List.

The unit list can be modified so that it is most useful to you. You really want to include only a small number of units that you use a lot in order for it to save you time. The buttons to the right facilitate this process. Units can be added or removed with the Add New and Remove buttons, respectively. The entire list can be cleared with the Clear button. If you change the units list and want that change to be recorded beyond the current EES session then you should save the new file using the Save button.

EES will automatically load the default SI and English units lists (SI_DefaultUnits.unt and NG_DefaultUnits.unt, respectively) upon opening. You may have your own, separate units list that you have developed and would like to use. This is easy to do because the units list is simply a text file that can be opened and edited with any text editor (e.g., Notepad®). For example, Figure 1-55(a) illustrates a text file that is saved as MyUnits.unt. This units list can be loaded using the Load button in Figure 1-54 and then used for the duration of the EES session, as shown in Figure 1-55(b). Alternatively, a custom units list can be merged with an existing list using the Merge button.



(a) **(b)**

Figure 1-55: The unit list saved as (a) MyUnits.unt file and (b) loaded into EES.

If you don't want to load your custom units list each time you start EES then you can use the $Include directive. The $Include directive provides an automatic method of loading a variety of files when you run your EES code. Place the statement:

```
$Include MyUnits.unt
```

at the top of the EES file and you will find that the units list is automatically set as shown in Figure 1-55(b). Note that regardless of how the units in the units list are organized, they are simply a subset of the large set of units that are recognized by EES.

Suggested Method for Working with Units

Every engineer should have his or her own, carefully thought out method for dealing with units and unit conversions. Unit conversion errors are one of the most common mistakes that engineers make and it is clear that the unit checking and unit conversion features of EES will help to avoid these problems. The authors have developed a specific procedure for solving engineering problems that makes unit consistency easy to maintain. Inputs to the problem reported in arbitrary units will be converted to the base SI system (i.e., kg, m, s, K, N, etc.). The calculations required to solve the problem will be carried out using the base SI system and unit checking will be rigorously applied in order to establish the dimensional consistency of each equation. The results will be converted from the SI system into whatever units are requested or are logical. This procedure is convenient because the units of each variable are self-consistent in the SI unit system. It is not necessary to constantly worry about applying the correct unit conversion to each equation during the development of a model. As a result, if you are working in the SI unit system and you check the units of your equations then you are actually carrying out a more powerful and complete check on your equations; you are establishing their dimensional (as well as their unit) consistency.

Applying this strategy to our pressure loss problem leads to the following equation set:

```
$TabStops 0.2 2.5 in
```

```
"Inputs - converted to base SI units"
K=0.3 [-]                                   "K factor for a 45 degree elbow, dimensionless"
D=2 [cm]*convert(cm,m)                      "diameter"
rho=1000 [kg/m^3]                           "density"
V_dot=100 [liter/min]*convert(liter/min,m^3/s)  "flow rate"

"Calculations - done in the base SI unit system"
A_c =pi*D^2/4                               "cross-sectional area for flow"
u=V_dot/A_c                                 "velocity"
DeltaP=K*rho*u^2/2                          "pressure loss"

"Results - converted to convenient units"
V_dot_lpm=V_dot*convert(m^3/s,liter/min)    "flow rate, in liter/min"
u_cmpm=u*convert(m/s,cm/min)                "velocity, in cm/min"
DeltaP_kPa=DeltaP*convert(Pa,kPa)           "pressure loss, in kPa"
```

and the Solution Window shown in Figure 1-56. While it is not necessary that you take this approach, it certainly makes solving systems of engineering equations much easier. The settings for the base SI unit system are as shown in Figure 1-45.

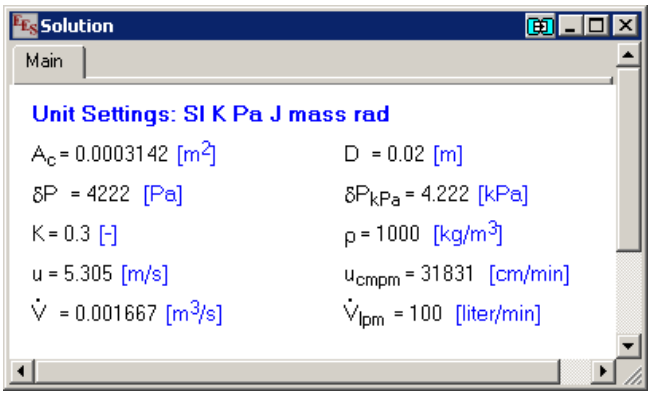


Figure 1-56: Solution Window.

The ConvertTemp Function

Most unit conversions require only a multiplicative correction that can be provided by the convert function in EES. However, both multiplication and addition operations are required to convert between temperature scales. Therefore, the convert function in EES cannot be used to convert a value from one temperature scale to another temperature scale. Instead, the ConvertTemp function in EES should be used for this purpose. The ConvertTemp function accepts three arguments. The first argument indicates the temperature scale to convert *from*, the second argument indicates the temperature scale to convert *to*, and the third argument is the numerical value, variable, or expression to be converted. The temperature scale indicator can be F, C, K or R, representing the Fahrenheit, Celsius, Kelvin, and Rankine scales, respectively. For example, the following equation converts a temperature of 70°F to the Celsius temperature scale.

```
T_F=70 [F]                          "temperature in F"
T_C=converttemp(F,C,T_F)            "temperature in C"
```

Adding Units

EES includes several hundred units and is therefore likely to include any unit you might run across. However, it is possible to add to the list of units recognized by EES. For example, braking distances are often expressed in car lengths where 1 car = 4 m. The unit 'car' is not recognized by EES. The equation:

```
L=4 [car]*convert(car,m)                    "length, in units of car"
```

will result in the error message shown in

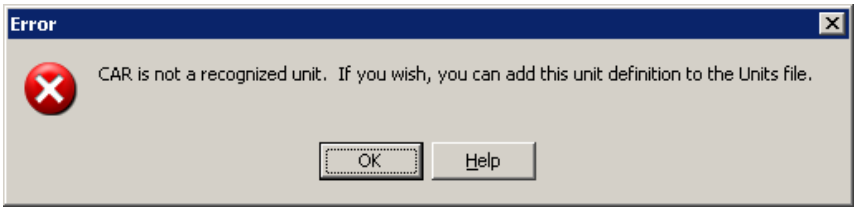


Figure 1-57: Error message.

We can add the unit car to the list of those that are recognized by EES by opening the file Units.txt found in the root EES installation directory using any convenient editor (e.g., Notepad®), as shown in Figure 1-58.

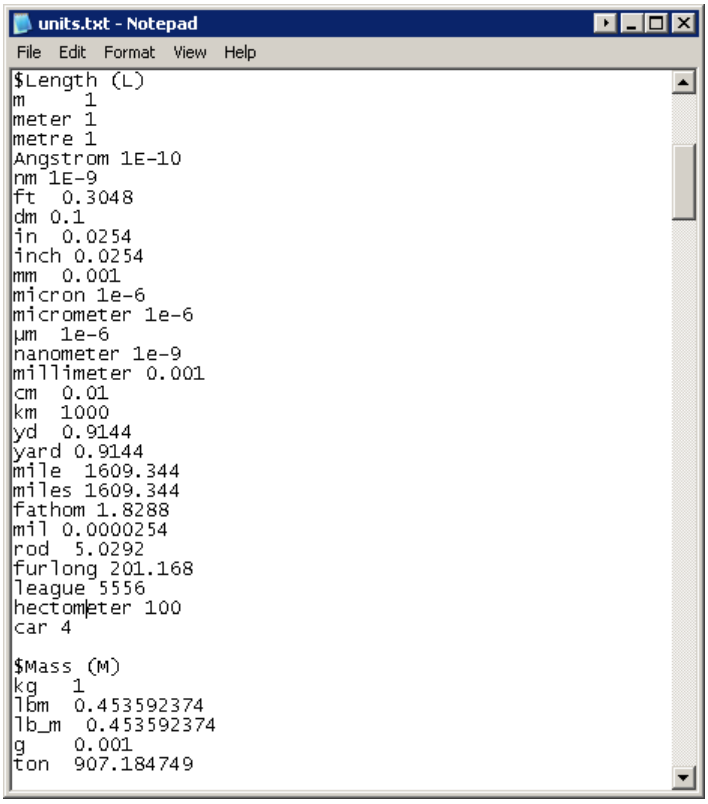


Figure 1-58: Units.txt file opened using Notepad® with the unit car added.

Instructions for adding units are found at the top of the Units.txt file. Unit conversions are organized according to dimension. Locate the dimension of interest (in this case, Length). The first entry in the unit list (in this case, m) is the basis used for conversion of all other units in that dimension. Reading down the list, this should be clear (1 Angstrom = $1x10^{-10}$ m, 1 nm = $1x10^{-9}$ m, etc.). Add the appropriate entry at the bottom of the list, as shown in Figure 1-58, and save the units.txt file. When EES is re-started it will load the new set of unit conversions and the statement:

```
L=4 [car]*convert(car,m)                    "length, in units of car"
```

will not result in an error. The units of the variable L should be set to m in the Solution Window, as shown in Figure 1-59. Note that you must de-select the Overwrite Units files selection when updating EES (see Figure 1-2) or your modified units.txt file will be overwritten when a new version of EES is installed.

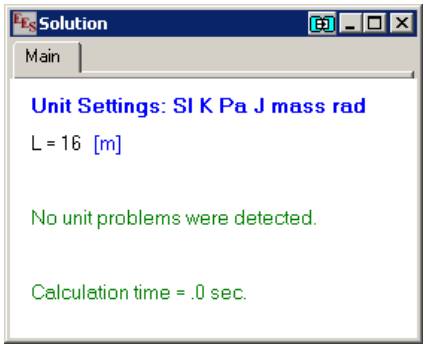


Figure 1-59: Solution Window.

Using String Variables for Units

The units of a variable can be assigned with a string variable. A string variable can be supplied in place of the unit designation within the square brackets that follow numerical constant or provided in the units column of the Variable Information dialog, as shown in Figure 1-60.

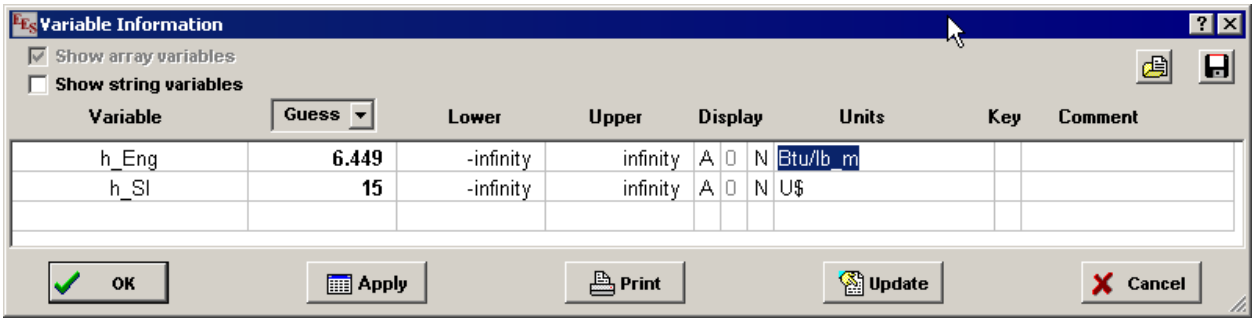


Figure 1-60: Variable Information Dialog shown a string variable used for the units of a variable.

A string variable can be also used in the Convert function.

```
U$='kJ/kg'
h_SI=15 [U$]
```

```
h_Eng=h_SI*convert(U$, Btu/lb_m)
```

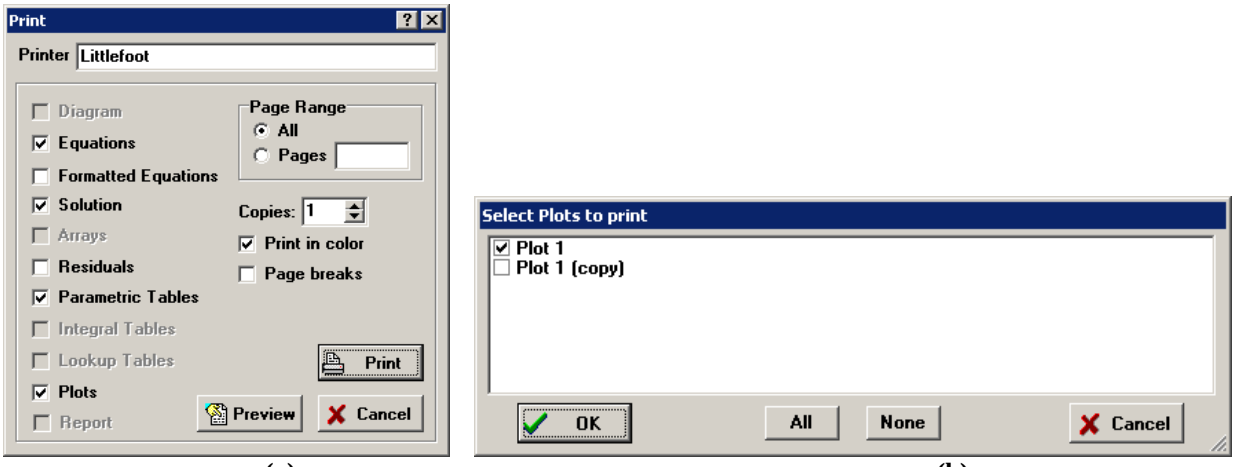
1.6 Printing

Once you have completed your EES solution, you will need to document the results in some way. This section discusses how you can output information from the EES file.

Printing a Hard Copy

The most direct method of output is simply to print a hard copy of your file to an installed printer. To do this, select Print from the File menu to bring up the Print dialog shown in Figure 1-61(a). The printer and number of copies can be selected. The Print in color option causes EES to attempt to print with the same color scheme that is used for the screen display. The Page breaks option causes EES to print each window on a different page.

Note that, by default, not all of the windows in your EES file will be printed. You need to select from among those shown along the left side of the dialog. While there is only one Equations Window, there may be more than one Parametric, Lookup, Array and Integral Tables and Plots. You will need to indicate both whether you which of these items are to be printed. A black check mark indicates that all of the items will be printed, an open box indicates that none will be printed, and a gray check mark indicates that a subset has been selected. For example, if more than one Plot window is present in the EES code, then clicking on the Plots box will bring up the Select Plots to print dialog shown in Figure 1-61(b). Select Preview in the Print dialog to bring up an on-screen display of the printed output.



(a) **(b)**

Figure 1-61: (a) Print and (b) Select Plots to print dialogs.

The Print Setup selection from the File menu allows you to set up the details of the printing process (e.g., orientation, paper size, etc.). By default, the font and style used for the printed output is the same as those used for the on-screen display. However, the Printer tab on the Preferences dialog allows you to adjust these values independently.

Copying Equations

It is also possible to copy equations from the Formatted Equations Window in EES so that they can be pasted into various applications. We will use Eq. (1-10), repeated below, as an example.

$$\overline{f} = \frac{4}{Re_D}\left[\frac{3.44}{\sqrt{L^+}} + \frac{\frac{1.25}{4L^+} + \frac{64}{4} - \frac{3.44}{\sqrt{L^+}}}{1 + \frac{0.00021}{\left(L^+\right)^2}}\right] \tag{1-20}$$

This equation is entered in the Equations Window as shown below.

```
f_bar=(4/Re_D)*(3.44/sqrt(L|plus)+(1.25/(4*L|plus)+64/4-3.44/sqrt(L|plus))/(1+0.00021/L|plus^2))
```

Open the Formatted Equations Window, shown in Figure 1-62. Highlight the equation and right click to access the pop-up menu shown in Figure 1-63.

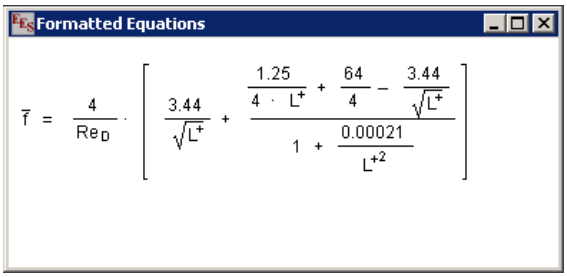


Figure 1-62: Formatted Equations Window.

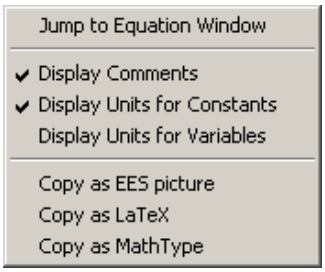


Figure 1-63: Pop-up menu for the formatted equation.

The three options at the bottom of the pop-up menu allow the equation to be copied and pasted in a variety of formats. If the Copy as EES picture option is selected then the equation will be placed on the clipboard as a picture (i.e., in windows metafile format), from which it can be pasted into another application.

The next two options are only available in the Professional version of EES. The Copy as LaTeX option translates the equation into LaTeX code that is placed on the clipboard. The Copy as MathType option translates the equation into MathType format that can subsequently be placed in another application such as Microsoft Word® provided that the MathType® software is installed.

LaTeX Report

EES can be used to create a high quality report that includes the Diagram, Equations, and Solution Windows as well as any tables and plots. The report is created in the form of a TeX document (.tex) that can be automatically processed by a LaTeX compiler to produce a device independent (.dvi) output file that can be printed by various utilities. The PDFLaTeX accessory available with the TeX compiler also produces a portable document interface (.pdf) file that is compatible with Adobe Reader. The entire process can be made transparent to the user so that the Latex Report feature effectively becomes a printer for .pdf files.

Both the LaTeX software and Adobe Acrobat Reader must be installed on your computer in order to use this feature. These programs are available at no cost. Download the distribution MiKTeX installation file (i.e., the latest version of the MiKTeX software) from the web page http://miktex.org using the instructions provided when you click the Help button in the Create LaTeX/PDF report dialog shown in Figure 1-64. Note that EES must eventually know the directory path in which the MiKTeX software is installed so you should make a note of it. In order to configure EES to work with the MiKTeX package, start EES and select Create LaTeX/PDF Report from the File menu in order to access the Create LaTeX/PDF Report dialog shown in Figure 1-64. Click the Setup button and navigate to the pdfLaTeX executable file (typically found in the miktex/bin subdirectory of the installation directory). If you install MiKTeX in its default directory, EES will likely find it without your intervention.

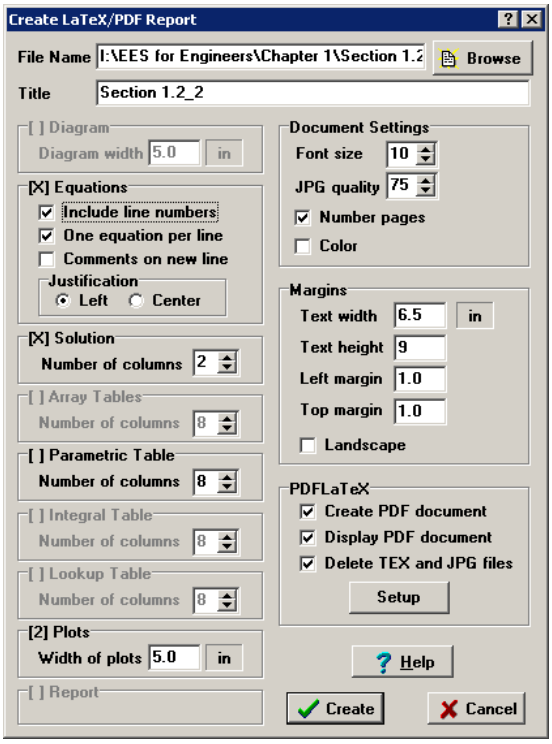


Figure 1-64: Create LaTeX/PDF Report dialog.

It is likely that Adobe Acrobat Reader is already installed on your computer. However, if necessary, you can download it from http://www.adobe.com/products/acrobat/readstep.html. Adobe Acrobat can also be installed directly from the EES CD when you install EES.

In order to generate a report, access the dialog shown in Figure 1-64 by selecting the Create LaTeX/PDF Report command in the file menu and enter the desired filename in the edit field at the top of the window. By default, the filename will be the name of the EES file with a .tex extension. The text entered in the Title field will be displayed at the top of the report. The controls at the left side of the dialog allow you to select the items that will be included in the report from among the various items in the EES file. If multiple items of the same type exist (e.g., there are two plots in the EES file used to create Figure 1-64) then clicking on the value within the square brackets will bring up a dialog that allows you to select a subset of the items. The Equations Window selection provides some additional controls such as line numbering and placing each equation and comments on separate lines. Very long equations may run off of the right side of the report. It may be necessary to use the & character to break long equations into segments in the Equations Window. The Solutions and Table group boxes each provide a control that allows you to specify the number of columns per page to ensure that the text does not run off the page of the report. The diagrams and plots that are to be included with the LaTeX document are stored in separate jpeg files. The resolution of these files can be controlled using the JPG quality control.

Towards the bottom right of the dialog there are three checkboxes. If the Create PDF document checkbox is selected then EES will attempt to start the PDFLaTeX application and compile the .tex file in order to create a .pdf file. If the Display PDF document checkbox is selected then EES will attempt to start Adobe Acrobat and display the .pdf file directly after it is created. Finally, if the Delete TEX document checkbox is selected then the .tex file and .jpg files will be deleted, leaving only the .pdf file. Select Create to start the process of generating and compiling the report.

1.7 Arrays

The variables that have been discussed in Sections 1.1 through 1.6 are all scalar variables; that is, they can be assigned only one value. It will often be useful to use array variables which can be assigned multiple values. Arrays are EES variables that have an array index enclosed in square brackets at the end of the variable name. Data stored in arrays can be used to generate plots in the same way that data stored in Parametric Tables can be used for this purpose, as discussed in Section 1.4.

Assigning Array Variables

The variable T_scalar, assigned below, has one value.

```
T_scalar = 100 [K]                    "a scalar variable"
```

The value and units of the variable T_scalar can be examined in the Solution Window. The array T_array[], assigned below, consists of three separate variables: T_array[1], T_array[2], and

T_array[3]. The value in subscripts is the index of the array; therefore, T_array[1] is the first element of the array, and so on.

```
T_array[1]=100 [K]                    "the first element of the array T_array"
T_array[2]=200 [K]                    "the second element of the array T_array"
T_array[3]=300 [K]                    "the third element of the array T_array"
```

The values in the variable T_array[] will be stored in the Arrays Table Window, shown in Figure 1-65 and accessed by selecting Arrays from the Windows menu.

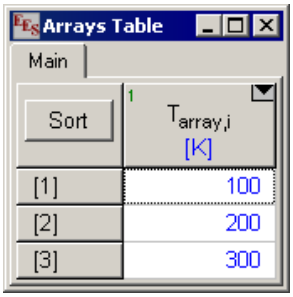


Figure 1-65: Arrays Table Window.

The array variables T_array[1], T_array[2], and T_array[3] behave just as ordinary EES variables. They each have units, limits, guess values, etc. One purpose of using arrays is to organize information; for example, each array variable may correspond to a thermodynamic state in a system. Also, mathematical operations that are common to each element of an array can be performed easily using the Duplicate command, discussed subsequently.

A string array variable must have the $ character just before the left bracket that holds the array index. For example:

```
A$[1]='Hello'
A$[2]='World'
```

Array Range Notation

A convenient shorthand method for referring to multiple array values uses the array range notation. A continuous range of indices in an array can be referred to by the first index and last index separated by two dots. For example, T_array[1..3] can be used in place of T_array[1], T_array[2], T_array[3]. Array range notation can be used in a variety of contexts including assignment statements. For example, the array T_array[] could be created using the statement:

```
T_array[1..3]=[100 [K], 200 [K], 300 [K]]    "assignment using array notation"
```

Note that EES expects the correct number of values (three in the example above) to exist within the square brackets on the right side of the equation above or it will post an error. If more than three values were provided in the square brackets on the right side of the equation then the first three would be used and EES would post a warning.

Two-Dimensional Arrays

If two indices are used for array assignments then the array will be two-dimensional (i.e., a matrix rather than a vector). For example, the statement below:

```
T_array[1,1..3]=[100, 200, 300]          "first row of array"
T_array[2,1..3]=[400, 500, 600]          "second row of array"
```

results in the two-dimensional array shown in Figure 1-66.

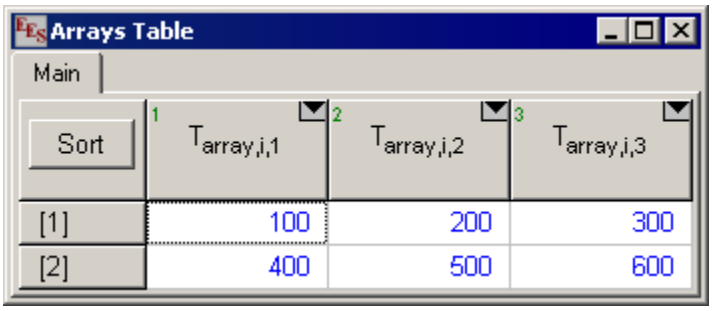


Figure 1-66: Two-dimensional array shown in the Arrays Table Window.

The statement below assigns the value in the 2nd row (the first subscript) and 3rd column (the second subscript) to the variable X. Examining the Solution Window shows that the scalar variable X is equal to 600.

```
X=T_array[2,3]                           "value in 2nd row and 3rd column"
```

The Array Editor

The Array Editor can be selected from the Edit menu in order to provide an easy way to enter or modify EES arrays. Select the Array Editor command while you are in the Equations Window. The dropdown menu at the top of the editor can be used to select from among any of the arrays that have currently been defined in the Equations Window using the Array editor. Alternatively, a new array variable can be created by selecting Enter array name from the Array drop down menu.

Modifications to existing arrays made in the Array Editor will show up in the Equations Window. (Note that the Array Editor will not detect or be able to modify arrays that were not created in the Array Editor.) This feature makes it easy to import data from other applications such as Excel or MATLAB. For example, suppose you want to access a 5x5 array of data generated in Excel, as shown in Figure 1-67.

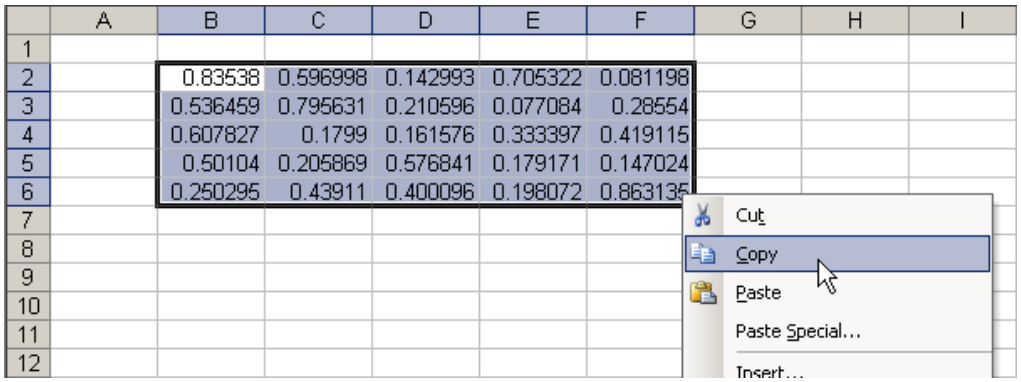


Figure 1-67: An array of data in Excel.

Select Array Editor from the Edit menu. Name the array (D) and choose the right number of rows and columns (5 and 5). Copy the data from Excel and select the Paste button ( ) to place the data into the Arrays Editor, as shown in Figure 1-68.

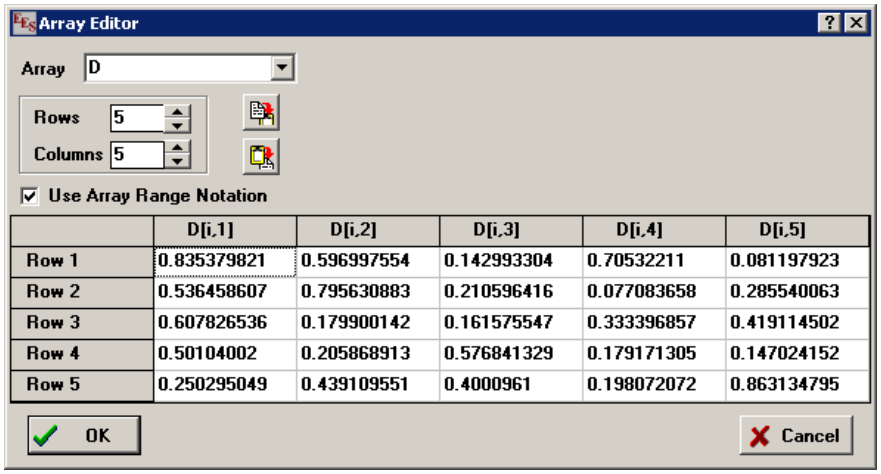


Figure 1-68: Arrays Editor.

Select OK and the array D[] will be created in the Equations Window.

```
{Array D}
D[1..5,1]=[0.835379821,0.536458607,0.607826536,0.50104002,0.250295049]
D[1..5,2]=[0.596997554,0.795630883,0.179900142,0.205868913,0.439109551]
D[1..5,3]=[0.142993304,0.210596416,0.161575547,0.576841329,0.4000961]
D[1..5,4]=[0.70532211,0.077083658,0.333396857,0.179171305,0.198072072]
D[1..5,5]=[0.081197923,0.285540063,0.419114502,0.147024152,0.863134795]
{Array D end}
```

Run the EES program and you will see the array D[] created in the Arrays Table, as shown in Figure 1-69. Do not modify or delete the comments that EES places around the array values, as these are used to identify the array to the Array editor.

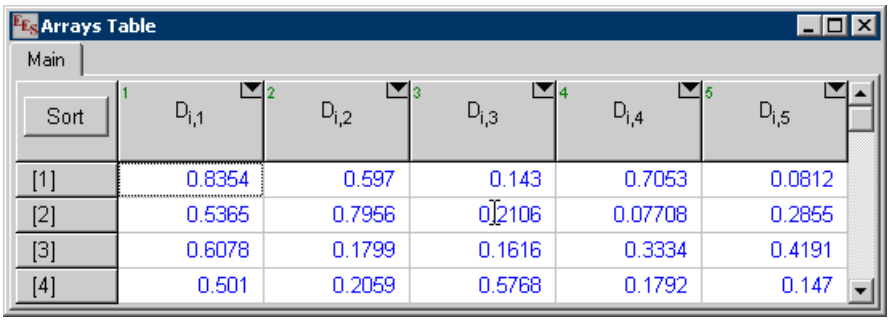


Figure 1-69: Arrays Table.

The Duplicate Statement

The Duplicate statement provides a convenient method for carrying out similar operations on multiple elements of an array. The protocol for a Duplicate statement is:

```
Duplicate index = start,stop
    equation(s) to be duplicated
End
```

The variable index is iterated from the value start to the value stop. The equation(s) contained between the Duplicate and End statements are written for each iteration. For example, the EES code:

```
duplicate i=1,4
    T[i]=i
end
```

is equivalent to writing:

```
T[1]=1
T[2]=2
T[3]=3
T[4]=4
```

When using Duplicate statements it is particularly important to remember that EES uses equations rather than assignments. That is, do not accidentally place equations within a Duplicate statement unless you actually want these equations to be duplicated in the Equations Window. A common mistake that new users of EES make is shown below:

```
duplicate i=1,4
    T[i]=i
    x=4
end
```

The above code is equivalent to writing:

```
T[1]=1
x=4
T[2]=2
```

```
x=4
T[3]=3
x=4
T[4]=4
x=4
```

which leads to 8 equations in 5 unknowns. Solving the EES code will lead to an error message.

Nested Duplicate statements can be used to generate two-dimensional arrays. For example, the code below:

```
duplicate i=1,3
    duplicate j=1,4
        D[i,j]=i+j
    end
end
```

leads to the two-dimensional array shown in Figure 1-70.

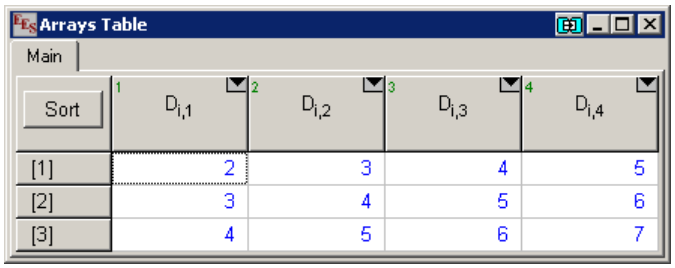


Figure 1-70: Two dimensional array generated using a nested Duplicate statement.

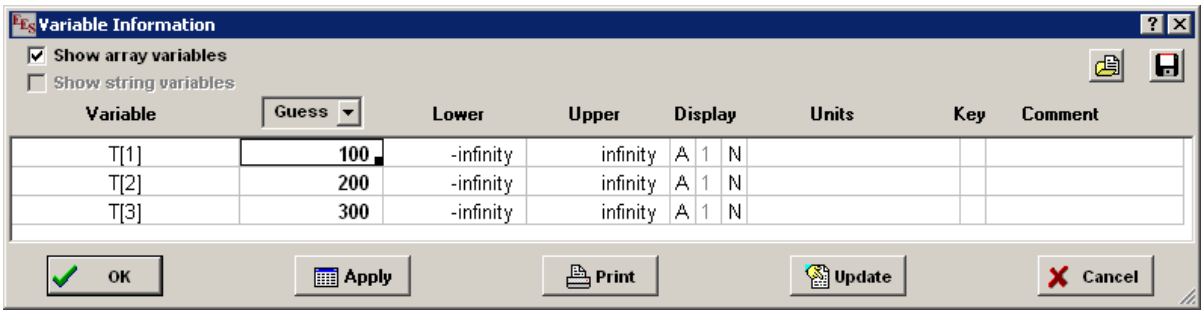
It is possible to define array variables with more than two indices but this practice is not recommended.

Arrays in the Variable Information Window

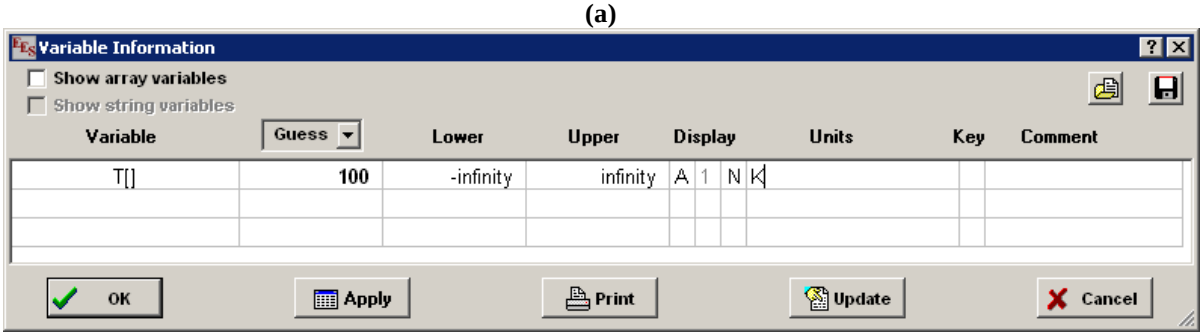
The units of all elements in an array are often the same because a typical array represents different values of the same physical quantity. For example, the array T[]:

```
T[1..3]=[100,200,300]
```

might represent a set of temperatures. Select Variable Info from the Options menu to access the Variable Information Window, shown in Figure 1-71(a). By default, each element of the array is shown and the guess values, limits, and units can be set differently for each element. This arrangement is inconvenient if all elements share a common set of characteristics. De-select the show array variables button in the upper left corner in order to collapse the array, as shown in Figure 1-71(b). Any characteristic set for the row T[] will be applied to each element of the array.



(a)



(b)

Figure 1-71: Variable Information Window (a) with each element of T shown and (b) with the array T collapsed.

It is possible to assign different values of the guess values or limits to each element in an array using a different array that is setup just for this purpose. This capability is discussed in Section 5.3.

Purge Unused Variables

EES saves every variable that you introduce as you develop your program. For example, if you start your problem with an array X[] that contains 500 entries and then later change its name to Y[], EES will retain in memory both X[] and Y[] for a total of 1000 variables. This is done so that you don't have to re-enter the characteristics (e.g., units and limits) of the variable X[] should you want to temporarily remove it. Similarly, if at some point your array X[] is defined with 2000 entries and later you reduce its size to 500 entries then EES will retain in memory 2000 variables.

It is often the case that you will want to redefine the size of the arrays that you use in an EES program. For example, arrays are particularly useful in order to solve the system of algebraic equations that result from numerical models. These equations correspond to a set of equations written for each node or control volume defined within a computational domain. One of the most important things that you should do with a numerical model is to verify that the number of nodes is sufficiently large that the numerical model has converged. This check is typically done by increasing the number of nodes and observing how the solution changes. The final number of nodes that is used is always a compromise between computational time and accuracy.

Based on this discussion, it is clear that you can easily introduce many variables using arrays that you do not ultimately wish to use. In order to clear these variables from memory you must use

the Purge Unused Variable command from the Options menu, at which point EES will check if there are unused variables in memory and ask whether you want to purge or keep them. Solving the EES code below generates a 50x50 two-dimensional array D[]. The EES program includes 2502 variables, all of the entries in the array D[] as well as the variables Nx and Ny.

```
$TabStops 0.25 0.5 in
Nx=50
Ny=50
duplicate i=1,Nx
      duplicate j=1,Ny
            D[i,j]=i*j
      end
end
```

Now reduce the values of the variables Nx and Ny, as shown below, in order to generate a 10x10 array and reduce the number of active variables to 102.

```
$TabStops 0.25 0.5 in
Nx=10
Ny=10
duplicate i=1,Nx
      duplicate j=1,Ny
            D[i,j]=i*j
      end
end
```

Because EES retained the original 2502 variables in memory there are currently 2400 unused variables. In order to purge these variables and reduce the size of the EES file select Purge Unused Variables from the Options menu. You will be asked to confirm that you want to purge the unused variables with the dialog shown in Figure 1-72.

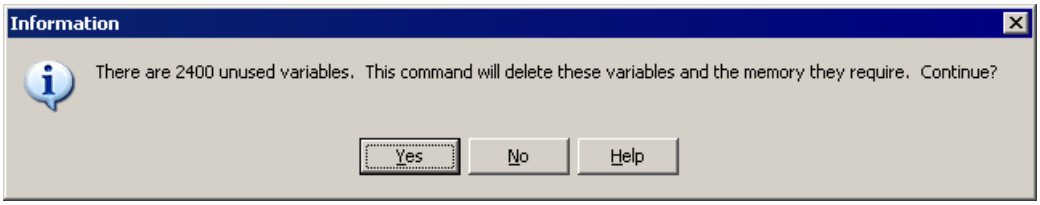


Figure 1-72: Purge unused variables.

It is wise to save the file with a different name before you purge variables.

1.8 Lookup Tables

Lookup Tables provide a convenient method of saving and accessing tabular data. These tables are useful for interpolation and curve fitting, as discussed in Chapter 2

Creating a Lookup Table

To create a Lookup Table, select New Lookup Table from the Tables menu. You can enter a name and the size of the table in the resulting dialog, as shown in Figure 1-73.

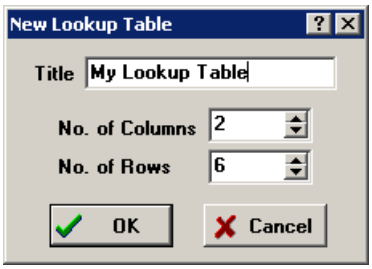


Figure 1-73: New Lookup Table dialog.

Select OK in order to generate the blank Lookup Table shown in Figure 1-74(a).

My Lookup Table

	1 Column1	2 Column2
Row 1		
Row 2		
Row 3		
Row 4		
Row 5		
Row 6		

(a)

My Lookup Table

Paste Special	1 Column1	2 Column2
Row 1	0	
Row 2	2	
Row 3	4	
Row 4	6	
Row 5	8	
Row 6	10	

(b)

My Lookup Table

	1 Column1	2 Column2
Row 1	0	0
Row 2	2	-0.3784
Row 3	4	0.4947
Row 4	6	-0.2683
Row 5	8	-0.144
Row 6	10	0.4565

(c)

Figure 1-74: Lookup Table (a) blank, (b) with 1st column filled in, and (c) with 2nd column filled in.

Entering Data

Data can be typed directly into the cells of the table or pasted from the clipboard. Alternatively, the values in a selected column can be entered, cleared, or changed by either selecting the triangle icon (▼) or right clicking on the column header and selecting Alter Values; either option brings up the Alter Values dialog. Select Enter Values and then indicate the pattern that should be used. For example, Figure 1-75(a) shows the Alter Values dialog set up in order to enter values ranging from 0 to 10 in equal increments, leading to Figure 1-74(b).

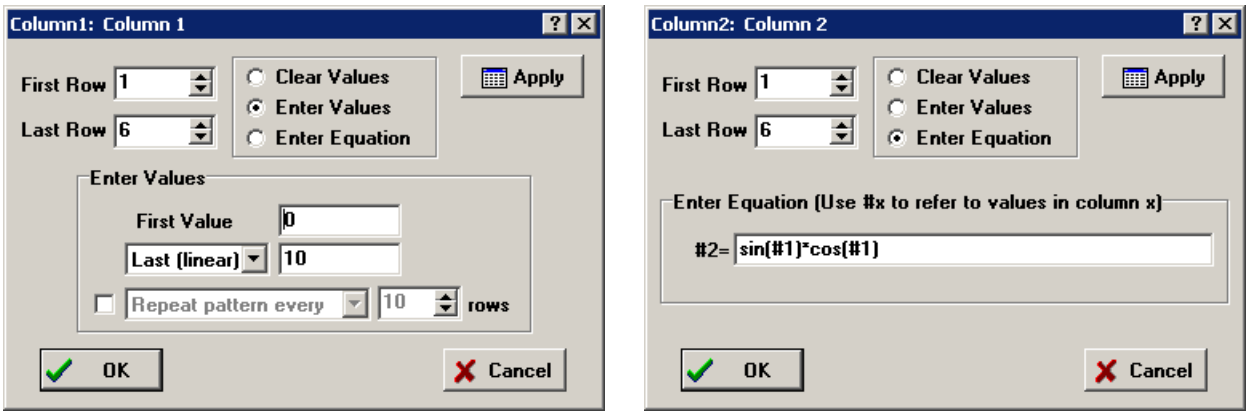


Figure 1-75: Alter Values dialog shown (a) with Enter Values selected and (b) with Enter Equation selected.

Using Equations

Values can also be entered using equations by selecting Enter Equations in the Alter Values dialog, as shown in Figure 1-75(b). The equation can refer to values in the EES workspace or values in other columns of the Lookup Table. For example, Figure 1-75(b) illustrates the Alter Values dialog setup so that the values in column 2 are calculated as the product of the sine and cosine of the values in column 1. The indicator #X refers to values in column X where X is an integer. Clicking OK results in the Lookup Table shown in Figure 1-74(c).

Saving and Loading Lookup Tables

Right click on the tab for a Lookup Table to bring up the Information Dialog for the Lookup Table. The dialog allows you to edit the name, add some descriptive text, duplicate the Lookup Table and alter its position with regard to other Lookup Tables in the EES file. The Save button allows you to save the Lookup Table in a variety of formats. The data alone can be saved in an ASCII text (.txt) or comma separated value (.csv) file that can be read by most other applications. The data with the headers can also be saved in a text file. The table can also be saved in the EES-specific Lookup File or Lookup Table formats. The Lookup Table (.lkt) format is a binary format that can subsequently be opened in other EES files. This format is typically easiest to use and corresponds to a Lookup Table that is saved by and can subsequently be opened by the EES software.

Lookup Commands

Data in a Lookup Table can be used as the basis of 1-D or 2-D interpolation, as discussed in Chapter 2. Data in a Lookup Table can be retrieved using the Lookup function (for numerical values) or the Lookup$ function (for strings). The calling protocol for the Lookup function is:

```
X = Lookup('TableName',Row,Column)
```

where 'TableName' is a string containing the name of the table of interest, Row is the number of the row in the table and Column is either the number or name of the column. For example, the command:

```
X=Lookup('My Lookup Table', 3, 2)
```

will result in the variable X being assigned 0.4947 if the Lookup Table shown in Figure 1-74(c) is available in the EES file. Non-integer values of the column or row will result in linear interpolation; however, the Interpolation functions discussed in Chapter 2 are more appropriate for this type of operation.

Data can also be written to a Lookup Table using the Lookup command in a Function or Procedure. Functions and Procedures are described in Chapter 3.

1.9 Other Features

There are a few additional basic features of EES that are worth discussing before closing out our introduction to the program.

Built-In Constants

Constants in EES are denoted by a # sign after the name; constants are like variables in EES except that their values cannot be changed in the Equations Window. A large number of common physical constants are pre-assigned in EES. These entries can be viewed by selecting Constants from the Options menu in order to access the Constants dialog, shown in Figure 1-76.

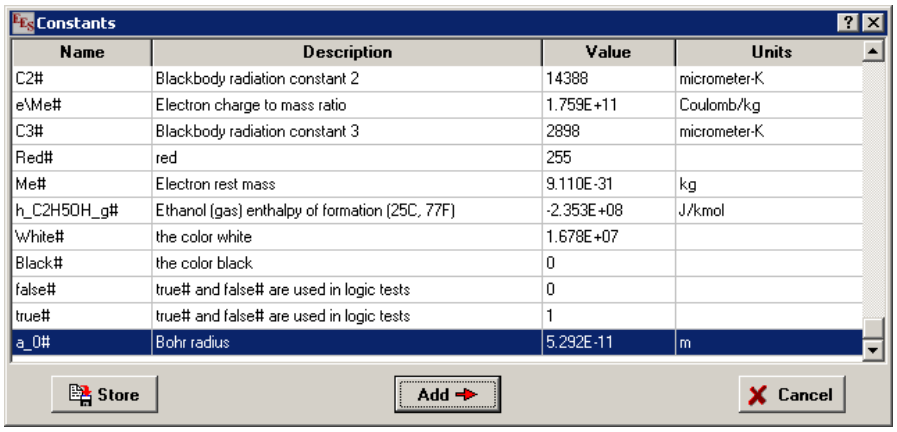


Figure 1-76: Constants dialog.

Many physical constants of interest are available; for example, pi# is the value of π (actually, pi also refers to π and is the only built-in constant in EES that does not require the use of #). R# is the universal gas constant, k# is Boltzmann's constant, etc. These constants are provided in the unit system that has been specified by the user, as discussed in Section 1.5. The order of the constants appearing in this dialog can be changed by clicking on the column header. For example, clicking on Name at the top of column 1 will rearrange the order of the table so that the entries are ordered alphabetically in terms of their names. Clicking a second time on Name will reverse the order.

Adding Constants

It is possible to add constants by selecting Add from the Constants dialog in Figure 1-77 in order to access the Add New Constant dialog shown in Figure 1-77.

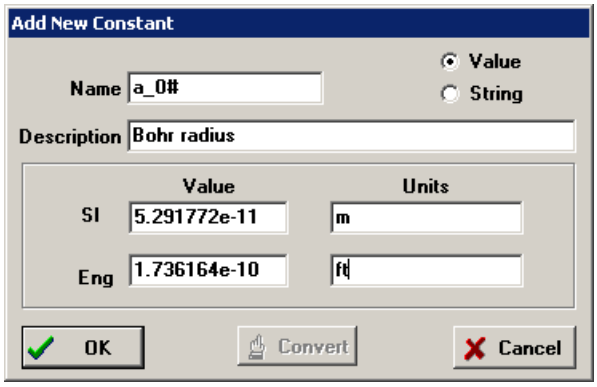


Figure 1-77: Add New Constant dialog.

The Add New Constant dialog prompts the user for the name, description, and value of the constant in both SI and English units. For example, suppose we want to enter the Bohr radius, a_0, to our EES program. The value of the Bohr radius is 5.291772×10^{-11} m (or 1.736146×10^{-10} ft) according to Mohr et al. (2011). The Add New Constant dialog should be filled out as shown in Figure 1-77. If you select OK and access the Constant dialog by selecting Constants from the Options menu then you will see that the constant a_0# has been added to the bottom of the dialog, as shown in Figure 1-76. The new constants list can be saved by selecting the Store button in the Constants dialog so that it is available each time EES is opened. Note that you will need to de-select the Overwrite Constants.txt file in the installation process, shown in Figure 1-2, in order to keep your added constants when you update or reinstall the EES program. Alternatively, make a copy of the Constants.txt file to ensure that it is not overwritten when you install a new version of EES.

The user can also specify the value of constants in an EES program using the $Constant directive. The calling protocol for the $Constant directive is:

```
$Constant Name# = ValueSI [UnitsSI] ValueEng [UnitsEng]
```

where Name# is the name of the constant, ValueSI and ValueEng are the values of the constant in the SI and English units specified by [UnitsSI] and [UnitsEng], respectively. For example, the Bohr radius can be entered in EES according to:

```
$Constant a_0#=5.291772e-11 [m] 1.736146 [ft]
```

Finally, it is possible to edit the Constants.txt file directly to add a constant that is loaded each time EES is opened. See Section 14.3 for more detail relating the $Constants directive.

The Calculator Window

The Calculator Window allows you to quickly carry out simple calculations without interfering with the equations that are entered in the Equations Window. This capability is often very useful when you are trying to debug an EES code or understand a solution. The Calculator Window allows you to enter and solve single-line expressions that involve any of the variables that have previously been defined and evaluated in the Equations Window. Any of the built-in functions in EES can be accessed from the Calculator Window. For example, enter and solve the following equations:

```
a=b+2
b^2=4+c
c=a-b
```

which should lead to the Solution Window shown in Figure 1-78.

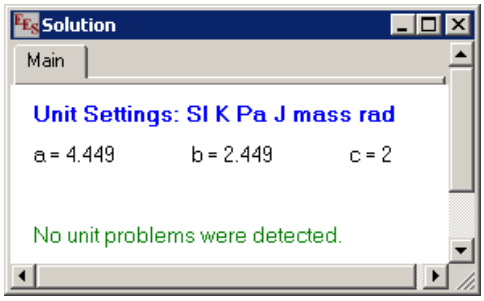


Figure 1-78: Solution Window.

Open the Calculator Window by selecting Calculator from the Windows menu (or selecting the button), as shown in Figure 1-79.

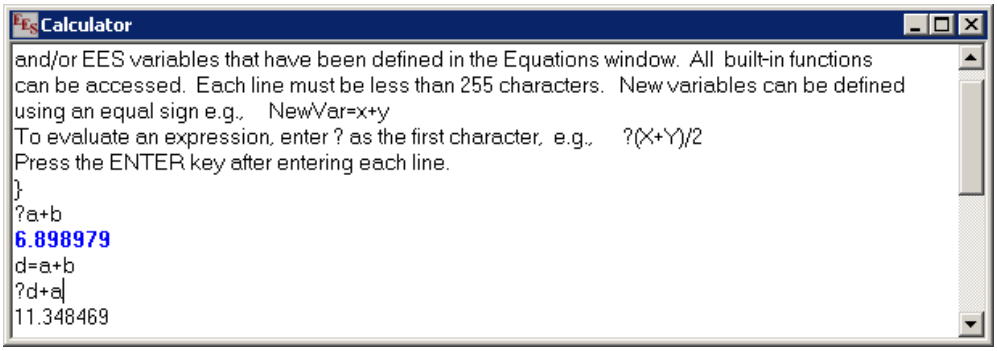


Figure 1-79: Calculator Window.

The result of any command preceded by a question mark will be displayed in the Calculator Window. For example, the command:

```
?a+b
```

will lead to 6.899, as shown in Figure 1-79. New variables can also be defined in the Calculator Window (provided that they don't already exist in the Equations Window). For example, we can define the variable d in the Calculator Window:

```
d=a+b
```

and then use the variable d in a new expression, also shown in Figure 1-79. Use % to refer the last calculated variable.

Preferences

Select Preferences from the Options menu in order to access the Preferences dialog shown in Figure 1-80. The Preferences dialog allows the user to specify a large number of characteristics of the EES program that are categorized according to the tabs at the bottom of the dialog. For example, in the Options tab shown in Figure 1-80 you can specify how often the file is auto-saved, whether units are checked automatically, and a number of other options. Each tab contains choices of this nature that will be discussed throughout this book.

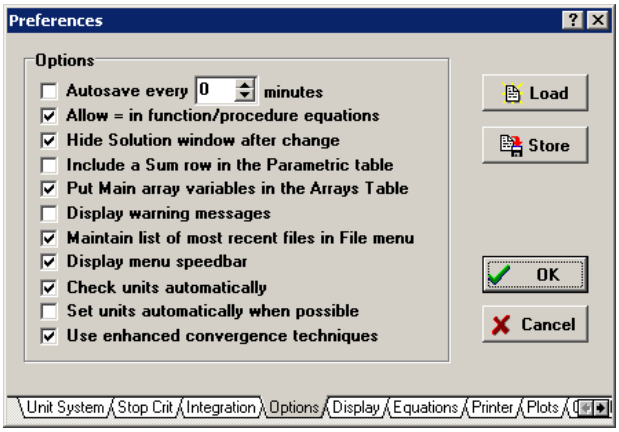


Figure 1-80: Preferences dialog.

Pressing OK will set the preferences you selected to be in effect for only the current session. However, selecting the Store button will cause the preferences to be saved in an EES Preferences (.prf) file. If you save the file as EES.prf in the EES working directory (which is the default selection that is used when you select Store), then these preferences will automatically be loaded whenever EES is started. Alternatively, you can save a .prf file that you want to apply only to the current file in an alternative directory and subsequently load it using the Load button in Figure 1-80,

Even more convenient is to load the preferences automatically using the $Include directive. The $Include directive is discussed more completely in Section 14.5. Briefly, it provides a method to automatically load various types of external files in an EES program. One type of file that can automatically be loaded is the Preferences file.

For example, select Preferences from the Options dialog and then select the Display tab. Adjust the font, size, etc. as you see fit and select Store. Save the .prf file in a directory of your

choosing (e.g., C:\temp\My Preferences.prf). Place an appropriate $Include directive at the top of your EES file.

```
$Include C:\temp\My Preferences.prf
```

If you close and re-open the file you should find that the selections that you made in the Preferences dialog and saved in the file My Preferences.prf are automatically loaded and used in your EES file.

References

Mohr, P.J., B.N. Taylor, and D.B. Newell, "The 2010 CODATA Recommended Values of the Fundamental Physical Constants (Web Version 6.0)," NIST, Gaithersburg, MD, 20899, (2011).

Nellis, G.F. and S.A. Klein, *Heat Transfer*, Cambridge University Press, New York, (2009).

Shah, R.K. and A.L. London, *Laminar Flow Forced Convection in Ducts*, Academic Press, New York, (1978).

2 CURVE FITTING & INTERPOLATION

Discrete data are often used to provide the basis for continuous functions. For example, we may measure the thermal conductivity of a material at a few discrete temperatures and then, either through curve fitting or interpolation, use these data to predict the thermal conductivity at any temperature within the measurement range (and even beyond). EES provides several powerful methods for accomplishing curve fitting and interpolation and these are discussed in this chapter.

2.1 Curve Fitting

This section provides information on how to fit a continuous curve to a set of data with one independent variable, e.g., thermal conductivity as a function of temperature.

Curve Fitting Plotted Data

Any data set that exist on a plot in EES can be used as the basis of a curve fit by selecting Curve Fit from the Plots menu. For example, consider the thermal conductivity data listed in Table 2-1.

Table 2-1: Thermal conductivity data.

Temperature (K)	Thermal conductivity (W/m-K)
200	4.53
250	5.21
300	6.05
350	7.02
400	8.48
450	10.00
500	12.11
550	15.58

These data can be entered in a Lookup Table as discussed in Section 1.8 and shown in Figure 2-1. The data in the Lookup Table can subsequently used to create a plot as discussed in Section 1.4 and shown in Figure 2-2(a).

Lookup Table

Conductivity

Paste Special	1 Temperature [K]	2 Conductivity [W/m-K]
Row 1	200	4.53
Row 2	250	5.21
Row 3	300	6.05
Row 4	350	7.02
Row 5	400	8.48
Row 6	450	10
Row 7	500	12.11

Figure 2-1: Lookup Table containing data in Table 2-1.

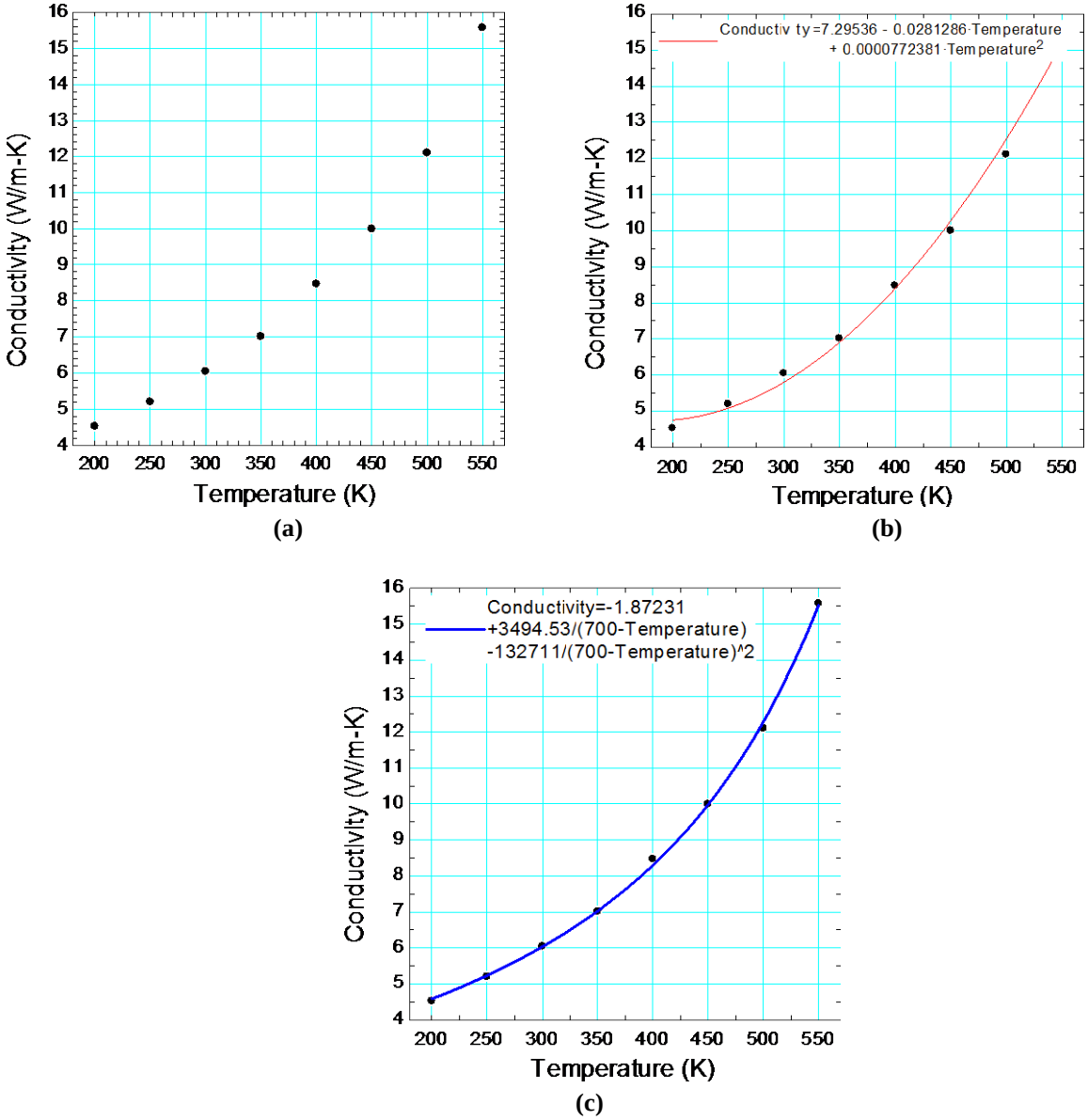


Figure 2-2: Plot (a) showing data only, (b) data with 2nd order polynomial curve fit shown, and (c) data with curve fit given by Eq. (2-3) shown.

Select Curve Fit from the Plots menu in order to access the Curve Fit Plotted Data dialog shown in Figure 2-3. The left window shows each of the data series that appear in the plot; in this case there is only conductivity as a function of temperature. On the right side, the user can select from a number of different mathematical forms for the curve fit. The corresponding equation is shown in the window at the bottom of the dialog.

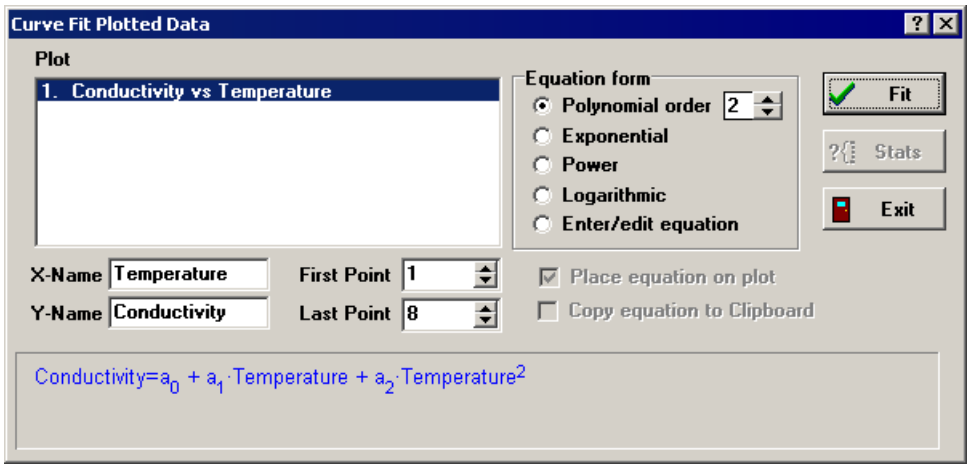


Figure 2-3: Curve Fit Plotted Data dialog with 2nd order polynomial fit selected.

In Figure 2-3, the curve fit selected is a second order polynomial and the equation that is listed is:

$$\text{Conductivity} = a_0 + a_1\,\text{Temperature} + a_2\,\text{Temperature}^2$$

When the Fit button is selected, EES will attempt to determine the best values of the undetermined coefficients in the equation (in this case the variables a_0, a_1, and a_2) using the linear least squares method. The result is shown in Figure 2-4; note that the equation is shown in red font and the unknown coefficients have been replaced with their best fit value.

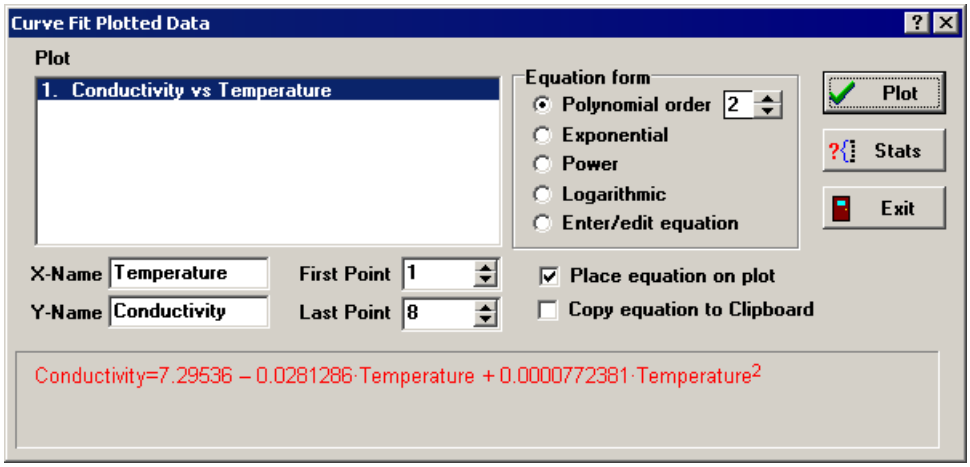


Figure 2-4: Curve Fit Plotted Data dialog with best-fit 2nd order polynomial fit shown.

There are several useful options that appear in the dialog once the curve fit has been determined. The Plot button causes the curve fit to be overlayed onto the plot. The Place equation on plot option causes the mathematical equation associated with the curve fit to be shown in the plot, as shown in Figure 2-2(b). The Copy equation to Clipboard option causes the equation to be placed on the clipboard so that it can subsequently be pasted into the Equations Window or another application. The Stats button allows access to the Curve Fit Statistics dialog that is shown in Figure 2-5.

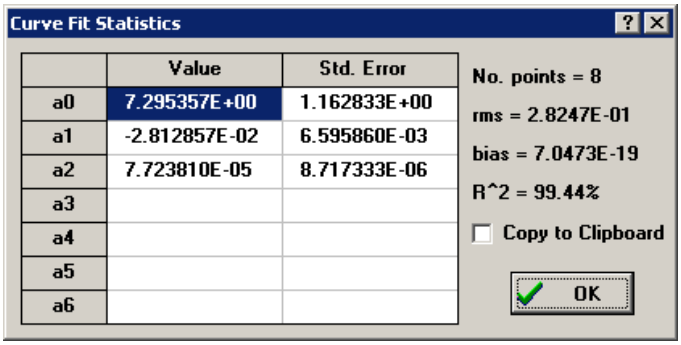


Figure 2-5: Curve Fit Statistics dialog.

The Curve Fit Statistics dialog lists the value and the standard error associated with each coefficient. The rms and bias errors of the curve fit are also shown. The rms error of a curve fit is computed according to:

$$rms = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - Y_{m,i}\right)^2} \tag{2-1}$$

where N is the number of data points, Y_i is the value of the dependent variable, and $Y_{m,i}$ is the value of the dependent variable that is estimated using the curve fit. The bias error is computed according to:

$$bias = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i - Y_{m,i}\right) \tag{2-2}$$

The rms error provides an indication of the deviation between the curve fit and the data. The bias error provides an indication of a more systematic overestimate or underestimate that is associated with the curve fit.

It is also possible to enter a custom equation form into the Curve Fit Plotted Data dialog. Any unknown coefficients that must be determined in order to complete the curve fit should be entered as a0, a1, etc. For example, you may want to curve fit the equation:

$$k = a_0 + \frac{a_1}{\left(700 - T\right)} + \frac{a_2}{\left(700 - T\right)^2} \tag{2-3}$$

where a_0, a_1, and a_2 are coefficients that should be adjusted in order to minimize the difference between the data and the curve fit. Select Curve Fit from the Plot menu and then select Enter/edit equation, as shown in Figure 2-6. Select Fit in order to bring up the dialog shown in Figure 2-7, which allows the user to enter guess values and limits on the unknown coefficients. Select OK in order to obtain the best fit values of a_0, a_1, and a_2. The resulting curve fit is shown in Figure 2-2(c).

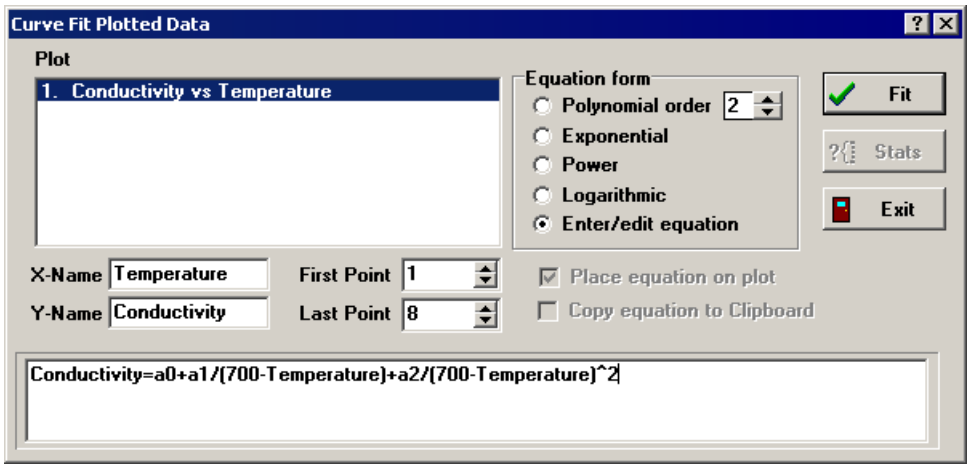


Figure 2-6: Curve Fit Plotted Data dialog with user-specified equation entered.

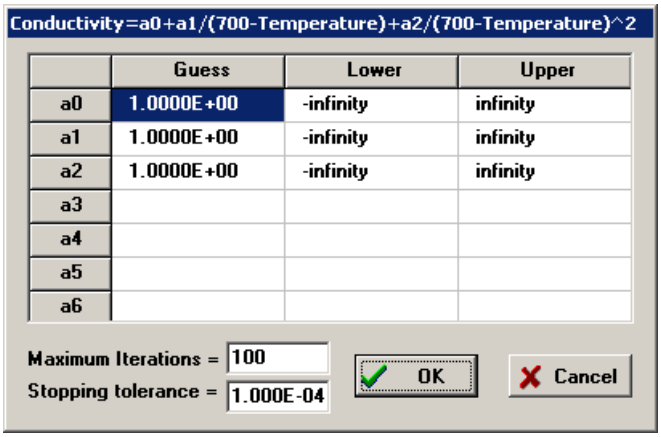


Figure 2-7: Dialog to set guess values and limits on unknown coefficients.

Curve Fitting Array Data

The CurveFit1D procedure in EES allows the standard curve fit options that can be accessed from the Curve Fit Plotted Data dialog, shown in Figure 2-3, to be applied to a data set that is not plotted but rather exists in array form. For example, the data in Table 2-1 can be entered into arrays in the Equations Window using a Duplicate loop and array notation (see Section 1.7).

```
Duplicate i=1,8
    T[i]=200+(i-1)*50
end
k[1..8]=[4.53, 5.21, 6.05, 7.02, 8.48, 10.00, 12.11, 15.58]
```

Note that if these data exist in a Lookup Table, as shown in Figure 2-1, then they can be placed in arrays using the Lookup command that is discussed in Section 1.8.

```
Duplicate i=1,8
    T[i]=Lookup('Conductivity',i,1)
    k[i]=Lookup('Conductivity',i,2)
end
```

The calling protocol for the CurveFit1D procedure is:

```
Call CurveFit1D(FitType$, X[1..N], Y[1..N]: a0, a1, ..., aM, RMS, Bias, R|2, a_stderr[1..M])
```

Procedures are discussed in more detail in Section 3.3. Procedures are very similar to functions in EES, which we have already encountered several times. Procedures allow multiple outputs and must be preceded by the Call statement. Any argument to the left of the colon is an input and those to the right of the colon are outputs. Inputs are typically provided to the procedure while output variables are assigned by the procedure. For the procedure CurveFit1D, the first input is FitType$ which is a string that specifies the functional form of the curve fit; the choices correspond to the standard choices that are available in the Curve Fit Plotted Data dialog in Figure 2-3 and are summarized in Table 2-2.

Table 2-2: Summary of curve fit types that are available in the CurveFit1D procedure.

Description	FitType$ String	Form of curve fit
Linear fit	'Linear'	$y = a_0 + a_1 x$
1st order polynomial	'Polynomial 1' or 'Poly1'	$y = a_0 + a_1 x$
2nd order polynomial	'Polynomial 2' or 'Poly2'	$y = a_0 + a_1 x + a_2 x^2$
3rd order polynomial	'Polynomial 3' or 'Poly3'	$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$
4th order polynomial	'Polynomial 4' or 'Poly4'	$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$
5th order polynomial	'Polynomial 5' or 'Poly5'	$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$
6th order polynomial	'Polynomial 6' or 'Poly6'	$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6$
Exponential fit	'Exponential' or 'Exp'	$y = a_0 \exp(a_1 x)$
Power	'Power'	$y = a_0 x^{a_1}$
Logarithmic	'Logarithmic' or 'Log'	$y = a_0 + a_1 \ln(x)$

The second and third input arguments are the dependent and independent data arrays, respectively. The first outputs are the unknown coefficients provided in the array a that has *m* elements, where *m* depends on the form of the curve fit. The remaining outputs are optional (that is, variables that do not need to be provided and assigned by the procedure). The outputs RMS and Bias are the rms and bias errors associated with the curve fit, defined by Eqs. (2-1) and (2-2), respectively. The output R|2 is the correlation coefficient between the dependent and independent variables. The array a_stderr is the standard error of the curve fit coefficients, defined as the square root of the estimated variance of the parameter. The outputs RMS, Bias, R|2, and a_stderr are optional and can be left out of the calling statement.

The EES code below:

```
Call CurveFit1D('Poly2',T[1..8],k[1..8]: a0, a1, a2, RMS, Bias, R|2)
```

leads to the Solutions Window shown in Figure 2-8. Note that the coefficients are identical to those shown in Figure 2-2(b) and Figure 2-4 and the statistics (e.g., the rms and bias errors) are identical to those shown in Figure 2-5.

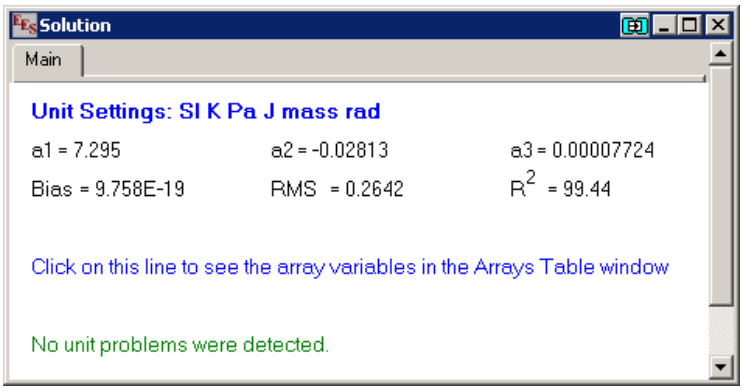


Figure 2-8: Solutions Window.

2.2 Linear Regression

The curve fitting options discussed in Section 2.1 are useful for situations where a dependent variable is regressed to a single independent variable. For example, in Table 2-1 thermal conductivity is assumed to be only a function of temperature. Often a dependent variable will be a function of several independent variables. For example, refrigeration compressor performance data is often provided as a function of both evaporating and condensing temperature. An example of such data can be found in the Lookup Table CompressorMap.lkt which is located in the UserLib\Examples subfolder within the folder that the contains the EES application. Open this Lookup Table by selecting Open Lookup Table from the Tables menu; the first few rows of the table are shown in Figure 2-9.

CompressorMap

Paste Special	1 T_{cond} [°F]	2 T_{evap} [°F]	3 $\dot{m}$ [lb_m/hr]	4 Power [W]
Row 1	70	10	1730	8800
Row 2	70	15	1950	9030
Row 3	70	20	2200	9210
Row 4	70	25	2470	9320
Row 5	70	30	2760	9330
Row 6	70	35	3090	9240
Row 7	70	40	3450	9020
Row 8	70	45	3850	8660
Row 9	80	10	1590	9750
Row 10	80	15	1800	10070
Row 11	80	20	2040	10400
Row 12	80	25	2300	10600
Row 13	80	30	2580	10700
Row 14	80	35	2900	10700

Figure 2-9: CompressorMap Lookup Table.

Notice that both the mass flow rate and power (two dependent variables) are provided for an array of condensing and evaporating temperatures (two independent variables). It would not be appropriate to use the curve fitting options discussed in Section 2.1 for this situation. The advantage of the Linear Regression command in EES is that it allows multiple (up to 9) independent variables to be considered. Select Linear Regression from the Tables menu to obtain the Linear Regression dialog shown in Figure 2-10.

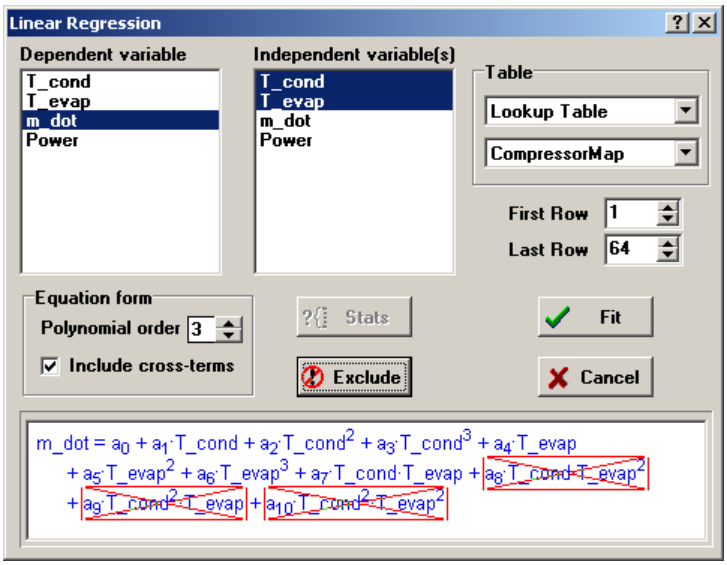


Figure 2-10: Linear Regression dialog while equation form is being selected.

Selecting the Equation Form

The upper right part of the Linear Regression dialog allows you to select the source of the data; the Lookup Table CompressorMap was selected in Figure 2-10, but linear regression can also be applied to Parametric Tables, Integral Tables, and Arrays. Select the dependent variable of interest (e.g., the variable m_dot) and the independent variables (e.g., the variables T_cond and T_evap). In the bottom box, the dependent variable will be displayed as a function of the independent variables using a polynomial fit. The order of the polynomial can be selected under Equation form and you can include or exclude cross-terms. In Figure 2-10, a third-order polynomial is used with cross-terms. Note that any specific term in the equation can be removed from the fit by selecting that term and hitting the Exclude button. For example, in Figure 2-10 all of the higher order cross-terms have been removed. Select the Fit button to initiate the fitting process that determines each of the unknown coefficients. The fitted equation will be displayed in the bottom box, as shown in Figure 2-11.

Equation Statistics

Selecting the Stats button provides the standard error of each coefficient as well as the rms error, bias error, and R^2 parameter of the curve fit as shown in Figure 2-12. The coefficients can be copied to the clipboard from this dialog.

Regression Plot

Select Plot in order to generate a plot showing the predicted value of the dependent variable obtained using the curve fit as a function of the value of the independent variable for each of the data points, as shown in Figure 2-13. The red line indicates a perfect fit and the scatter of the data about this line indicates the goodness of the fit.

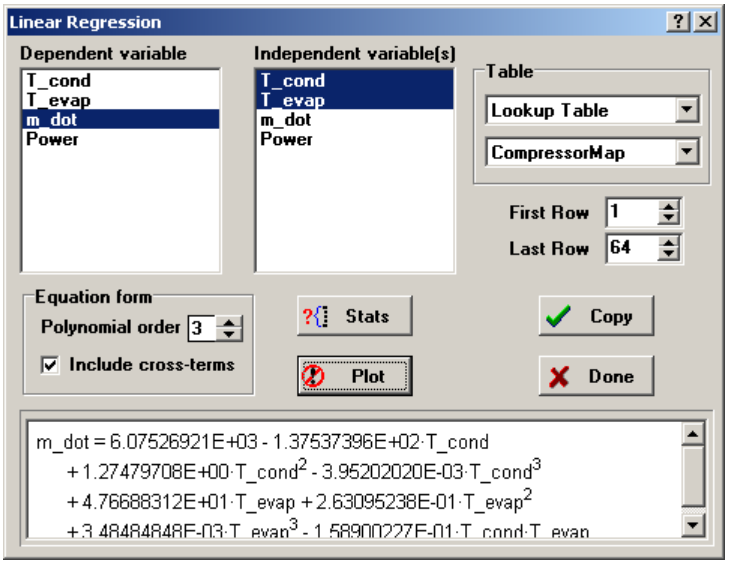


Figure 2-11: Linear Regression dialog with fitted equation shown.

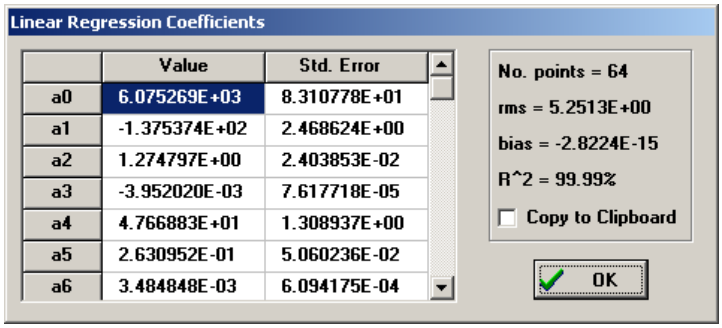


Figure 2-12: Linear Regression Coefficients dialog.

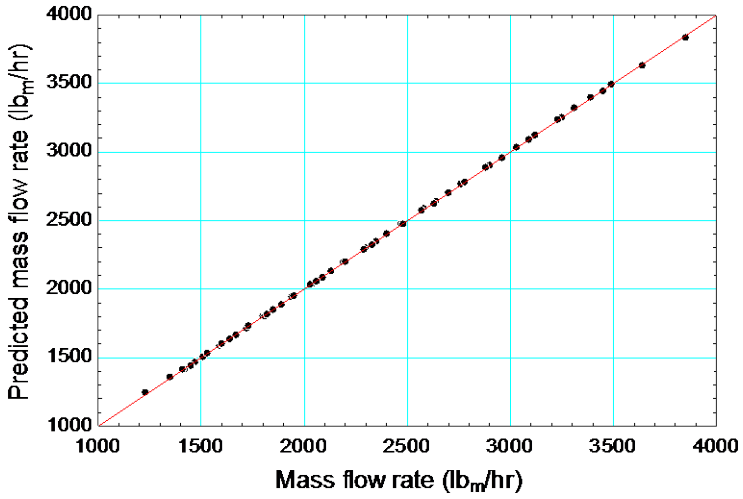


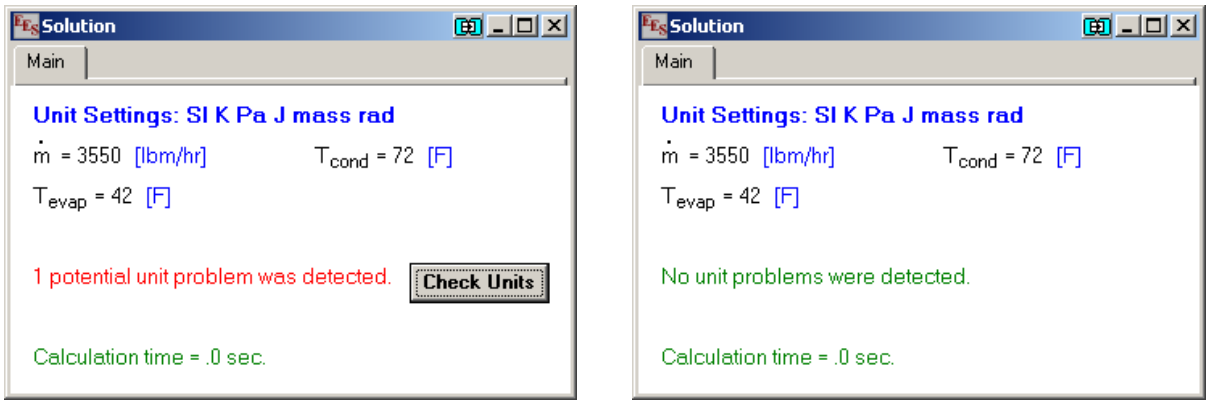
Figure 2-13: Predicted value of mass flow rate as a function of the mass flow rate.

Copying the Regression Equation

The Copy button copies the equation to the clipboard so that it can be pasted into another application or into the Equations window:

```
T_evap=42 [F]
T_cond=72 [F]

m_dot=6.07526921E+03-1.37537396E+02*T_cond+1.27479708E+00*T_cond^2&
    -3.95202020E-03*T_cond^3+4.76688312E+01*T_evap+2.63095238E-01*T_evap^2&
    +3.48484848E-03*T_evap^3-1.58900227E-01*T_cond*T_evap
```

Note that running the program will result in the Solutions Window shown in Figure 2-14(a), which shows that there are unit warnings, even though the units of the variables T_cond, T_evap, and m_dot have all been set correctly (to °F, °F, lb_m/hr, respectively).



(a) **(b)**

Figure 2-14: Solutions Window (a) with unit warning and (b) without unit warnings.

The $CheckUnits Directive

This unit warning occurs because the units of the constants in the curve fit have not been set correctly. To overcome this problem, either enter units manually:

```
m_dot=6.07526921E+03 [lbm/hr]-1.37537396E+02[lbm/hr-F]*T_cond+&
    1.27479708E+00[lbm/hr-F^2]*T_cond^2-3.95202020E-03[lbm/hr-F^3]*T_cond^3&
    +4.76688312E+01[lbm/hr-F]*T_evap+2.63095238E-01[lbm/hr-F^2]*T_evap^2&
    +3.48484848E-03[lbm/hr-F^3]*T_evap^3-1.58900227E-01[lbm/hr-F^2]*T_cond*T_evap
```

or use the $CheckUnits directive to disable unit checking for the curve fit. The $CheckUnits directive allows unit checking to be selectively disabled for an equation or set of equations; $CheckUnits Off disables unit checking and $CheckUnits On restores unit checking.

```
$CheckUnits Off
m_dot=6.07526921E+03-1.37537396E+02*T_cond+1.27479708E+00*T_cond^2&
    -3.95202020E-03*T_cond^3+4.76688312E+01*T_evap+2.63095238E-01*T_evap^2&
    +3.48484848E-03*T_evap^3-1.58900227E-01*T_cond*T_evap
$CheckUnits On
```

Either option should result in the Solutions Window shown in Figure 2-14(b).

2.3 One Dimensional Interpolation

The curve fit and linear regression options discussed in Sections 2.1 and 2.2 provide methods for obtaining mathematical functions that represent an underlying discrete data set. An alternative technique is interpolation, where some mathematical technique is used to estimate values of the independent variable that fall between the values that are provided in the underlying data set.

Interpolation in EES is accomplished using data provided in a Lookup Table. For example, the data in Table 2-1 are entered in the Lookup Table shown in Figure 2-1. The Interpolate1, Interpolate2, and Interpolate3 functions in EES provide 1-D interpolation using linear, quadratic, and cubic interpolation methods, respectively. The Interpolate function is identical to the Interpolate3 function and uses cubic interpolation. The calling protocol for each of these functions is the same:

```
Interpolate('Table Name', 'Dependent Variable', 'Independent Variable', Independent Variable = Value)
```

The string variable or constant 'Table Name' corresponds to the name of the Lookup Table to use for the interpolation. The string variables or constants 'Dependent Variable' and 'Independent Variable' correspond to the column header names associated with the dependent and independent variables, respectively. The final argument specifies the value of the independent variable using the numerical value or expression Value. For the Interpolate function to work properly, it is necessary that the entries in the column containing the independent variable vary monotonically (either increasing or decreasing).

To carry out cubic interpolation using the data in the Lookup Table 'Conductivity' use the command:

```
T=311 [K]                                                    "Temperature"
k=Interpolate('Conductivity', 'Conductivity', 'Temperature', Temperature=T)
```

which returns k = 6.24 W/m-K. If the value of the independent variable lies outside of the range included in the table, then the Interpolate commands will attempt to extrapolate but EES will display a warning. For example, the code:

```
T=150 [K]                                                    "Temperature"
k=Interpolate('Conductivity', 'Conductivity', 'Temperature', Temperature=T)
```

will lead to the reasonable extrapolation k = 4.04 W/m-K, but it will also lead to the warning shown in Figure 2-15 (provided that Display Warning Messages is selected in the Options tab of the Preferences dialog).

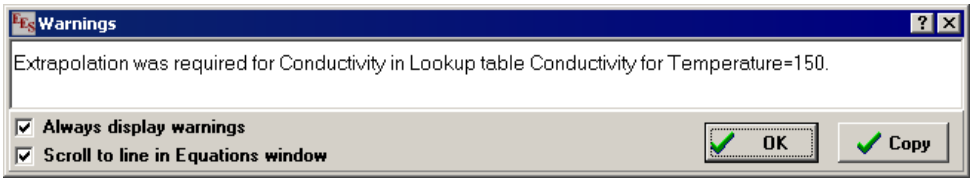


Figure 2-15: Warning message indicating that extrapolation was required.

Note that it is not necessary to load a Lookup Table into the EES program in order to use it as the basis for interpolation. The string 'Table Name' can also refer to the name of an existing Lookup Table that is stored in a file with an .lkt file name extension. In this case, the .lkt file name extension must be included in 'Table Name'.

2.4 Two Dimensional Interpolation

The Interpolate2D and Interpolate2DM function allow a dependent variable to be expressed as a function of two independent variables by interpolation of data that are stored in a table.

The Interpolate2D Function

By default, the Interpolate2D function utilizes bi-quadratic interpolation employing the 16 nearest data points but it is also possible to use radial basis functions for the interpolation process. The Interpolate2D function operates on a table in which the dependent and independent variables occupy single columns. An example is the CompressorMap.lkt Lookup Table shown in Figure 2-9 in which the independent variables are the condenser and evaporator temperatures contained in the columns T_cond and T_evap while the dependent variable is the mass flow rate contained in the column m_dot. The calling protocol for the Interpolate2D function is:

```
Interpolate2D('Table Name', 'Independent Variable 1', 'Independent Variable 2', 'Dependent Variable',
    Independent Variable 1 = Value1, Independent Variable 2 = Value2)
```

The parameter 'Table Name' refers to the Lookup Table where the data are stored (again, either in the EES program or in a disk file). The parameters 'Independent Variable 1' and 'Independent

Variable 2' are the column headings corresponding to the two independent variables while the parameters 'Dependent Variable' is the column heading corresponding to the dependent variable.

For example, the following code uses the Interpolate2D function in conjunction with the CompressorMap.lkt Lookup Table shown in Figure 2-9:

```
T_evap=42 [F]
T_cond=72 [F]
m_dot=Interpolate2D('CompressorMap', 'T_cond', 'T_evap','m_dot' , T_cond=T_cond, T_evap=T_evap)
```

and leads to $\dot{m}$ = 3561 lb_m/hr; this value is similar to the value obtained using the linear regression curve fit ($\dot{m}$ = 3550 lb_m/hr) at the same conditions in Figure 2-14.

The Interpolate2D function will accept a seventh, optional parameter, N, that controls the interpolation algorithm. If N is less than zero then the default method, bi-quadratic interpolation, is used. If N is greater than zero then a multi-quadric radial basis function interpolation method is used. The absolute value of the variable N sets the maximum number of data points that are used in the interpolation process. For example, if N is set to -32 then bi-quadratic interpolation with 32 data points will be used but if N is set to 32 then multi-quadric radial basis functions with 32 data points will be used. If N is not specified then 16 data points are used. More data points will lead to more accurate results, but the interpolation process will be more computationally intensive. The minimum number of data points that can be specified is 8 or the number of data points in the table. In order to use radial basis functions with 32 data points, enter the following equation:

```
T_evap=42 [F]
T_cond=72 [F]
m_dot=Interpolate2D('CompressorMap', 'T_cond', 'T_evap','m_dot' , &
    T_cond=T_cond, T_evap=T_evap, 32)
```

which leads to $\dot{m}$ = 3552 lb_m/hr.

The Interpolate2DM Function

Because the Interpolate2D function uses nonlinear interpolation methods, the underlying data set does not need to be provided at equal values of the independent variables. The Interpolate2DM function uses two-dimensional linear interpolation and therefore the data must be provided in a uniform matrix. For example, the CompressorMap.lkt Lookup Table shown in Figure 2-9 is placed in matrix format in Figure 2-16. The first row of the table must contain each of the values of the first independent variable. In Figure 2-16, the first independent variable is the condenser temperature. The first column must contain each of the values of the second independent variable. In Figure 2-16, the first column is the evaporator temperature. The remaining entries are the values of the dependent variable at each unique combination of independent variables; for example, the mass flow rate at T_{evap} = 35ºF and T_{cond} = 100ºF is $\dot{m}$ = 2700 lb_m/hr.

Lookup Table

CompressorMap | CompressorMapMatrix

Paste Special

	1 T_{evap} [F]	2 $\dot{m}_{70F}$ [lbm/hr]	3 $\dot{m}_{80F}$ [lbm/hr]	4 $\dot{m}_{90F}$ [lbm/hr]	5 $\dot{m}_{100F}$ [lbm/hr]	6 $\dot{m}_{110F}$ [lbm/hr]	7 $\dot{m}_{120F}$ [lbm/hr]	8 $\dot{m}_{130F}$ [lbm/hr]	9 $\dot{m}_{140F}$ [lbm/hr]
Row 1		70	80	90	100	110	120	130	140
Row 2	10	1730	1590	1510	1470	1450	1420	1350	10
Row 3	15	1950	1800	1720	1670	1640	1600	1530	15
Row 4	20	2200	2040	1940	1890	1850	1810	1730	20
Row 5	25	2470	2300	2190	2130	2090	2030	1950	25
Row 6	30	2760	2580	2470	2400	2350	2290	2200	30
Row 7	35	3090	2900	2780	2700	2640	2570	2480	35
Row 8	40	3450	3250	3120	3030	2960	2880	2780	40
Row 9	45	3850	3640	3490	3390	3310	3230	3120	45

Figure 2-16: CompressorMap lookup table converted to matrix format.

The calling protocol for the function Interpolate2DM is:

Interpolate2DM('Table Name', Independent Variable 1 Value, Independent Variable 2 Value)

The variable 'Table Name' again refers to the name of the Lookup Table or Lookup file on disk. The values of the two independent variables follow; Independent Variable 1 Value is the value of the independent variable that defines the individual columns (e.g., the condenser temperature in Figure 2-16) and Independent Variable 2 Value is the value of the independent variable that defines the individual rows (e.g., the evaporator temperature in Figure 2-16). The EES code:

```
T_evap=42 [F]
T_cond=72 [F]
m_dot=Interpolate2DM('CompressorMapMatrix',T_cond,T_evap)
```

provides $\dot{m}$ = 3569 lb_m/hr, which is similar to the value obtained using the Interpolate2D function.

3 FUNCTIONS AND PROCEDURES

An EES function is a code segment that accepts one or more inputs and returns a single result associated with the Function name. An EES procedure is similar to a function, but it can return one or more results and it is accessed by the Call statement. The code that is employed in functions and procedures is dramatically different from the code used in the main part of an EES program; functions and procedures utilize assignment statements rather than equalities. Functions and procedures provide several important advantages. First, they make it easier to program a large model by allowing the code to be broken up into a number of smaller parts that are each easier to understand and debug. Second, EES functions and procedures allow the use of programming logic statements, such as If-Then-Else, Repeat-Until and GoTo statements, which cannot be used in the main body of EES. Third, functions and procedures can be saved in a file after they have been debugged and verified, allowing them to be re-used in other EES programs or in a library file, as explained in Chapter 11. This chapter presents the basic capabilities of functions and procedures. Subprograms and modules share some of the advantages of functions and procedures but utilize equations rather than assignments as discussed in Chapter 10.

3.1 Equations and Assignment Statements

The instructions that are entered into the main body of an EES program provide a set of equations rather than assignment statements. Assignment statements are used in most high-level programming languages. The difference between an equation and an assignment statement can be understood with the following simple statement:

```
X = X + 1
```

This statement can not be a valid equation, since the variable X can never equal X + 1. If you enter this statement into the EES Equation Window, it will attempt to solve it numerically, using the methods described in Section 5.1. With the default stopping criteria, EES will find a solution of $X = 5x10^{17}$. With this value of X, the residual (which is related to the difference between the left and right sides of the equation) is smaller than the default tolerance; therefore, it appears to EES to be a valid solution. However, this is probably not the solution that was intended.

The equation X = X+1 is an assignment statement. Assignment statements explicitly assign the value of the variable on the left side of the equal sign (in this case X) to the value of the expression on the right side of the equal sign (in this case X + 1). All of the variables appearing on the right side of the equal sign in an assignment statement must have previously defined values. For example, if the current value of X is 8 then the execution of this assignment statement would change it to be 9.

The Assignment Operator

All of the equations appearing in EES functions and procedures must be assignment statements rather than equations. Early versions of EES required that assignment statements use the assignment operator (:=) in place of an equal sign. Thus, the assignment statement above would appear as:

```
X:=X+1
```

This requirement has been relaxed in current versions of EES by the implementation of the option Allow = in Function/Procedure equations that can be selected in the Options tab of the Preferences dialog as shown in Figure 3-1. This option is selected by default and it causes EES to accept both the equal sign and the assignment operator in functions and procedures.

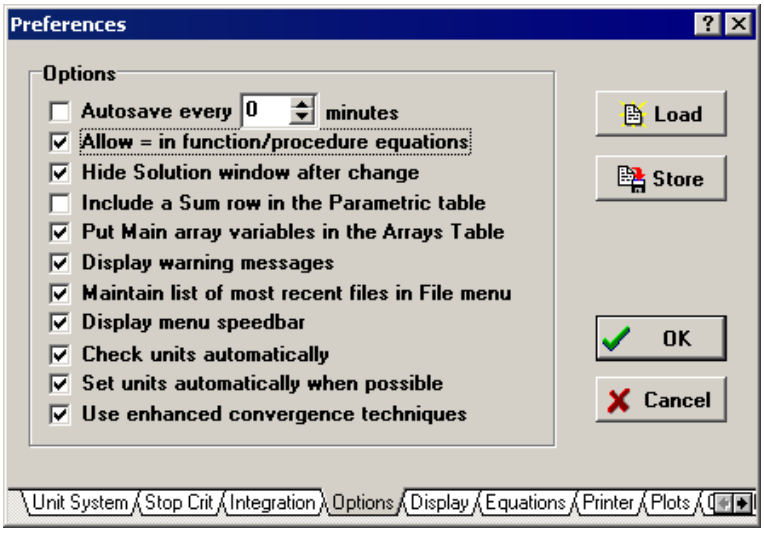


Figure 3-1: Preferences dialog showing the option that allows the = sign for assignment statements.

Assignment statements are much more structured than equations. Each statement entered in a function or procedure must be arranged so that the variable that is being assigned a value appears by itself on the left side of the statement. The statements are executed in exactly the order that they are entered. A major advantage of assignment statements is that they allow the use of logic constructs, such as If-Then-Else statements, as described in Section 3.4.

3.2 Functions

Internal EES functions can be written directly in the Equations Window. External functions can be written in any compiled language, as discussed in Chapter 19.

Format of Functions

Internal functions must be formatted as shown below:

```
Function Function_Name(Input 1, Input 2, ..., Input N)

    Assignment statement(s) - note that one of these must have the form:
    Function_Name = ...

End
```

Function declarations must appear at the top of the Equations Window, before any of the equations in the main body of the EES program. Function declarations must begin with the keyword Function. The function name (Function_Name) and arguments (Input 1, etc.) follow on the same line. The arguments are enclosed in parentheses and separated by a list delimiter (which is the comma for the U.S. system and the semicolon for the European system). The function declaration is terminated by the keyword End. The statements appearing in functions (and procedures) must be assignment statements (as discussed in Section 3.1) or logic statements (discussed in Section 3.4). EES processes these statements in the order that they appear unless directed otherwise by the logic statements.

Functions are called by using their name in an equation:

```
X = Function_Name(Input 1, Input 2, ..., Input N)
```

A function must have at least one argument and it must be called with the same number of arguments that appear in the Function declaration. The names of the arguments in the calling statement need not match the name of the arguments in the function declaration; only their order matters. The statements within the function can only refer to variables that are passed to the function as input arguments or previously defined within the function itself. All variables used in the body of a function are local to the function except global variables in the main body of an EES program that have been defined using the $Common directive, which is described in Section 14.3. Functions return the value assigned in the statement Function_Name = ... in the function body.

Functions may refer to any other built-in function, procedure, or other program entities that have been loaded using a library file (see Chapter 11). However functions cannot call themselves, i.e., they cannot be used recursively. Functions may not call a module but they may call subprograms (described in Chapter 10). Functions (and procedures) MUST use the unit system settings that are set in the main program. Different unit system for each function or procedure can not be specified.

First Example of a Function

The format and utility of functions will be demonstrated with a few examples. The first example returns the Darcy friction factor (*f*) for internal flow given inputs of Reynolds number (*Re*) and relative roughness (*RR*). The correlation that we will use was developed by Churchill (1977) and it shown in Eq. (3-1).

$$f = 8\left\{\left(\frac{8}{Re}\right)^{12} + \left[\left(2.457\ \ln\left[\frac{1}{\left(\frac{7}{Re}\right)^{0.9} + 0.27\,RR}\right]\right)^{16} + \left(\frac{37530}{Re}\right)^{16}\right]^{-1.5}\right\}^{1/12} \quad (3\text{-}1)$$

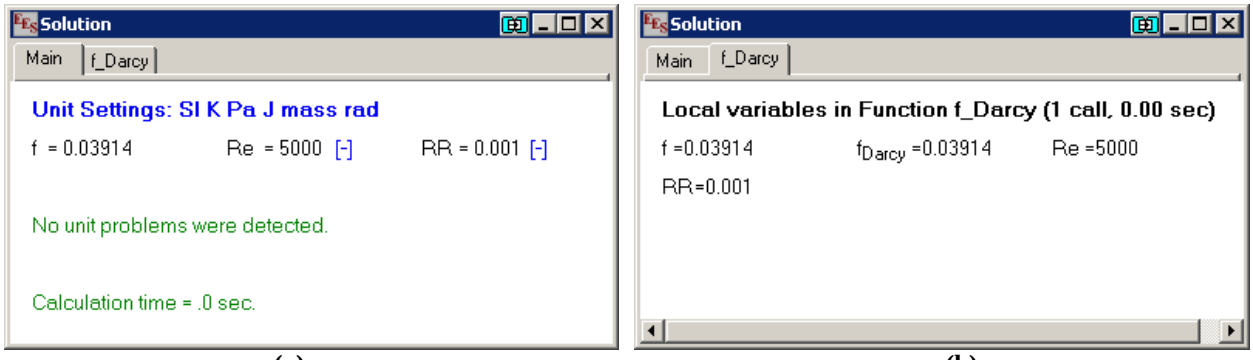
The function that returns the friction factor f will be called f_Darcy. A listing of the function follows.

```
Function f_Darcy(Re,RR)
    f=8*((8/Re)^12+((2.457*ln(1/((7/Re)^0.9+0.27*(RR))))^16+(37530/Re)^16)^(-1.5))^(1/12)
    f_Darcy=f
end
```

The Function should appear at the top of the Equations Window, before any of the equations in the main section of the EES program. The Function is accessed by its name with values provided for Reynolds number and relative roughness. The following code, entered after the function declaration:

```
Re=5000 [-]                    "Reynolds number"
RR = 0.001 [-]                 "relative roughness"
f=f_Darcy(Re,RR)               "Darcy friction factor"
```

will result in the Solution Window shown in Figure 3-2(a). Notice that in addition to the Main tab that shows the variables in the Equations Window there is now a second tab with a label that corresponds to the function workspace. Selecting the f_Darcy tab brings up the Solutions Window shown in Figure 3-2(b), which contains the values of the variables in the function workspace during the last call to the function.



(a) **(b)**

Figure 3-2: Solutions Window showing (a) Main and (b) f_Darcy tabs.

Second Example of a Function

As a second example, we will develop a function that implements the Peng-Robinson equation of state, shown in Eq. (3-2), in order to estimate the pressure of a fluid (P) given its specific volume (v) and temperature (T). See Klein and Nellis (2012) for more information about this and other equations of state.

$$P = \frac{RT}{(v-b)} - \frac{a}{v(v+b)+b(v-b)} \tag{3-2}$$

For the fluid of interest, the parameters in the Peng-Robinson equation of state are R = 188.9 J/kg-K, a = 70.89 N-m^4/kg^2 and b = 0.0006059 m^3/kg. The function is named PR and placed at the top of the Equations Window:

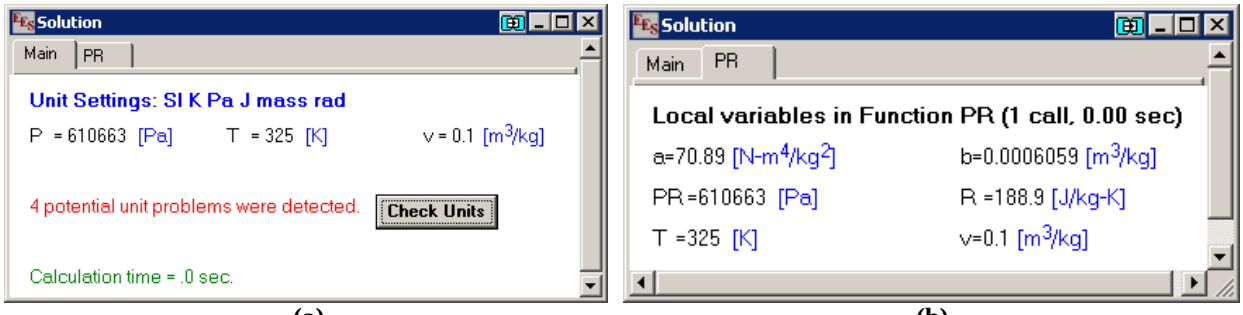
```
Function PR(v,T)
    R = 188.9 [J/kg-K]
    a = 70.89 [N-m^4/kg^2]
    b = 0.0006059 [m^3/kg]
    PR = R*T/(v-b)-a/(v*(v+b)+b*(v-b))
End
```

Note that the units of the constants are specified within square brackets, as discussed in Section 1.5. The function PR is called in order to estimate the pressure at v = 0.1 m^3/kg and T = 325 K with the following code:

```
v=0.1 [m^3/kg]          "specific volume"
T=325 [K]               "temperature"
P=PR(v,T)               "pressure, estimated from the Peng-Robinson equation of state"
```

Setting and Checking Units for Function Variables

Again, the Solution Window will have two tabs: one for the Main Equations Window and one for the PR Function workspace. Additional functions or procedures will result in additional tabs. The units of each variable in the Main program can be set by right-clicking on the variable name in the Main tab of the Solutions Window, which is shown in Figure 3-3(a).



(a) **(b)**

Figure 3-3: (a) Main tab of the Solutions Window showing unit warnings and (b) PR function tab showing units set for each variable.

Notice that even with the units of all of the variables in the Equations Window set correctly, EES is still reporting several possible unit problems. Select Check Units in order to view the Check Units dialog shown in Figure 3-4. The unit warnings result from the fact that the units of the variables used within the PR function workspace have not been set and are therefore inconsistent with each other and also inconsistent with the units of the arguments passed back and forth between the function and the Equations Window. EES checks the unit consistency of the variables in functions and procedures (and modules and subprograms, discussed in Chapter 10) as well as the variables in the Equations Window. Select the PR tab in the Solutions Window and specify the units of each of the variables in the PR workspace, as shown in Figure 3-3(b). Note that the units of each of the variables internal to the PR workspace (a, b, and R) as well as the input arguments (v and T) and the output (PR) must all be set to avoid unit consistency warning .

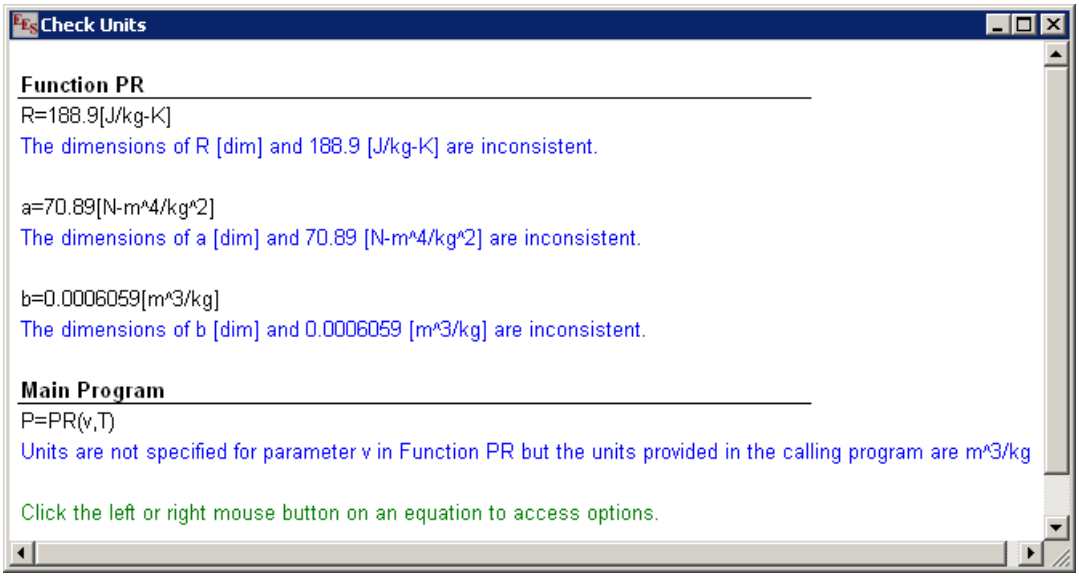


Figure 3-4: Check Units dialog.

Variable Information Page for Function

There is also a page in the Variable Information dialog corresponding to each defined function and procedure. Open the Variable Information Window (select Variable Info from the Options menu) and then select Function PR from the drop down menu to access the dialog shown in Figure 3-5. The units of each variable in the PR workspace can also be set using the Variable Information Window as can the display format. Guess values and limits are not applicable in a function or procedure because they use assignment statements rather than equations.

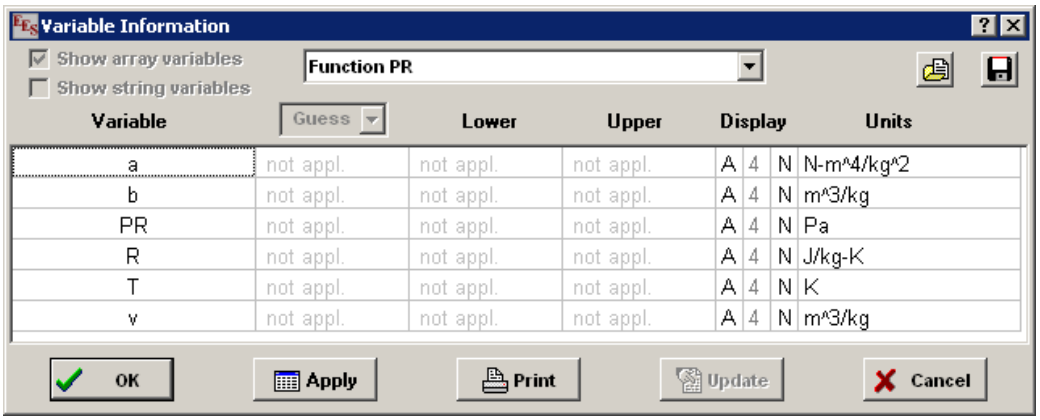


Figure 3-5: PR Function page of the Variable Information dialog.

3.3 Procedures

EES procedures are very much like EES functions, except that they allow multiple outputs and must be accessed using the Call command.

Format of Procedures

The format of a procedure is shown below:

```
Procedure Procedure_Name(Input 1, Input 2, ..., Input N: Output 1, Output 2, ..., Output M)

    Assignment statement(s) - note that some of these must have the form:
    Output 1 = ...
    Output 2 = ...
    ...
    Output M = ...

End
```

Procedure declarations must appear at the top of the Equations Window together with any function declarations and they must begin with the keyword Procedure. The procedure name (Procedure_Name) and argument list follow on the same line. The argument list is enclosed in parentheses. The first set of arguments are inputs (Input 1, Input 2, etc.); these are values that are typically known and specified at the time that the procedure is called. The input list is terminated by a colon that is followed by the output list (Output 1, Output 2, etc.). The outputs must be calculated within the procedure body. The procedure is terminated by the keyword End. Like functions, discussed in Section 3.2, the code used to write procedures must be assignment statements or logic statements. EES processes these statements in the order that they appear.

Procedures are accessed using the Call command:

```
Call Procedure_Name(Input 1, Input 2, ..., Input N: Output 1, Output 2, ..., Output M)
```

Following the Call command, the values of the output variables (Output 1, Output 2, etc.) will be assigned by the procedure. A procedure must be called with the same number of arguments that appear in the Procedure statement. The name of the arguments in the calling statement need not match the name of the arguments in the Procedure statement; only their order matters. The arguments may be constants, string variables, numerical variables, or algebraic expressions. Just as with a function, the procedure has its own local workspace. Statements within the procedure cannot refer to variables that are not passed to the procedure, previously defined within the procedure itself, or defined to be global using the $Common directive.

Functions may refer to any other built-in function, procedure, or other program entities that have been loaded using a library file (Chapter 11). However procedures, like functions, cannot call themselves. Procedures may not call a module but they may call subprograms (described in Chapter 10).

Example of a Procedure

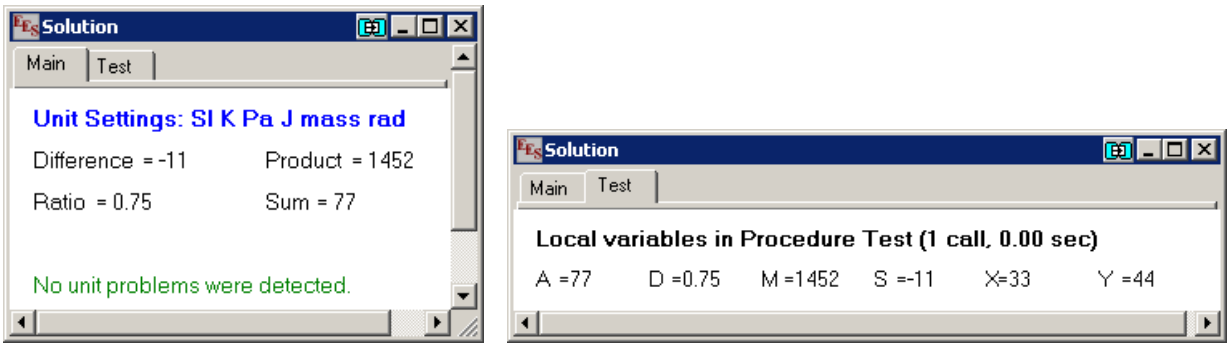
We will illustrate the use of a procedure with a simple example that finds the product, ratio, sum and difference (M, D, A, S) of two values (X and Y):

```
Procedure Test(X,Y: M,D,A,S)
    M:=X*Y                                  "multiply"
    D:=X/Y                                  "divide"
    A:=X+Y                                  "add"
    S:=X-Y                                  "subtract"
end
```

A valid Call statement for Procedure Test shown above is:

```
Call Test(33,44: Product, Ratio, Sum, Difference)        "Call to the Test Procedure"
```

When this code is executed, EES will assign X and Y in Procedure Test to be 33 and 44, respectively. It will evaluate each of the four outputs and assign them to the four values in the output list: Product, Ratio, Sum, and Difference. A tab in the Solutions Window shows the variables in the Main Equations Window and the workspace for the procedure Test, as shown in Figure 3-6.



(a) **(b)**

Figure 3-6: Tabs in the Solution Window for (a) Main Equation Window and (b) the workspace for the procedure Test.

Although a procedure is normally written to accept inputs and return outputs, it is also possible to call the procedure from the Equations Window, subprogram, or module by supplying it with some of the output values in order to have it calculate one or more of the input values. This is possible because EES interprets the procedure as a set of equations relating each output to the inputs. Therefore, it will attempt to solve the resulting set of equations iteratively. For example, the procedure Test could be called in the following manner:

```
Call Test(X, Y: Product, Ratio, 88, 32)          "Alternative call to the Test Procedure"
```

In this case, the values of the last two outputs are specified. These outputs are the sum and difference of the inputs, X and Y. EES will attempt to determine the values of the inputs X and Y that provide the specified outputs, using iteration if necessary. After solving, the Solution window will appear as shown in Figure 3-7.

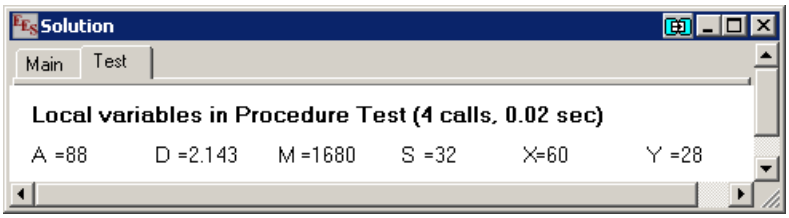


Figure 3-7: Solution window showing local results for Procedure Test.

When a procedure is called from within a function or another procedure, it is interpreted as a set of assignment statements, one for each output. In this case, it is not possible to provide values any of the outputs in order to determine one or more of the inputs.

Like EES functions, procedures can be saved separately and used in other EES programs. The most convenient way to do this is to save the procedure as a library file, as detailed in Chapter 11. EES supports both internal and external procedures. Internal procedures are entered directly at the top of the Equations Window, as described in this section. External procedures are written in a high-level language such as C, Pascal, or FORTRAN and called from EES. The Call statement for both types of procedures is identical. Chapter 19 provides instructions for writing and using external functions and procedures.

3.4 Logic Statements

A major advantage of the assignment statements that are used in EES functions and procedures is that they can include logic statements in order to control the order of the execution of the assignment statements. EES functions and procedures recognize several types of logic statements, as described in this section. These logic statements can not be used in modules, subprograms or in the main body of an EES program.

If-Then-Else Statements

The most common logic statement is the If-Then-Else statement. EES provides two formats for these types of statements, which are referred to as the single-line and the multiple-line formats. The single-line format has the following form:

```
If (Conditional Test) Then Statement 1 Else Statement 2
```

The Then keyword and Statement 1 are required. Statement 1 can be either an assignment statement or a GoTo statement, as described below. The Else keyword and Statement 2 are optional. A single-line If-Then-Else statement must be placed on one line with no line breaks; however, there is no limit on the number of characters that can be used in this line.

The multiple-line format has the following form:

```
If (Conditional Test) Then
    Statement(s)
Else
    Statement(s)
EndIf
```

The major difference between these two formats is that the multiple-line format allows one or more assignment statements or logic statements to be executed depending on the result of the Conditional Test, whereas the single-line format allows only one statement to be executed. The Else and EndIf keywords appearing in the multiple-line format must each appear on a line by itself, as shown above. Indentation can be used to make the logic flow more clear. Multiple-line If-Then-Else statements can include additional If-Then-Else statements, which provides nested conditional evaluation.

The Conditional Test yields either a true or false result using relational operators. The relational operators recognized by EES are summarized in Table 3-1.

Table 3-1: Relational operators recognized in conditional tests.

Relational Operator	Description
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal to
and	logical And between two relational tests
or	logical Or between two relational tests

The parentheses around the conditional test are usually optional, but they are recommended for reading clarity. However, there are situations in which the parentheses are required in order to obtain the intended result. EES processes the logical operations from left to right unless parentheses are provided in order to change the parsing order. Note that the parentheses around the (x>0) and (y<>3) are required in the following single-line If-Then-Else statement in order to override the default left to right logical processing order:

```
If (x>y) or ((x<0) and (y<>3)) Then z:=x/y Else z:=x
```

All variables (except string variables) used in EES are represented with real floating-point 10-byte numbers that provide 20 significant figures of numerical precision. EES does not provide integer data types. For this reason, care should be taken when using the = relational operator, as two values may be quite close to one another, but not exactly equal. The <= or >= operators should usually be used instead of the = operator. Alternatively the Round and Trunc functions can be used to ensure that a value is exactly equal to an integer. String variables should be compared with the = operator.

Return Statements

The Return statement can only be used within functions and procedures. When EES encounters a Return statement, it will exit the function or procedure and control will resume at the point where the function or procedure was called. The Return statement is used in logic constructions with the If-Then-Else or Repeat-Until statements.

The following example develops a function named Nusselt which uses logic to provide the appropriate value of the non-dimensional Nusselt number for fully-developed flow in a circular pipe subjected to a constant heat flux given the Reynolds number (*Re*), Prandtl number (*Pr*), and the relative roughness (*RR*). The function uses correlations presented in Nellis and Klein (2009)[1]. For laminar flow (*Re* < 2300), the Nusselt number is a constant:

$$Nu = 4.36 \tag{3-3}$$

The function begins by assigning Nusselt a value consistent with laminar flow and the using the Return statement to return to the main program if *Re* < 2300.

[1] Note that the Heat Transfer library, discussed in Chapter 12, provides a much more powerful version of this function as well as other functions and procedures based on the Nellis and Klein Heat Transfer textbook.

```
Function Nusselt(Re, Pr, RR)
    "assume flow is laminar"
    Nusselt = 4.36 [-]                                  "laminar flow Nusselt number"
    if (Re<2300) then Return                            "return if flow is laminar"
```

For turbulent flow (Re > 2300), the friction factor is computed according to:

$$f = \left\{ -2.0 \log_{10} \left[\frac{2\,RR}{7.54} - \frac{5.02}{Re} \log_{10} \left(\frac{2\,RR}{7.54} + \frac{13}{Re} \right) \right] \right\}^{-2} \tag{3-4}$$

and used to compute the Nusselt number according to the Gnielinski correlation:

$$Nu = \frac{\left(\frac{f}{8} \right) (Re - 1000)\, Pr}{1 + 12.7 \left(Pr^{2/3} - 1 \right) \sqrt{\frac{f}{8}}} \tag{3-5}$$

Note that the Gnielinski correlation is only valid for $2300 < Re < 5 \times 10^6$. The function continues by using Eqs. (3-4) and (3-5) to assign Nusselt a value that is consistent with turbulent flow. The Return statement is used to return to the calling program if the Reynolds number is less than 5×10^6.

```
    "assume Reynolds is turbulent"
    f = (-2*log10(2*RR/7.54-5.02*log10(2*RR/7.54+13/Re)/Re))^(-2)   "friction factor"
    Nusselt = f/8*(Re-1000)*Pr/(1+12.7*(Pr^(2/3)-1)*sqrt(f/8))      "Gnielinski correlation"
    if (Re<=5e6) then Return                                        "return if Re is not out of range"
```

Finally, the function returns a value of -9 when the Reynolds number provided is greater than 5×10^6 in order to indicate that the correlation is not valid.

```
   "Reynolds number must be out of range"
   Nusselt = -9 [-]                                     "value if Re is out of range"
end
```

A much better way to handle this situation is to use the Warning or Error Procedures, which are described in Section 3.7. The function is tested at various Reynolds numbers:

```
"test the function"
Nu#_laminar=Nusselt(1000,0.7,0.001)                     "test for laminar flow"
Nu#_turbulent=Nusselt(5000,0.7,0.001)                   "test for turbulent flow"
Nu#_outofRange=Nusselt(6E6,0.7,0.001)                   "test for out of range"
```

which leads to the Solution Window shown in Figure 3-8.

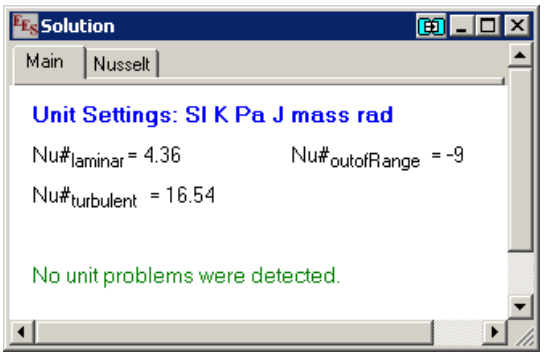


Figure 3-8: Solutions Window.

GoTo Statements and Statement Labels

EES will normally process the assignment statements in a function or procedure in the order that they appear, starting with the first statement. However, the flow control can be altered using GoTo statements. The format of a GoTo statement is simply:

```
GoTo  Line#
```

where Line# is a statement label that must be an integer number between 1 and 30000. Statement labels precede an assignment statement and are separated from the statement by a colon (:). A statement label may also appear on a line by itself. The GoTo statement must be used with If-Then-Else statements to be useful.

The following function illustrates the use of GoTo and If-Then-Else statements in the calculation of the factorial of a value supplied as the argument.

```
Function Fact(N)
        F:=1                                    "factorial"
        i:=0                                    "counter"
10:     i:=i+1                                  "increment counter"
        F:=F*i                                  "multiply factorial by counter"
        If (i<N) Then GoTo 10                   "see if counter has reached N"
        Fact:=F                                 "assign factorial"
End
```

The Fact function is tested and compared to the value returned by the built-in Factorial function in EES.

```
Y= Fact(7)                                      "Fact(7)=5040"
Y2=Factorial(7)                                 "EES built-in factorial function"
```

Solving leads to Y = 5040 and Y2 = 5040.

Repeat-Until Construct

Looping within functions and procedures can be implemented with If-Then-Else and GoTo statements described above. However, it is generally more convenient and readable to use a Repeat-Until construct. The Repeat-Until construct has the following format.

```
Repeat
    Statement(s)
    ...
Until (Conditional Test)
```

The Conditional Test is the same type of conditional test used in If-Then-Else statements. The Fact function written above using a GoTo statement can be implemented more simply using a Repeat-Until construct, as shown in the following example:

```
Function Fact(N)
    Fact:=1                          "factorial"
    If (N=0) then Return             "check for N=0"
    Repeat
        Fact:=Fact*N                 "start with argument"
        N:=N-1;                      "decrement argument by one"
    Until (N<=1)
End
```

Note the use of the Return statement to terminate the function in the event that the argument is zero. The Repeat-Until construct can only be used in functions and procedures.

3.5 Units in Functions and Procedures

The units of variables that appear in EES functions and procedures can be set from the Variable Information dialog or the Solution Window, as discussed in Section 3.2 in the context of the example function PR, which implements the Peng-Robinson equation of state.

```
Function PR(v,T)
    R = 188.9 [J/kg-K]
    a = 70.89 [N-m^4/kg^2]
    b = 0.0006059 [m^3/kg]
    PR = R*T/(v-b)-a/(v*(v+b)+b*(v-b))
End
```

In some cases, you may not know the units of the variables in a function or procedure until it is executed. The PR function above, for example, is written with the expectation that it will be called with the EES unit system configured for standard SI units of m, K, J, and Pa. It would be nice to be able to have this function work properly when called from an EES program that is using any set of units. This is particularly important for those functions or procedures that are placed in libraries. This capability can be implemented using the UnitSystem$ function.

The UnitSystem$ function has the calling protocol shown below:

```
Unit$ = UnitSystem$(Dimension$)
```

where Dimension$ is a string containing the dimension of interest. The UnitSystem$ function returns the string Unit$ containing the expected units of the dimension for the unit system settings that EES is configured to work in. The unit system settings can be specified as discussed in Section 1.5 using either the $UnitSystem directive in the Main body of the EES program or the Unit System tab of the Preferences dialog.

Let's rewrite function PR so that it operates correctly regardless of the unit system that EES is configured to use. First, the UnitSystem$ function is used to determine the units that are associated with specific volume, temperature, and pressure.

```
Function PR(v,T)
    "Obtain the unit settings in EES"
    v$=UnitSystem$('Volume')                    "V$ is the specific volume units that EES is set to"
    T$=UnitSystem$('Temperature')               "T$ is the temperature unit that EES is set to"
    P$=UnitSystem$('Pressure')                  "P$ is the pressure units that EES is set to"
```

Next, the values of the specific volume and temperature are converted from the unit system that EES is currently configured to work in (i.e., the units of the inputs v and T) to the base SI units required to implement the Peng-Robinson equation of state (v_SI and T_SI). This conversion is accomplished using the convert and converttemp functions using the strings v$ and T$.

```
    "Convert inputs to base SI units"
    v_SI=v*convert(v$,m^3/kg)                   "convert v to base SI units"
    T_SI=converttemp(T$,K,T)                    "convert T to base SI units"
```

The calculations are carried out in base SI units, as before, in order to obtain the pressure predicted by the Peng-Robinson equation of state in base SI units (PR_SI).

```
    "Carry out calculations"
    R = 188.9 [J/kg-K]
    a = 70.89 [N-m^4/kg^2]
    b = 0.0006059 [m^3/kg]
    PR_SI = R*T_SI/(v_SI-b)-a/(v_SI*(v_SI+b)+b*(v_SI-b))    "pressure, in Pa"
```

Finally, the pressure is converted to the unit system expected by EES based on the current unit system settings. This is accomplished using the convert function and the string P$:

```
    "Convert output to EES units"
    PR = PR_SI*convert(Pa,P$)                   "pressure converted to units that EES is set to"
End
```

The function can now be used with any EES unit settings. For example, in the code below the units are set to the English unit system.

```
$UnitSystem English Mass Btu F psi
v=0.1 [ft^3/lbm]                                "specific volume"
T=300 [F]                                       "temperature"
```

P=PR(v,T) "pressure"

The units of the variables in the PR function workspace should be set in the appropriate page of the Variable Information Window, as shown in Figure 3-9.

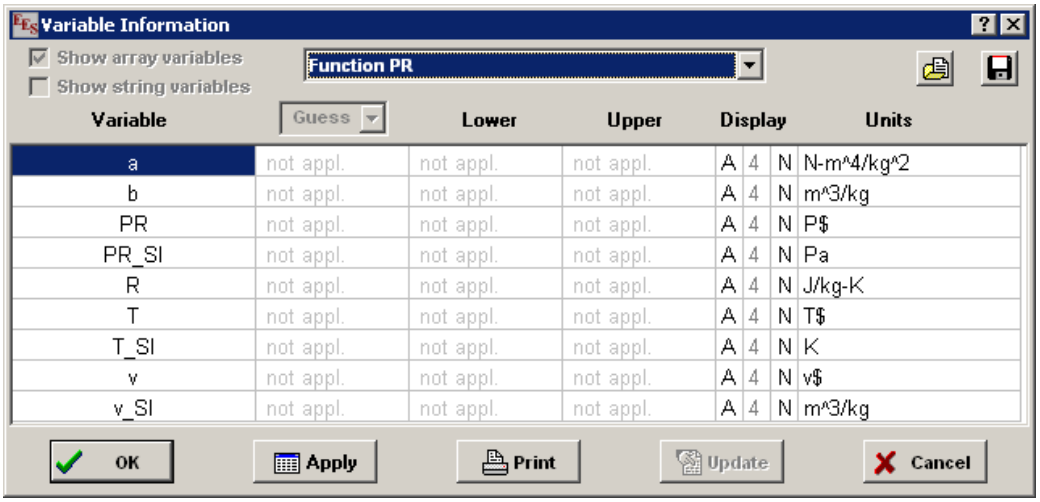


Figure 3-9: PR Function workspace page of the Variable Information Window.

Note that the units for variables such as a, b, and T_SI can all be set in a straightforward way because they are unambiguous; these are variables that will be in base SI units regardless of the EES unit settings. However, there is no way in general to know the units of variables such as v, T, and PR as these variables will have units that are consistent with the unit settings in EES. Therefore, the units for these variables are set using the strings v$, T$, and P$, respectively, so that they will automatically adjust based on the current EES unit settings.

3.6 Arrays in Functions and Procedures

Array variables in EES have an integer array index that is enclosed in square brackets at the end of the variable name, as discussed in Section 1.7. Array variables are in most ways similar to any other variable. However, they provide some additional processing capabilities because operations involving the array index can be programmed using Duplicate loops in the Equations Window and Repeat-Until constructs in functions and procedures. Also, the array values can be displayed in the Arrays Table Window.

Arrays as Arguments

Shown below is a short function that is designed to calculate the sum of the squares of all of the elements in the array A[] that is provided to it as the second argument. The number of elements in the array is provided as the first argument to the function. Note that array range notation, i.e., [1..N], is used to indicate all of the array elements between index 1 and index N. The function SumSquares uses a Repeat-Until construct to loop through all of the elements and sum their squares.

```
Function SumSquares(N, A[1..N])
    S:=0                                  "initialize sum"
    i:=1                                  "initialize index"
    Repeat
        S:=S+A[i]^2                       "sum the square of each element"
        i:=i+1                            "increment the element"
    Until (i>=N)                          "stopping condition"
    SumSquares:=S                         "set the Function name to the calculated value"
end
```

We can test this function with the following code:

```
N=5                                       "number of array elements"
duplicate i=1,N
    X[i]=i                                "set the value of the array elements"
end
SS=SumSquares(N,X[1..N])                  "call the sum of squares function"
```

which should return a value of SS = 30 in this case.

There are, however, several potential problems with this code. For example, if you change N from 5 to 150, you will see the error message shown in Figure 3-10.

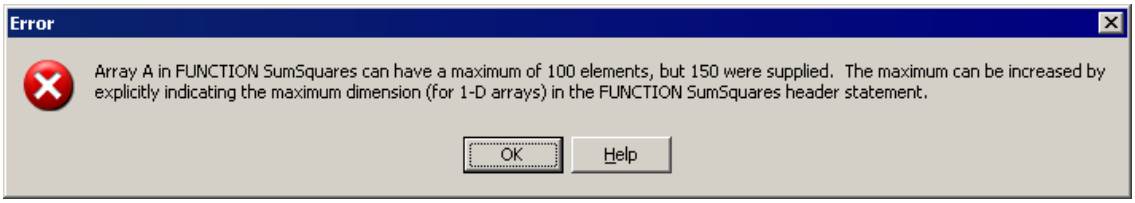


Figure 3-10: Error message that appears when you try to call the function SumSquares with N = 150.

Why did EES solve this program when N = 5 but not when N = 150? The explanation for this problem requires an understanding of how EES handles the equations that are entered in the Equations Window. All equations in EES are compiled; that is, they are translated into an internal form that minimizes the computational effort required to evaluate them. EES starts at the top of the Equations Window and compiles the equations one by one. The Equations Window starts with the function declaration for SumSquares. However, EES cannot possibly know the value of N that will be provided to the function as an argument. Without knowing the value of N, EES cannot compile the equations that define the function.

When EES does not know the value of an array index limit, it will assume that it is 100 by default. The A[1..N] appearing in the function declaration for SumSquares is therefore interpreted to be A[1..100] when the function is compiled. Any value of N that is less than or equal to 100 will work properly when this function is called, but the function fails when N is greater than 100.

There are several ways to solve this problem. First, we could change the upper limit of the array index to be 200 in the function declaration.

```
Function SumSquares(N, A[1..200])
```

Now, the program will solve when N is set to 150 or any value that is less than or equal to 200 because EES is reserving 200 elements for array A in the function. Note that the upper array index does not need to equal the size of the array provided to the function in the calling program (although that is desirable). It only has to be greater than the size of the array provided by the calling program. The maximum allowable array size that can be passed to a function in this manner is 2000 in the current implementation of EES.

Using the $Common Directive to Access Arrays

We could also access array variables from the Main program within the function using the $Common directive placed within the function. As explained in Section 14.3, the $Common directive in the function or procedure provides read-only access to the variables (i.e., the variables cannot be modified from within the function). This method is more efficient than passing the array to the function because it avoids some of the overhead associated with passing the variables from the Main program to the function. However, we would need to correctly specify the name and size of array variables in the $Common directive in order for this method to work, as shown in the following example where N = 150 and the name of the array in the function has been changed to match its name in the Main program.

```
Function SumSquares(N)
    $Common X[1..150]                    "provides read-only access to the array X[]"
    S:=0                                 "initialize sum"
    i:=1                                 "initialize index"
    Repeat
        S:=S+X[i]^2                      "sum the square of each element"
        i:=i+1                           "increment the element"
    Until (i>=N)                         "stopping condition"
    SumSquares:=S                        "set the Function name to the calculated value"
end

N=150                                    "number of array elements"
duplicate i=1,N
    X[i]=i                               "set the value of the array elements"
end
SS=SumSquares(N)                         "call the sum of squares function"
```

This method of passing information to a function is also not convenient because the value of N is not adjustable. If the upper limit is set to 200 in the $Common directive, an error message will be issued. Similarly if N is changed to 50 in the calling program then an error message will be issued. The above example requires N to be changed in both the Main program and the function if its value is changed.

Using the $Constant Directive to Set Array Limits

Probably the best alternative is to use a global constant to set the array limit. Global constants can be defined using the $Constant directive, as discussed in Section 1.9. Constants must have # as the last character in their name. In the example below, the size of the array is specified by the global constant N# and used to set the upper limit in the array passed to the function. Because the value of the constant is known at the time that the equations are compiled, the size of the array can be correctly allocated. Only the value of N# in the program needs to be changed to accommodate different array sizes.

```
$Constant N# = 150                          "size of array"

Function SumSquares(A[1..N#])
    S:=0                                    "initialize sum"
    i:=1                                    "initialize index"
    Repeat
        S:=S+A[i]^2                         "sum the square of each element"
        i:=i+1                              "increment the element"
    Until (i>=N#)                           "stopping condition"
    SumSquares:=S                           "set the function name to the calculated value"
end

duplicate i=1,N#
    X[i]=i                                  "set the value of the array elements"
end
SS=SumSquares(X[1..N#])                     "Call the sum of squares function"
```

Arrays Table Window for Functions and Procedures

Passing arrays as arguments to or from a function or procedure is inefficient because EES stores the array elements as separate variables. Often, the only reason an array is passed out of a function or procedure is so that it can be viewed or plotted. It is possible to display the arrays that are used in a function or procedure in its own tabbed Arrays Table Window by placing a $Arrays On directive anywhere within the function or procedure.

The following example computes the temperature of an object that is initially at T_{ini} and is subjected to convection and radiation from an environment at T_∞. The surface area of the object is A_s and the total heat capacity is C. The convective heat transfer coefficient is $\overline{h}$ and the object is assumed to have an emissivity of $\varepsilon = 1$. Assuming that the object is at a uniform temperature, the time rate of change of its temperature is given by:

$$\frac{dT}{dt} = \frac{\left[A_s \overline{h}\left(T_\infty - T\right) + A_s \sigma\left(T_\infty^4 - T^4\right)\right]}{C} \tag{3-6}$$

where σ is the Stefan-Boltzmann constant. The Integral command, discussed in Chapter 7, provides a powerful technique for integrating differential equations numerically in EES. However, here we will write a function named Temp that uses a simple Euler numerical integration technique to determine the temperature of the object after a specified duration of time. The arguments to the function are the time duration to simulate (duration), heat transfer

coefficient (h_bar), surface area (A_s), initial temperature (T_ini), ambient temperature (T_infinity), heat capacity (C), and the total number of time steps (N).

```
Function Temp(duration, h_bar, A_s, T_ini, T_infinity, C, N)
```

The duration of each time step is computed according to:

$$\Delta t = \frac{duration}{N} \tag{3-7}$$

```
    Dtime:= duration/N                    "Time step"
```

The temperature at the initial time (T_1) is the initial temperature and the initial time ($time_1$) is zero. The index of the time step (i) is initially set to 1.

```
    T[1]:= T_ini                          "initial temperature"
    time[1]:=0
    i:=1
```

Each time step is taken using a Repeat-Until construct. Within the construct, the time at index i+1 is computed:

$$time_{i+1} = time_i + \Delta t \tag{3-8}$$

The use of the Euler technique to integrate Eq. (3-6) leads to the temperature at index i+1:

$$T_{i+1} = T_i + \frac{\left[A_s \bar{h} \left(T_\infty - T_i \right) + A_s \sigma \left(T_\infty^4 - T_i^4 \right) \right]}{C} \Delta t \tag{3-9}$$

Finally, the index is incremented. The loop is terminated when $i > N$.

```
    Repeat
        time[i+1]=time[i]+Dtime
        T[i+1]=T[i]+(A_s*h_bar*(T_infinity-T[i])+A_s*sigma#*(T_infinity^4-T[i]^4))*Dtime/C
        i:=i+1
    Until(i>N)
```

The output of the function is assigned to be the temperature at index N+1.

```
    Temp=T[N+1]
End
```

The function Temp is tested with arbitrary inputs:

```
N=5 [-]
duration=100 [s]                                        "duration of process"
h_bar=100 [W/m^2-K]                                     "heat transfer coefficient"
A_s=0.01 [m^2]                                          "surface area"
T_ini=300 [K]                                           "initial temperature"
T_infinity=500 [K]                                      "ambient temperature"
C=100 [J/K]                                             "total heat capacity"
T_final=Temp(duration, h_bar, A_s, T_ini, T_infinity, C, N)   "final temperature"
```

which leads to the result T_final = 449 K. Note that the units for the variables used in the function Temp can be set most easily be selecting the option Function TEMP from the drop down menu in the Variable Information dialog and then de-selecting the Show array variables option. In this way, all of the units in the arrays T[] and time[] can be specified at the same time, as shown in Figure 3-11.

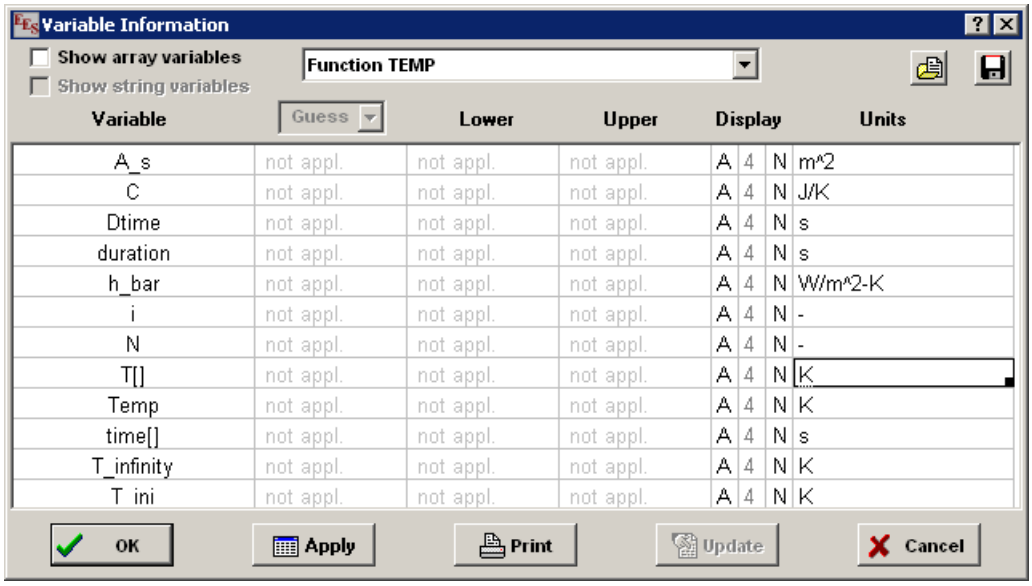


Figure 3-11: The Function TEMP page of the Variable Information dialog.

The arrays T[] and time[] are not passed from the function workspace back to the EES program. The array variables from the last call to the function are available by examining the Temp tab of the Solutions Window, as shown in Figure 3-12.

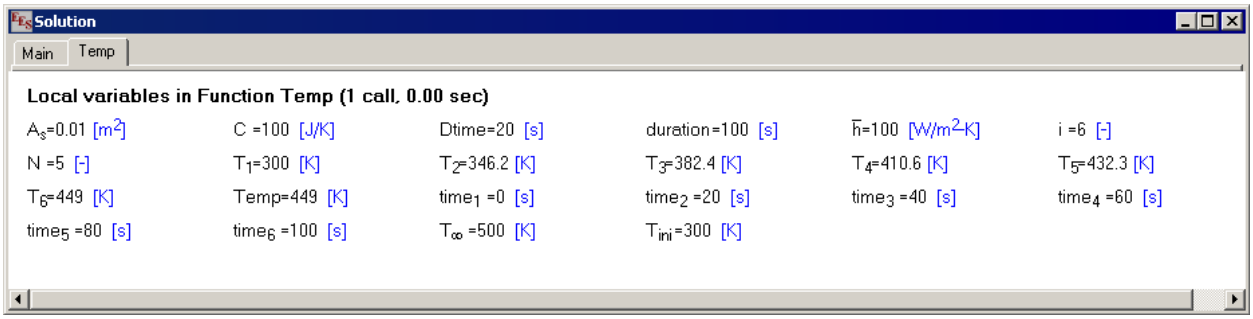
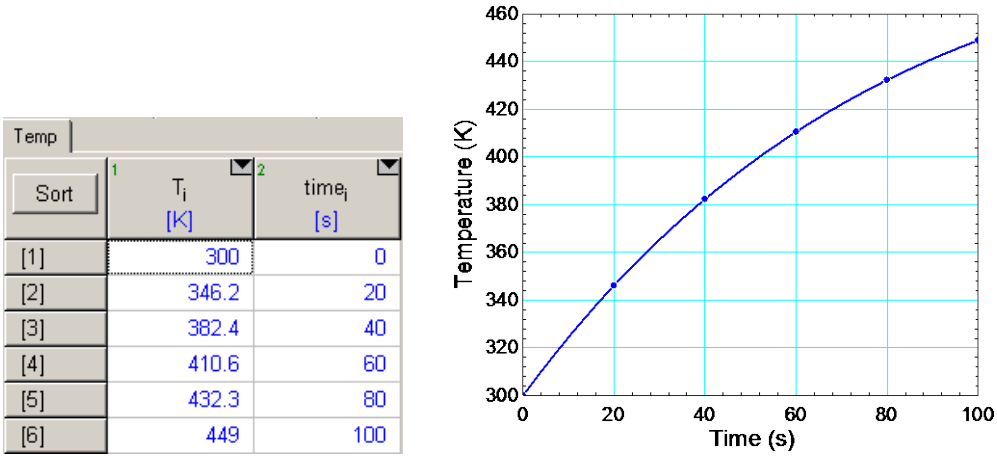


Figure 3-12: Temp tab of the Solutions Window.

There may be times when it is useful to plot the results contained in the arrays that are used within a function or procedure. The array variables associated with the last call to function Temp can be placed in a separate Arrays Table by placing the $Arrays On directive at the top of the function.

```
Function Temp(duration, h_bar, A_s, T_ini, T_infinity, C, N)

    $Arrays On
    Dtime:= duration/N                    "Time step"
    T[1]:= T_ini                          "initial temperature"
    time[1]:=0
    i:=1
    Repeat
        time[i+1]=time[i]+Dtime
        T[i+1]=T[i]+(A_s*h_bar*(T_infinity-T[i])+A_s*sigma#*(T_infinity^4-T[i]^4))*Dtime/C
        i:=i+1
    Until(i>N)
    Temp=T[N+1]
End
```

The Arrays Window for the Temp function is shown in Figure 3-13(a). The data stored in the Arrays Window can be plotted, as shown in Figure 3-13(b). Note that the $Arrays Off directive can be used to restore the array variables to the Solution Window.



(a) **(b)**

Figure 3-13: (a) Arrays Window for the Temp function and (b) temperature as a function of time.

3.7 The Warning and Error Procedures

The Warning and Error procedures provide a means of displaying warning or error messages. The Warning and Error procedures can only be called from within an internal function or procedure.

The Warning Procedure

The Warning procedure generates a warning message that is placed in the warning message queue. The warning message queue is displayed after calculations are completed provided that Warnings are enabled. Warnings can be enabled by clicking in the Warnings box of the status bar at the bottom of the Equations Window, as shown in Figure 3-14.

Figure 3-14: Warnings box of the Equations Window status bar.

Warnings can also be enabled by placing a $Warnings On directive in the Equations Window or by selecting the Display Warning Messages checkbox in the Options tab of the Preferences dialog.

The Warning procedure can be called with a warning string provided according to:

```
Call Warning ('My warning string')
```

where the string would contain a descriptive message about the problem. In this case, EES will generate the warning message contained in the string. The Warning procedure can also be called with both a warning string and a numerical variable, according to:

```
Call Warning ('My warning string, XXXF1', Y)
```

where Y is a number or a numerical variable. In this case, the characters XXX in the string will be replaced with the value of the parameter Y. If a formatting option (e.g., F1 for fixed decimal with one significant figure in the call above) follows the characters XXX then the display of the variable Y will be formatted accordingly. If no format is supplied, automatic format is assumed. The Warning procedure can be called with a warning string and a string variable, according to:

```
Call Warning ('My warning string, XXX$', Y$)
```

where Y$ is a string variable. In this case, the string contained in Y$ will replace the characters XXX$ when the warning is issued. Finally, the Warning procedure can be called with only a numerical variable:

```
Call Warning (Y)
```

In this case, EES will generate the following generic warning message: "A warning message was issued due to the value XXX in ZZZ." Where the characters XXX will be replaced with the value of Y and the characters ZZZ will be replaced by the name of the function or procedure in which the Call Warning statement appears.

As an example, a call to the Warning procedure has been added to the Nusselt function that was presented in Section 3.4. The Warning procedure is called if the Reynolds number provided by the user is outside of the range of the correlation.

```
Function Nusselt(Re, Pr, RR)
    "assume flow is laminar"
    Nusselt = 4.36 [-]                                          "laminar flow Nusselt number"
    if (Re<2300) then Return                                    "return if flow is laminar"

    "assume Reynolds is turbulent"
    f = (-2*log10(2*RR/7.54-5.02*log10(2*RR/7.54+13/Re)/Re))^(-2)  "friction factor"
    Nusselt = f/8*(Re-1000)*Pr/(1+12.7*(Pr^(2/3)-1)*sqrt(f/8))  "Gnielinski correlation"
    if (Re<=5e6) then Return                                    "return if Re is not out of range"

    "Reynolds number must be out of range"
    Call Warning ('Reynolds number must be <= 5e6 in function Nusselt. &
          A value of XXXE2 was provided', Re)
    Nusselt = -9 [-]                                            "value if Re is out of range"
end
```

Calling the function with the following arguments:

```
Nu#_outofRange=Nusselt(6E6,0.7,0.001)          "test for turbulent flow"
```

will display the warning shown in Figure 3-15 (provided that warnings are enabled).

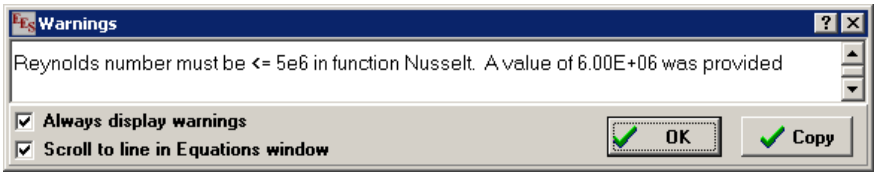


Figure 3-15: Warning message displayed when function Nusselt is called with Re = 6e6.

The calculations will complete and the Solutions Window will indicate that the value of the variable Nu#_outofRange has been set and that 1 warning was issued, as shown in Figure 3-16.

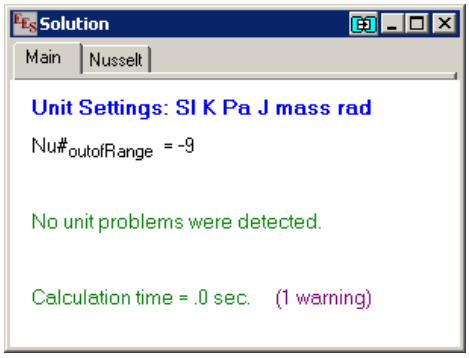


Figure 3-16: Solutions Window.

The Error Procedure

The Error procedure has the same calling format as the Warning procedure. The difference is that the error message will be displayed regardless of whether warnings are enabled or not, and the calculations will be halted immediately after the message is displayed. If Call Warning is replaced with CALL Error in the above example,

```
Call Error ('Reynolds number must be <= 5e6 in Function Nusselt.  &
      A value of XXXE2 was provided', Re)
```

then the error message shown in Figure 3-17 will be displayed when the solve command is issued.

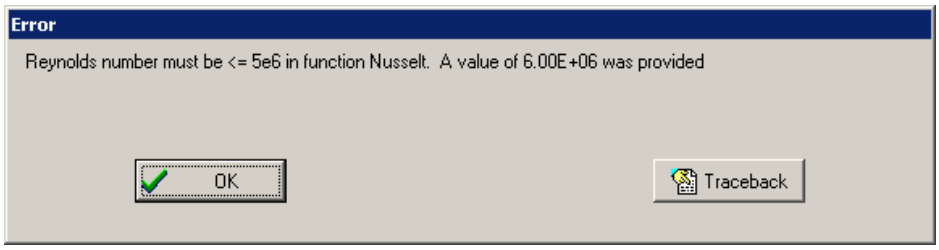


Figure 3-17: Error message displayed when calling Function Nusselt with Re = 6e6.

Clicking the Traceback button will show the line that initiated the error message provided that the code is contained in the Equations Window rather than in a library file stored on disk

References

Churchill, S.W. 1977, 'Friction Factor Spans All Fluid Flow Regimes,' Chem. Eng. (Rugby, U.K.) 84(24), pp.91-91.

Klein, S.A. and Nellis, G.F., *Thermodynamics*, Cambridge University Press, New York, (2012).

Nellis, G.F., and Klein, S.A, *Heat Transfer*, Cambridge University Press, New York, (2009).

4 PROPERTY DATA

Historically, thermodynamic and transport property data for fluids were provided in tabular and graphical forms. You do not have to solve many engineering problems before the limitations associated with the use of tabular or graphical property information become evident. Looking up property values in tables usually requires either single or double interpolation. The process is time-consuming and likely to introduce mathematical errors. Graphical property data do not require interpolation, but using these graphs is tedious and the accuracy of the data is limited. It is not easy or even practical to carry out the parametric studies that are required for optimization or design using tabular or graphical property information.

EES provides high accuracy thermodynamic and transport thermophysical property data for many substances. These data are accessed with the built-in property functions described in this chapter. These property functions, integrated with the equation solving and plotting capabilities, make EES a useful tool for engineering calculations in which property data are required.

4.1 Unit System

It is necessary to specify the unit system that EES will use for the thermodynamic and transport property functions. The unit system controls both the units of the input parameters that EES expects for variables provided to the property functions as well as the units of the properties that are returned by the functions.

Unit System Dialog

The unit system can be specified in two ways, as discussed in Section 1.5. One way is to select Unit System from the Options menu in order to display the Unit System tab of the Preferences dialog, shown in Figure 4-1. Note that the unit system can be specified using SI or English units, but not a combination of both. As shown in Figure 4-1, the units of temperature can be either Celsius or Kelvin for SI units. Energy units can be specified to be in J or kJ. Pressure can be specified to be in units of Pa, kPa, bar, or MPa.

Specific Properties on a Molar vs Mass Basis

Values of specific properties, e.g., specific volume, specific heat capacity, and specific enthalpy, can be specified on either a mass or molar basis. A mole of a substance is defined as the amount of mass that is equal to the molar mass of the substance (*MW*, also referred to as the molecular weight). Therefore, the mass units need to be specified when specifying a mole. For example, if the mass unit is chosen to be kg then the corresponding mole is a kmol (or kgmol). The molar mass of helium is 4 and therefore a kmol of helium has a mass of 4 kg. A pound mole (lbmol) of helium has a mass of 4 lb_m. A gram mole (gmol) of helium has a mass of 4 grams, and so on. Occasionally, the term "mole" is expressed without reference to mass units; in this case, mole usually refers to a gmol. The number of moles (n) and mass (m) of a substance are related by Eqn. (4-1):

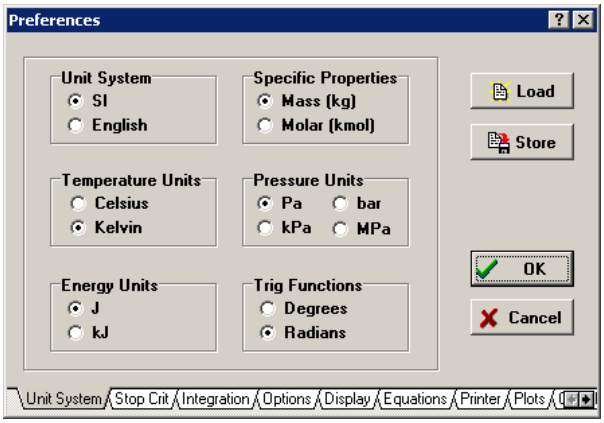


Figure 4-1: Unit System dialog.

$$n = \frac{m}{MW} \tag{4-1}$$

The molar base unit in EES is the kmol in SI units and the lbmol in English units.

The $UnitSystem Directive

An alternative way to specify the unit system is to use the $UnitSystem directive in the Main body of an EES program. This directive has the following format for SI units:

```
$UnitSystem SI  Mass [or Mole]  Deg [or Rad]  Pa [or kPa, bar, MPa]  C [or K]  J [or kJ]
```

Notice that each of the selections required by the user in the Unit System dialog, shown in Figure 4-1, can be made using the $UnitSystem directive. As an example, the following line placed in the Equations Window will set the EES unit system to standard SI units with temperature in K, pressure in Pa, and energy in J. Specific properties are expressed on a mass basis and trigonometric functions will expect and return angles in radians.

```
$UnitSystem SI K Pa J Mass Rad
```

The selections can be made in any order. The $UnitSystem directive in English units has the following format:

```
$UnitSystem Eng  Mass [or Mole]  Deg [or Rad]  psia [or atm]  F [or R]
```

Mixed SI/English unit system specifications are not allowed. Each specification in the $UnitSystem directive is separated with a space. It is not necessary to enter all of the units, although it is good practice to do so. The $UnitSystem directive is normally placed at the top of the Equations Window. If a $UnitSystem directive is used, it will override any settings that are made with the Unit System dialog.

Status Bar

The unit system settings are displayed in the center of the status bar at the bottom of the Equations Window, as shown in Figure 4-2. Clicking on the units section in the status bar will bring up the Unit System dialog window shown in Figure 4-1.

Figure 4-2: Units in the status bar.

Importance of Unit Selection

The choice of unit system will control both the expected units of the parameters that are provided to a property function as well as the units of the property value that are returned by the function. If, for example, the units for temperature are selected to be Kelvin (as shown in Figure 4-1 and Figure 4-2) then any temperature provided to an EES property function is assumed to be in Kelvin units, regardless of what units may have been assigned to the variable itself. If the units of the input parameter do not match the specified unit system, then EES will display a unit warning message when the Check Units command in the Calculate menu is selected or after the calculations are completed (assuming that the Check units automatically option in the Preferences tab of the Options dialog is selected).

4.2 Function Information

EES provides property information for many substances. The calling format and the number of input parameters for the property functions depend on the nature of the substance and the type of property. Some property functions (e.g., ammonia-water mixtures) are provided by external programs (see Chapter 19) and these are called with a different format than is used for built-in property functions. The format of all of the property functions can be examined by selecting Function Information from the Options menu in order to access the Function Information dialog shown in Figure 4-3.

There are eight radio buttons at the top of the Function Information dialog. The internal property data functions that are the subject of this chapter are accessed by selecting either the Fluid properties radio button (as shown in Figure 4-3) or the Solid/liquid properties button. External property functions may be accessed by selecting either the EES library routines or External routines buttons. Information on each of these alternatives is provided in the following sections.

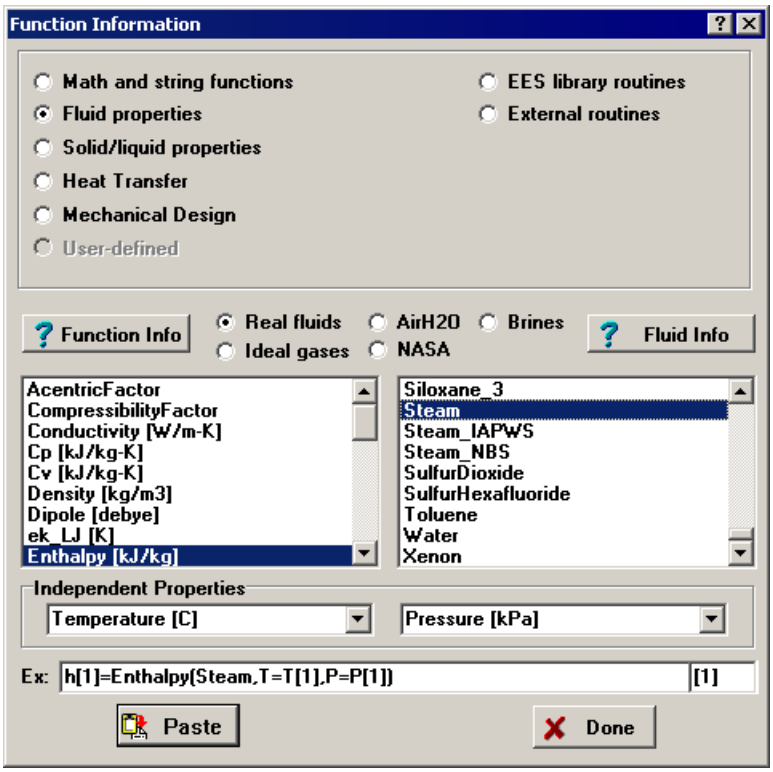


Figure 4-3: Function Information for Fluid properties.

4.3 Property Functions for Real Fluids

In EES, the term real fluids refers to fluids that are represented with compressible fluid models that describe the substance liquid, two-phase or superheated states. There are many thermodynamic and transport properties, but they are all related by the phase rule. The phase rule dictates that the state of a pure compressible fluid in a single-phase state is fixed by specifying the values of two mass-independent (i.e., intensive or specific) properties.

Calling Protocol for Property Function

The calling protocol for the typical property function is given below:

```
X = FunctionName(Fluid$, Property1 = Value1, Property2 = Value2)
```

FunctionName is the name of the property function being called; the name of the function corresponds to the property that is returned. For example, the Enthalpy function returns the specific enthalpy, the Volume function returns specific volume, etc. The first argument (Fluid$) is a string that specifies the name of the fluid being considered. In most cases, two properties are required to fix the state; the second and third arguments indicate which two properties are specified and their values. Property1 is an indicator that identifies the first property that is specified and Value1 is the value of that property (in the units specified by the unit system, as discussed in Section 4.1). If Property 1 is T, then the temperature is specified, H indicates that the specific enthalpy is specified. (See Table 4-3 for a list of the indicators that can be used with real

fluids.) Property2 and Value2 are the indicator and value of the second property that is specified to fix the state. The function will return the value of the property in the units specified by the unit system.

As a simple example, the code below determines the specific volume of the refrigerant R134a at 350 K and 250,000 Pa.

```
$UnitSystem SI Mass J K Pa Rad
v=Volume(R134a, T=350 [K], P=250000 [Pa])     "specific volume"
```

Solving provides v = 0.1108 m^3/kg. Note that EES will check to ensure that the units of the input temperature is K and the units of the input pressure is Pa, consistent with the $UnitSystem directive. Further, EES will check that the units of variable v are set to m^3/kg. If any of these conditions are not met then a warning will be issued when the Check Units command is selected from the Calculate menu. A warning will also be issued when the calculations are completed if the Check Units Automatically option is selected in the Options tab of the Preferences dialog (Options menu).

List of Property Functions

A list of all of the thermodynamic and transport property functions in EES that are applicable for real fluids is provided in Table 4-1. The possible units of the value that is returned are shown in the SI and English unit systems. The units of specific properties in Table 4-1 are shown on a mass basis. EES can also be configured to return properties on a molar basis, as discussed in Section 4.1. Some of these property functions (e.g., acentric factor and fugacity) may be unfamiliar to you. The Thermodynamics textbook by Klein and Nellis (2012) provides information about these properties.

Table 4-1: Thermodynamic and transport property functions for real fluids and their units.

EES Function Name	Returns	SI Units	English Units
AcentricFactor[1]	acentric factor	none	none
CompressibilityFactor	compressibility factor	none	none
Conductivity	thermal conductivity	W/m-K	Btu/hr-ft-R
Cp	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Cv	constant volume specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Density	density	kg/m^3	lb_m/ft^3
Dipole[1]	dipole moment	Debye	Debye
ek_LJ[1]	Lennard-Jones energy potential	K	R
Enthalpy	specific enthalpy	J/kg, kJ/kg	Btu/lb_m
Enthalpy_fusion[1]	specific enthalpy of fusion	J/kg, kJ/kg	Btu/lb_m
Entropy	specific entropy	J/kg-K, kJ/kg-K	Btu/lb_m-R
Fugacity	fugacity	Pa, kPa, bar, MPa	psia, atm
IntEnergy	specific internal energy	J/kg, kJ/kg	Btu/lb_m
IsIdealGas	0 for real fluid, 1 for ideal gas	none	none
Pressure	absolute pressure	Pa, kPa, bar, MPa	psia, atm
MolarMass[1]	molecular weight	kg/kmol	lb_m/lbmol
P_crit[1]	critical pressure	Pa, kPa, bar, MPa	psia, atm
P_sat[3]	saturation pressure	Pa, kPa, bar, MPa	psia, atm
Phase$	phase, e.g., 'superheated'	none	none
Prandtl	Prandtl number	none	none
Quality	quality	none	none
sigma_LJ[1]	Lennard-Jones length potential	m	ft
SoundSpeed	speed of sound	m/s	ft/s
SpecHeat	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
SurfaceTension[3]	surface tension	N/m	lb_f/ft
T_crit[1]	critical temperature	°C, K	°F, R
T_sat[2]	saturation temperature	°C, K	°F, R
T_triple	triple point temperature	°C, K	°F, R
Temperature	temperature	°C, K	°F, R
v_crit[1]	critical specific volume	m^3/kg	ft^3/lb_m
Viscosity	viscosity	kg/m-s	lb_m/ft-hr
VolExpCoef	coefficient of thermal expansion	1/K	1/R
Volume	specific volume	m^3/kg	ft^3/lb_m

1. The function requires only the name of the fluid.
2. The function requires the name of the fluid and the pressure.
3. The function requires the name of the fluid and the temperature.

List of Real Fluids

All of the property functions require a string with the name of the fluid as the first argument in the function call. The names of the built-in real fluids are provided in Table 4-2; note that fluids are constantly being added to EES and therefore this list continues to grow. A list of the fluids that are implemented in your version of EES can be seen by selecting Function Information from Options menu and then clicking the Fluid Properties button at the upper left of the dialog, as shown in Figure 4-3.

Table 4-2: Names of the built-in real fluids.

Air_ha	n-Octane	R245fa
Acetone	n-Pentane	R290
Ammonia	Neon	R404A [1]
Argon	Nitrogen	R407C [1]
Benzene	NitrousOxide	R410A [1]
CarbonMonoxide	Parahydrogen	R423A [1]
CarbonDioxide	Propane	R500 [1]
Cyclohexane	Propylene	R502 [1]
Deuterium	p-Xylene	R507A [1]
DiMethyEther	R11 [1]	R508B [1]
Ethane	R12 [1]	R600
Ethanol	R13 [1]	R600a
Ethylbenzene	R14 [1]	R717
Fluorine	R22	R718
Helium	R23	R744
HFE7500 [1]	R32	RC318
Hydrogen	R41	R1234yf
HydrogenSulfide	R114 [1]	R1234ze
Ice	R116	Siloxane_1
Isobutane	R123	Siloxane_2
Krypton	R124	Siloxane_3
Methane	R125	SF6
Methanol	R131B	Steam
Oxygen	R134a	Steam_IAPWS [2]
o-Xylene	R141b [1]	Steam_NBS
n-Butane	R142b	SulfurDioxide
n-Decane	R143a	SulfurHexafluoride
n-Dodecane	R143m	Toluene
n-Heptane	R152a	Water
n-Hexane	R218	Xenon
n-Nonane	R227ea	

1. This fluid used the Martin-Hou (1955) equation of state to relate properties.
2. Steam_IAPWS uses the high accuracy properties issued by the Int. Assoc. for the Properties of Water and Steam (IAPWS) which is only available in the Professional version of EES.

List of Indicators

The thermodynamic and transport functions for real fluids typically require that two properties be set in order to fix the state and allow the calculation of the property of interest. Some properties (e.g., the saturation pressure which is returned by the function P_sat, or the saturation temperature that is returned by the function T_sat) require only one input property. The critical properties (returned by the functions T_crit, P_crit, and v_crit) are unique for a specific fluid and therefore require no input properties. Table 4-1 indicates the real fluid property functions that require less than two input parameters to fix the state. EES is flexible with regard to what properties can be used to fix the state and in what order they are provided. Therefore, an indicator is required to specify the properties that are provided. This indicator is a single, case-insensitive letter that is followed by an equal sign. The numerical constant or algebraic expression that provides the value of the specified property follows the equal sign. The property indicators that are recognized in function arguments and their meaning are listed in Table 4-3.

Table 4-3: Property indicators for use with real fluids.

Indicator	Description
H	specific enthalpy
P	pressure
S	specific entropy
T	temperature
U	specific internal energy
V	specific volume
X	quality

Fixing the State

The EES code shown below provides examples of functions that require 0, 1, and 2 arguments in addition to the fluid name. Also, note that the fluid name can be specified using a string variable, e.g., F$. String variables must end with a $. The use of a string variable to specify the fluid name makes it easy to change fluids.

```
$UnitSystem SI K Pa J mass rad

F$='Water'
MW=molarMass(F$)                        "molar mass of water"
P_c=P_crit(F$)                          "critical pressure of water"

T=300 [K]                               "variable T set to 300 K"
ST=surfaceTension(F$,T=T)               "surface tension of water at temperature T"
P_s=P_sat(F$,T=T)                       "vapor pressure of water at temperature T"

P=100 [kPa]*convert(kPa,Pa)             "variable P set to 100 kPa"
v=volume(F$,T=T,P=P)                    "specific volume of water"
h=enthalpy(F$,T=T,P=P)                  "specific enthalpy of water"
k=conductivity(F$,T=T,P=P)              "thermal conductivity of water"
```

After solving, the Solution Window will appear as shown in Figure 4-4.

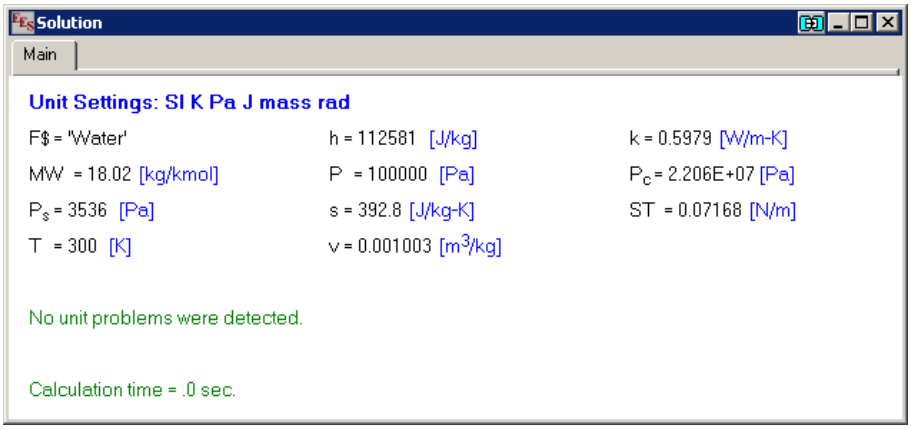


Figure 4-4: Solution window that appears after solving the equations with units specified.

The examples shown above use temperature (T=) and pressure (P=) as the arguments for the two-argument functions. However, the property functions accept any valid combination of properties

in order to fix the state. For example, the specific entropy can be determined from the specific volume and specific enthalpy by adding the following equation to the example.

```
s=entropy(F$,V=v,h=h)                    "specific entropy"
```

which leads to s = 392.8 J/kg-K.

Two-Phase State

Some combinations of properties can not be used to fix the state. For example, the temperature and pressure for a two-phase state are not independent for a pure fluid. Therefore, the temperature and pressure associated with a two-phase state can not be used to determine other properties. Consider the following example, which determines the normal boiling point of ethanol.

```
$UnitSystem SI K Pa J mass rad

F$='Ethanol'                             "fluid"
P=Po#                                    "standard barometric pressure"
T_bp=T_sat(F$,P=P)                       "normal boiling point"
```

Note the use of the built-in constant Po#, which corresponds to the standard barometric pressure in the pressure units specified with the Unit System dialog or $UnitSystem directive. Solving these equations will show that the normal boiling point of ethanol is 351.4 K. Now, add the following equation, which attempts to determine specific volume at the normal boiling point and standard barometric pressure.

```
v=volume(F$,T=T_bp,P=P)                  "attempt to determine specific volume"
```

If you try to solve this set of equations you will receive the error message shown in Figure 4-5.

Figure 4-5: Error message resulting from using saturation temperature and pressure as arguments.

The specific volume cannot be determined given the saturation temperature and pressure since there are many states that all have this temperature and pressure (ranging from saturated vapor to saturated liquid). EES should catch this error.

For two-phase states, it is often convenient to specify the quality as one of the arguments. The quality is the mass fraction of the substance that is in the vapor state. A quality of zero therefore corresponds to saturated liquid whereas a quality of one corresponds to saturated vapor. Values of quality below zero or above one are meaningless. The Quality function will return the quality of the state if it is the two-phase region. The Quality function will return -100 for a state that is in the subcooled liquid phase and 100 for a superheated state.

The specific volumes of ethanol in saturated liquid state and vapor states are found using the following equations:

```
v_f=volume(F$,T=T_bp,x=0)          "specific volume of saturated liquid"
v_g=volume(F$,T=T_bp,x=1)          "specific volume of saturated vapor"
```

which leads to $v_f = 0.001358$ m^3/kg and $v_g = 0.5971$ m^3/kg.

The Example Box

All of the property functions provide an example of a proper function call in the Example box that appears just above the Paste and Done buttons in the Function Information dialog. For example, select Function Information from the Options menu and then Fluid Properties. Select the Real fluids radio button, as shown in Figure 4-6. In the left box is a list of all of the available property functions (from Table 4-1). Select a property (e.g., Enthalpy) and the right box will be populated with all of the possible fluids that can be used with that property (from Table 4-2).

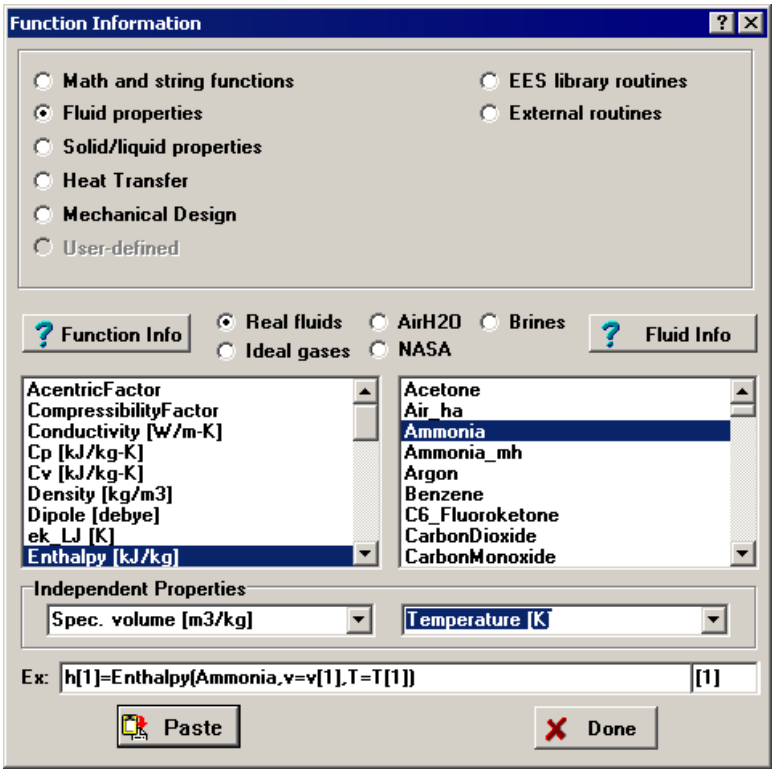


Figure 4-6: Function Information dialog for Real fluids showing the Example box for Ammonia.

Select Fluid Info to obtain information regarding the source(s) of the property information that were used to develop the property functions. In the box labeled Independent Properties the user can select the two properties that are used to fix the state. The text provided in the Example box at the bottom of the dialog will adjust to conform to the selected property, fluid, and independent variables, as shown in Figure 4-6. In many cases, each independent property corresponds to an entry in an array of properties where each index is a state in a cycle. Therefore, the value entered

in the edit box to the right of the Example box is appended to each independent variable. Select Paste to enter the resulting property call in the Equations Window:

```
h[1]=Enthalpy(Ammonia,v=v[1],T=T[1])
```

Vapor Compression Cycle Example

The vapor compression cycle is a closed, steady-state cycle in which the working fluid changes state as it circulates through a compressor, condenser, throttle valve, and evaporator. Figure 4-7 illustrates a schematic of the simple vapor compression cycle that is providing refrigeration to (i.e., receiving heat from) a low temperature reservoir at T_C = 260 K while rejecting heat to a high temperature reservoir at T_H = 320 K using ammonia as the refrigerant.

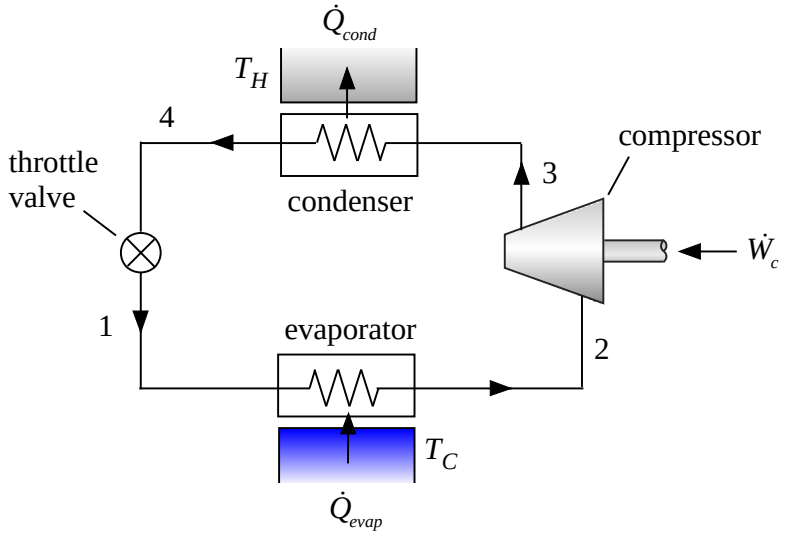


Figure 4-7: The simple vapor compression cycle.

The inputs are entered in EES and the unit system is specified using the $UnitSystem directive.

```
$UnitSystem SI Radian Mass J kg K
T_H=320 [K]                    "high temperature reservoir"
T_C=260 [K]                    "low temperature reservoir"
R$='Ammonia'                   "refrigerant"
```

Assuming that the condenser is a perfect heat exchanger, the refrigerant at state 4 is saturated liquid (x_4 = 0) at $T_4 = T_H$.

```
"State 4, Condenser exit/Throttle inlet"
T[4]=T_H                       "temperature"
x[4]=0 [-]                     "quality"
```

These two intensive properties fix state 4, allowing the specific entropy, pressure, and specific enthalpy (s_4, P_4, and h_4) to be determined using the property functions Entropy, Pressure, and Enthalpy, respectively.

```
s[4]=entropy(R$,T=T[4],x=x[4])                "specific entropy"
P[4]=pressure(R$,T=T[4],x=x[4])               "pressure"
h[4]=enthalpy(R$,T=T[4],x=x[4])               "specific enthalpy"
```

Similarly, the refrigerant leaving the evaporator at state 2 is saturated vapor ($x_2 = 0$) at $T_2 = T_C$.

```
"State 2, Evaporator exit/Compressor inlet"
T[2]=T_C                                      "temperature"
x[2]=1 [-]                                    "quality"
```

These two intensive properties fix state 2, allowing the specific entropy, pressure, and specific enthalpy (s_2, P_2, and h_2) to be determined.

```
s[2]=entropy(R$,T=T[2],x=x[2])                "specific entropy"
P[2]=pressure(R$,T=T[2],x=x[2])               "pressure"
h[2]=enthalpy(R$,T=T[2],x=x[2])               "specific enthalpy"
```

The fluid leaving the condenser is expanded to state 1 in the isenthalpic throttling valve ($h_1 = h_4$). Assuming that there is no pressure loss in the evaporator, $P_1 = P_2$. The specific enthalpy and pressure together fix state 1, allowing the specific entropy and temperature (s_1 and T_1) to be determined.

```
"State 1, Throttle exit/Evaporator inlet"
h[1]=h[4]                                     "specific enthalpy"
P[1]=P[2]                                     "pressure"
s[1]=entropy(R$,h=h[1],P=P[1])                "specific entropy"
T[1]=temperature(R$,h=h[1],P=P[1])            "temperature"
```

The fluid leaving the evaporator is compressed to state 3 in a compressor that is assumed to be reversible and adiabatic, therefore $s_3 = s_2$. Provided that there is no pressure loss in the condenser, $P_3 = P_4$. The specific entropy and pressure together fix state 3 allowing the specific enthalpy and temperature (h_3 and T_3) to be determined.

```
"State 3, Compressor exit/Condenser inlet"
s[3]=s[2]                                     "specific entropy"
P[3]=P[4]                                     "pressure"
h[3]=enthalpy(R$,s=s[3],P=P[3])               "specific enthalpy"
T[3]=temperature(R$,s=s[3],P=P[3])            "temperature"
```

Solving provides all of the properties at each state conveniently organized in the Arrays Table, shown in Figure 4-8. In Section 4.6, we will show how properties in the Arrays Table can be used to create property plots in which the cycle state points are shown (e.g., a *T*-*v* or *P*-*h* diagram).

Main

Sort	1 h_i [J/kg]	2 P_i [Pa]	3 s_i [J/kg-K]	4 T_i [K]	5 x_i [-]
[1]	424767	255361	1871	260	
[2]	1.446E+06	255361	5800	260	1
[3]	1.749E+06	1.872E+06	5800	408.8	
[4]	424767	1.872E+06	1751	320	0

Figure 4-8: Arrays Table with properties.

Energy balances on the condenser, evaporator, and compressor are:

$$\frac{\dot{Q}_{cond}}{\dot{m}} = h_3 - h_4 \tag{4-2}$$

$$\frac{\dot{Q}_{evap}}{\dot{m}} = h_2 - h_1 \tag{4-3}$$

$$\frac{\dot{W}_c}{\dot{m}} = h_3 - h_2 \tag{4-4}$$

The Coefficient of Performance (*COP*) for the cycle is:

$$COP = \frac{\dot{Q}_{evap} / \dot{m}}{\dot{W}_{comp} / \dot{m}} \tag{4-5}$$

The Energy Efficiency Ratio (*EER*) for the cycle is equal to the *COP* expressed in units Btu/hr-W.

```
"Energy balances"
Q_dot_cond\m_dot=h[3]-h[4]                    "condenser"
Q_dot_evap\m_dot=h[2]-h[1]                    "evaporator"
W_dot_c\m_dot=h[3]-h[2]                       "compressor"
COP=Q_dot_evap\m_dot/W_dot_comp\m_dot         "Coefficient of Performance"
EER=COP*convert(-,Btu/hr-W)                  "energy efficiency rating"
```

Solving provides *COP* = 3.371 and *EER* = 11.5 Btu/hr-W.

Equations of State

The relationship between the properties of real fluids can be quite complex. The EES database employs equations of state to relate these properties. Most of the fluids use the fundamental equation of state as the basis for the property relations, as described by Span (2000). This manner of relating properties is regarded as the most accurate method available. However, some fluids in the EES database are represented by the Martin-Hou (1955) equation of state. The fluids that use this equation of state are identified in Table 4-2. The Martin-Hou equation of state implementation provides accurate property information in the saturated and superheated regimes, but assumes that subcooled liquids are incompressible. The specific volume, specific internal energy and specific entropy of subcooled liquids are assumed to be equal to the values of the specific volume, specific internal energy and specific entropy of a saturated liquid state at the same temperature. Specific information regarding the source of information for each fluid can be seen by selecting the fluid name and then clicking the Fluid Info button in Figure 4-3.

Properties of Water

There are several fluid names that all correspond to water. The fluid names Water, Steam, Steam_NBS, and R718 are treated identically. All of these fluid names provide water properties using property correlations published by Harr, Gallagher, and Kell (Hemisphere, 1984). These property correlations were the basis for the international standard for water properties prior to 1995.

The fluid Steam_IAPWS provides the most accurate property data for water using the 1995 Formulation for the Thermodynamic Properties of Ordinary Water Substance for General and Scientific Use, issued by The International Association for the Properties of Water and Steam (IAPWS). This correlation replaced the 1984 formulation of Haar, Gallagher, and Kell. The new formulation is based on the correlations of Saul and Wagner (1987), with modifications to adjust to the International Temperature Scale of 1990. The modifications are described by Wagner and Pruss (1993). The Saul and Wagner correlation provides accurate results for temperatures between 273.15 K and 1273.15 K at pressures up to 1000 MPa. The formulation allows extrapolation of properties to 5000 K. The fluid Steam_IAPWS is available in the Professional and Academic Commercial versions of EES.

Water is an unusual substance in that its specific volume in the solid state can be larger than it is in the liquid state. The fluid Ice provides access to the properties of solid water (i.e., ice) in the calculations when specific volume is provided as one of the arguments. However, water may have two states (one solid and one liquid) with the same specific volume and temperature. Specifying Ice as the substance will force the lower temperature (i.e., solid state) solution, as in the following example. Note that setting the quality to zero results in a specification of the solid state at this specific volume.

```
$UnitSystem SI K Pa J mass rad
v=1.088e-3 [m^3/kg]                          "specific volume"
T_ice=temperature(Ice,v=1.088e-3,x=0)        "temperature of ice"
T_steam=temperature(Steam,v=1.088e-3,x=0)    "temperature of liquid water"
```

Solving indicates that T_ice = 256.3 K and T_steam = 420.9 K.

The Reference State

The values of specific internal energy, specific enthalpy, and specific entropy are not absolute. These values are only defined relative to reference values that are specified at a reference state. The choice of reference state is arbitrary if chemical reactions do not occur. The specification of different reference states is the primary reason that different sources of property information may appear to be providing very different property values. There are several common reference state choices, including:

1. The *International Institute of Refrigeration (IIR)* reference state sets the value of specific enthalpy to be 200 kJ/kg and the value of specific entropy to be 1.0 kJ/kg-K for saturated liquid at 0°C (273.15 K). Note that this option is not applicable to fluids for which the critical temperature is less than 0°C.
2. The *ASHRAE Standard (ASH)* reference state sets the values of specific enthalpy and specific entropy to 0 for saturated liquid at -40°C (-40°F). Note that this option is not applicable to fluids for which the critical temperature is less than -40°C.
3. The *Normal Boiling Point (NBP)* reference state sets the values of specific enthalpy and specific entropy to 0 for saturated liquid at the normal boiling point (i.e., the saturation temperature at one atmosphere). Note that this option is not applicable to fluids for which the critical pressure is less than one atmosphere.

The $Reference Directive

The default reference state for each fluid is specified in the online help property information for that fluid. The $Reference directive allows the reference state to be changed from the default setting to any of those discussed above (IIR, ASH, or NBP). The format of this directive is

```
$Reference  FluidName  ReferenceID
```

where FluidName is the name of the real fluid (as it appears in Table 4-2). Note that the $Reference directive is not applicable to fluids that are modeled with the Martin-Hou or Ideal Gas equations of state. The ReferenceID must be IIR, ASH, NBP, or DFT (which indicates the default reference state). If the reference state choice is not applicable to the fluid then the reference state will remain at its default value.

By default, the reference state for the refrigerant R134a is set to the ASHRAE Standard. The EES code below changes the reference state for the fluid to the IIR standard.

```
$Reference R134a IIR
```

It is possible for the user to add real fluid property information to EES based on the Martin-Hou equation of state, as described in Section 4.10.

4.4 Property Functions for Ideal Gases

The behavior of a gas approaches ideal gas behavior as its pressure is reduced and it temperature is increased. The ideal gas model is computationally simple and provides property information with little computational effort. Many fluids in EES can be modeled as an ideal gas or real fluid.

The Ideal Gas Model

An ideal gas is defined as a fluid that obeys the ideal gas equation of state:

$$PV = n R_{univ} T \tag{4-6}$$

where P is the absolute pressure, V is the volume, n is the number of moles of gas, R_{univ} is the universal gas constant (which does not depend on the type of gas), and T is the absolute temperature (i.e., the temperature expressed in either the Kelvin or Rankine scale). The value of R_{univ} is provided in EES in the specified unit system with the constant R#. The ideal gas law can also be expressed on a mass basis according to:

$$PV = m R T \tag{4-7}$$

where m is mass of gas and R is the ideal gas constant, expressed on a mass basis ($R = R_{univ}/MW$ where MW is the molar mass of the gas). The specific volume of the gas can be defined on either a molar or mass basis by dividing the volume by the number of moles or mass, respectively. The definition of an ideal gas also requires that the specific internal energy (and thus the specific enthalpy) is a function only of temperature.

List of Ideal Gas Fluids

The ideal gas property data are implemented in two groups, referred to as the built-in ideal gases and the NASA ideal gases. Table 4-4 provides the names of all of the built-in ideal gases.

Table 4-4: Names of built-in ideal gases in EES.

Air	C6H14
Ar	C8H18
CH3OH	CO
CH4	CO2
C2H2	H2
C2H4	H2O
C2H5OH	He
C2H6	N2
C3H8	NO2
C4H10	O2
C5H12	SO2

Ideal Gas vs Real Gas Fluids

Note that some substances are represented in EES both as a real fluid and an ideal gas. For example, EES recognizes the fluid names N2 and Nitrogen, H2O and water, and He and Helium. The convention used in EES is that a substance will be modeled as an ideal gas if its fluid name is the chemical name, e.g., N2, H2O, and He. An exception to this rule is the fluid Air, which is modeled as an ideal gas, whereas Air_ha is modeled as a real fluid.

All of the property functions listed in Table 4-1 can be used with the built-in ideal gases. Some of the property functions are not needed for ideal gases, but they still work properly. For example, the compressibility factor of an ideal gas is always 1 at any state and its fugacity is always equal to its pressure. Additional ideal gas property information can be added, as explained in Section 4.10.

The use of an ideal gas fluid affects the number of arguments that the property functions require in some cases and also the reference state used for specific internal energy, specific enthalpy and specific entropy. The EES code below determines the specific enthalpy of nitrogen at T = 300 K and P = 100000 Pa using the real gas fluid Nitrogen.

```
$UnitSystem SI K Pa J mass rad
T=300 [K]                          "temperature"
P=100000 [Pa]                      "pressure"
h_RG=enthalpy(Nitrogen,T=T,P=P)    "enthalpy of Nitrogen"
```

Solving leads to h_{RG} = 311,197 J/kg.

If you attempt to determine the specific enthalpy of nitrogen at T = 300 K and P = 100000 Pa with the ideal gas fluid N2 using the same two properties (temperature and pressure) to fix the state:

```
h_IG=enthalpy(N2,T=T,P=P)          "enthalpy of N2"
```

you will receive the error message shown in Figure 4-9.

Figure 4-9: Error message.

Because N2 is an ideal gas, the specific enthalpy (and specific internal energy) can only be a function of temperature. Therefore, the Cp, Cv, Enthalpy and IntEnergy functions in EES will only accept temperature (or combinations of two properties from which temperature can be determined, such as specific volume and pressure) in addition to the fluid name. Therefore, the specific enthalpy should be determined according to:

```
h_IG=enthalpy(N2,T=T)              "enthalpy of N2"
```

which leads to h_{IG} = 1,920 J/kg. The large difference between h_{RG} and h_{IG} is due to the fact that the reference states used for the fluids Nitrogen and N2 are very different. The reference states for the specific enthalpy of all ideal gases (except air) are chosen relative to the elements from which the gas is formed having a specific enthalpy of 0 kJ/kmol at 25°C (298.15 K). This reference state choice makes it convenient to use the ideal gas property information for energy calculations involving chemical reactions, as explained below.

The NASA Ideal Gas Database

The NASA ideal gas database contains data for 1262 ideal gases. A list of the ideal gas names in this database can be obtained from the Function Information dialog shown in Figure 4-6 by clicking the NASA radio button and then clicking the Fluid Info button. The property data for the NASA ideal gases were obtained from McBride et al., (2002). The property functions that are implemented for the NASA ideal gases are listed in Table 4-5. Note that transport properties, such as thermal conductivity and viscosity, are not available for the NASA ideal gases.

The NASA ideal gas property library was integrated into EES in version 8.528. Prior to this version, ideal gas data from the NASA database had to be accessed with a call to the NASA external procedure. This external procedure is still of interest because it provides data for solid materials as well as gases. The use of this external procedure is discussed in Section 4.9.

Table 4-5: Thermodynamic property functions available for the NASA ideal gas database and their units.

Function Name	Returns	SI Units	English Units
CompressibilityFactor	compressibility factor	none	none
Cp	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Cv	constant volume specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Density	density	kg/m^3	lb_m/ft^3
Enthalpy	specific enthalpy	J/kg, kJ/kg	Btu/lb_m
Entropy	specific entropy	J/kg-K, kJ/kg-K	Btu/lb_m-R
Fugacity	fugacity	Pa, kPa, bar, MPa	psia, atm
IntEnergy	specific internal energy	J/kg, kJ/kg	Btu/lb_m
IsIdealGas	0 for real fluid, 1 for ideal gas	none	none
Pressure	absolute pressure	Pa, kPa, bar, MPa	psia, atm
MolarMass[1]	molecular weight	kg/kmol	lb_m/lbmol
SoundSpeed	speed of sound	m/s	ft/s
SpecHeat	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Temperature	temperature	°C, K	°F, R
VolExpCoef	coefficient of thermal expansion	1/K	1/R
Volume	specific volume	m^3/kg	ft^3/lb_m

1. The function requires only the name of the fluid.

The IsIdealGas Function

The IsIdealGas function accepts as its only argument the name of the fluid and returns 1 if the fluid is any built-in or NASA ideal gas and 0 otherwise. This function is particularly useful if you are developing a function or procedure that takes the fluid name as an input and must work properly regardless of whether the fluid corresponds to an ideal gas or a real fluid. For example, the function Turbine, developed below, computes the outlet temperature for a turbine given its isentropic efficiency. The inputs to the function include the fluid name (in the string F$) as well as the inlet temperature and pressure (T_{in} and P_{in}), the outlet pressure (P_{out}), and the isentropic efficiency (η).

```
$UnitSystem SI Mass J K Pa Rad
$TabStops 0.25 0.5 3

Function Turbine(F$,T_in,P_in,P_out,eta)
```

The inlet state is fixed by the inlet temperature and pressure. The inlet specific entropy is determined using the Entropy function. Note that specific entropy (s_{in}) is a function of temperature and pressure regardless of whether the fluid is an ideal gas or a real fluid.

```
    s_in=Entropy(F$,T=T_in,P=P_in)     "inlet specific entropy"
```

The inlet specific enthalpy (h_{in}) is determined using the Enthalpy function. Note that the Enthalpy function must be called using only temperature if the fluid provided by the string F$ is an ideal gas. Otherwise both temperature and pressure must be used. This logic is accomplished using the IsIdealGas function and an If-Then-Else construct (Section 3.4).

```
    If (IsIdealGas(F$)=1) Then
        h_in=Enthalpy(F$,T=T_in)                "inlet specific enthalpy for ideal gas fluid"
    Else
        h_in=Enthalpy(F$,T=T_in,P=P_in)         "inlet specific enthalpy for real fluid"
    EndIf
```

The specific enthalpy at the outlet state for a reversible turbine ($h_{out,s}$) is obtained from the outlet pressure and inlet specific entropy using the Enthalpy function. Note that pressure and specific entropy together are required in order to determine temperature; therefore, both inputs are required regardless of whether the fluid is a real fluid or an ideal gas.

```
    h_out_s=Enthalpy(F$,P=P_out,s=s_in)     "outlet specific enthalpy for reversible turbine"
```

The outlet specific enthalpy of the actual turbine (h_{out}) is computed according to:

$$h_{out} = h_{in} - \left(h_{in} - h_{out,s}\right)\eta \tag{4-8}$$

```
    h_out=h_in-(h_in-h_out_s)*eta     "outlet specific enthalpy for actual turbine"
```

The outlet state is fixed by the outlet specific enthalpy and outlet pressure. Specific enthalpy is a function only of temperature for an ideal gas. Therefore, the converse must also be true;

temperature must be a function only of specific enthalpy for an ideal gas. If the fluid is an ideal gas then the outlet temperature is obtained using the Temperature function called only with specific enthalpy. Otherwise the Temperature function is called with both specific enthalpy and pressure.

```
    If (IsIdealGas(F$)=1) Then
        Turbine=Temperature(F$,h=h_out)              "outlet temperature for ideal gas fluid"
    Else
        Turbine=Temperature(F$,h=h_out,P=P_out)      "outlet temperature for real fluid"
    EndIf
End
```

The function Turbine will work when called with either an ideal gas (e.g., Air):

```
T_out=Turbine('Air',1100 [K], 300000 [Pa], 100000 [Pa], 0.75 [-])
```

or a real fluid (e.g., Air_ha):

```
T_out=Turbine('Air_ha',1100 [K], 300000 [Pa], 100000 [Pa], 0.75 [-])
```

The Ideal Gas Reference State & Chemical Reactions

An important application of ideal gas properties is for modeling chemical reactions. In this type of application, it is typically best to set the unit system to be on a molar basis so that all specific properties are reported per mole of gas rather than per unit mass. The reference states for specific enthalpy and specific entropy are important for applications in which chemical reactions occur, as discussed in Klein and Nellis (2012). The reference state used to provide the specific enthalpy for all of the ideal gases implemented in EES is based on the stable elements having a zero specific enthalpy at 25°C. Therefore, the absolute value of specific enthalpy includes the enthalpy of formation associated with forming chemical bonds. The reference state used to provide the specific entropy for all ideal gases in EES is based on the Third Law of Thermodynamics, which dictates that the entropy of any pure substance must be zero at 0 K. These choices make it easy to work with chemical reactions. For example, the stoichiometric reaction between methane and oxygen is given by:

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O \tag{4-9}$$

The lower heating value of methane is the difference between the molar specific enthalpy of the reactants and the molar specific enthalpy of the products for this reaction evaluated at 25ºC and assuming that the water in the products is all in the gas phase. This calculation can be easily accomplished using EES because the reference state used for the ideal gases involved in the reaction includes the specific enthalpy associated with the chemical bonds in the molecules. Therefore, the specific enthalpy reported by EES for an ideal gas is the standardized specific enthalpy that can be directly used in energy balances for chemical reactions.

The $UnitSystem directive is used to specify a molar rather than a mass basis. The molar specific enthalpies of each of the substances involved in the reaction ($\bar{h}_{CH_4}$, $\bar{h}_{O_2}$, $\bar{h}_{CO_2}$, and $\bar{h}_{H_2O}$) are determined using the Enthalpy function and assuming that the substances behave as an ideal gas.

```
$UnitSystem SI Mole J K Pa

T=ConvertTemp(C,K,25 [C])                "temperature"
h_bar_CH4=Enthalpy(CH4,T=T)              "molar specific enthalpy of CH4"
h_bar_O2=Enthalpy(O2,T=T)                "molar specific enthalpy of O2"
h_bar_CO2=Enthalpy(CO2,T=T)              "molar specific enthalpy of CO2"
h_bar_H2O=Enthalpy(H2O,T=T)              "molar specific enthalpy of H2O"
```

The enthalpy of the reactants and the products of the reaction shown in Eq. (4-9) per mole of CH_4 (H_R and H_P) are computed:

$$H_R = \bar{h}_{CH_4} + 2\bar{h}_{O_2} \tag{4-10}$$

$$H_P = \bar{h}_{CO_2} + 2\bar{h}_{H_2O} \tag{4-11}$$

and used to determine the lower heating value:

$$LHV = H_R - H_P \tag{4-12}$$

```
H_R=h_bar_CH4+2*h_bar_O2                 "enthalpy of reactants per mole of CH4"
H_P=h_bar_CO2+2*h_bar_H2O                "enthalpy of products per mole of H2O"
LHV=H_R-H_P                              "lower heating value"
```

Solving leads to LHV = 8.025x10^8 J/kmol. Dividing by the molar mass of CH_4 results in the lower heating value expressed on a unit mass basis.

```
LHV_mass=LHV/MolarMass(CH4)              "lower heating value on a mass basis"
```

The lower heating value for methane is 50,020 kJ/kg.

4.5 Psychrometric Properties

Psychrometics is the term used to refer to the study of mixtures of air and water vapor at conditions near atmospheric pressure and temperature. The properties of these mixtures are of practical interest for heating and cooling applications.

The Fluid AirH2O

EES provides property information for psychrometric mixtures when AirH2O is used as the fluid name. Air and water vapor mixtures behave according to the ideal gas law at atmospheric pressure. However, psychometric property functions differ from the ideal gas property functions presented in Section 4.4 in that there is an extra degree of freedom introduced related to the concentration of water vapor that is contained in the air. Therefore, an additional (third) property is required to fix the state when AirH2O is used; this third property must be related to the concentration of water vapor.

Properties Specific to Psychrometrics

The amount of water vapor is quantified in terms of its humidity ratio, ω, which is defined as the mass of water vapor to the mass of dry air. The amount of water in the air can also be quantified in terms of the relative humidity (ϕ), which is defined as the ratio of the vapor pressure of the water to the saturation vapor pressure of water at the same temperatures.

All of the properties that are of interest for other fluids (e.g., specific enthalpy, thermal conductivity, etc.) are also available for air water vapor mixtures using the fluid AirH2O. Other properties that are specific to psychrometrics include the dew point and wet bulb temperatures. The dew point temperature (T_{dp}) is defined as the temperature at which water will condense when the mixture is cooled at constant pressure. The wet bulb temperature (T_{wb}) refers to the temperature a wetted material such as a piece of cotton will come to when it is exposed to humid air at a specified state. The wet bulb temperature is usually approximated with the adiabatic saturation temperature of the air water mixture (T_{as}), which is the temperature that an air water vapor mixture will achieve if it is humidified adiabatically. Detailed information on these properties is provided in Klein and Nellis (2012).

List of Psychrometric Properties

The property functions that are available for fluid AirH2O are listed in Table 4-6.

List of Indicators for Psychrometric Properties

Three properties must be included in order to fix the state when using any of the property functions with the substance AirH2O, rather than the two properties that are required to fix the state of a pure fluid. One of the three properties must be the pressure. Note that there is no function to return pressure listed in Table 4-6 as there is for real fluids and ideal gases. The properties used to fix the state are identified in the usual way, with a single case-insensitive letter (i.e., the indicator) followed by an equal sign. The one letter indicators that are recognized in psychrometric functions and their meaning are listed in Table 4-7. Some of these indicators are applicable for real fluids and ideal gases, but there are several that are only applicable to the substance AirH2O.

Table 4-6: Thermodynamic and transport property functions for air-water vapor mixtures that can be accessed using the fluid AirH2O.

Function Name	Returns	SI Units	English Units
CompressibilityFactor	compressibility factor	none	none
Conductivity	thermal conductivity	W/m-K	Btu/hr-ft-R
Cp	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Cv	constant volume specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Density	density	kg/m^3	lb_m/ft^3
DewPoint	dew point temperature	°C, K	°F, R
Entropy	specific entropy	J/kg-K, kJ/kg-K	Btu/lb_m-R
Fugacity	fugacity	Pa, kPa, bar, MPa	psia, atm
HumRat	humidity ratio	none	none
IntEnergy	specific internal energy	J/kg, kJ/kg	Btu/lb_m
RelHum	relative humidity	none	none
SpecHeat	constant pressure specific heat	J/kg-K, kJ/kg-K	Btu/lb_m-R
Temperature	temperature	°C, K	°F, R
Viscosity	viscosity	kg/m-s	lb_m/ft-hr
Volume	specific volume	m^3/kg	ft^3/lb_m
Wetbulb	adiabatic saturation temperature	°C, K	°F, R

Table 4-7: One letter indicators for EES property functions with AirH2O.

Indicator	Thermodynamic Property
B	thermodynamic wet bulb temperature (only for substance AirH2O)
D	dew point temperature (only for substance AirH2O)
H	dry air specific enthalpy[1]
P	pressure
R	relative humidity (only for substance AirH2O)
S	dry air specific entropy[1]
T	temperature
U	dry air specific internal energy[1]
V	dry air specific volume[1]
W	humidity ratio (only for substance AirH2O)

1. Note that the term dry air specific indicates that the property is returned per mass of dry air in the air water mixture rather per mass of mixture. This is the conventional method for providing specific properties for an air water mixture.

Psychrometrics Example

The following example determines all of the available thermodynamic and transport properties for an air water mixture at 25°C and 101.3 kPa with a 40% relative humidity.

```
$UnitSystem SI Mass K Pa J Rad
T=ConvertTemp(C,K,25 [C])                    "dry bulb temperature"
P=101.3 [kPa]*convert(kPa,Pa)                "pressure"
phi=0.40 [-]                                 "relative humidity"

T_dp=DewPoint(AirH2O,T=T,P=P,R=phi)          "dew point temperature"
T_dp_C=ConvertTemp(K,C,T_dp)                 "dew point temperature, in C"
T_wb=WetBulb(AirH2O,T=T,P=P,R=phi)           "wet bulb temperature"
T_wb_C=ConvertTemp(K,C,T_wb)                 "wet bulb temperature, in C"
```

```
omega=HumRat(AirH2O,T=T,P=P,R=phi)      "humidity ratio"
v=Volume(AirH2O,T=T,P=P, R=phi)         "dry air specific volume"
rho=Density(AirH2O,T=T,P=P,R=phi)       "density"
u=IntEnergy(AirH2O,T=T,P=P,R=phi)       "dry air specific internal energy"
h=Enthalpy(AirH2O,T=T,P=P,R=phi)        "dry air specific enthalpy"
s=Entropy(AirH2O,T=T,P=P,R=phi)         "dry air specific entropy"

k=Conductivity(AirH2O,T=T,P=P,R=phi)    "thermal conductivity"
mu=Viscosity(AirH2O,T=T,P=P,R=phi)      "viscosity"
```

The Solution Window that results after solving this example is shown in Figure 4-10.

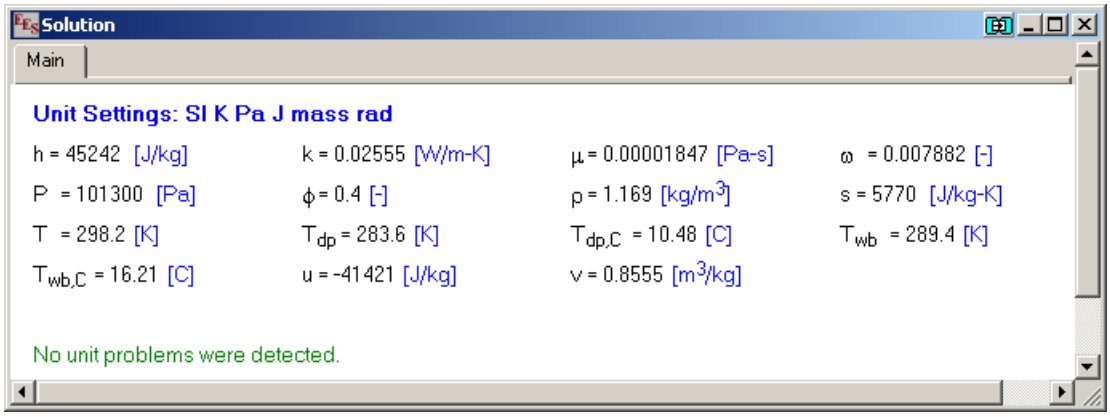


Figure 4-10: Solution Window resulting from solving the psychrometric example.

As indicated in Table 4-7, a variety of different combinations of properties can be used as the independent variables in the property calls. For example, the same solution that is shown in Figure 4-10 would result if the thermodynamic function calls were written as:

```
v=Volume(AirH2O,T=T,P=P,w=omega)        "dry air specific volume"
rho=Density(AirH2O,T=T,P=P,D=T_dp)      "density"
u=IntEnergy(AirH2O,h=h,P=P,R=phi)       "dry air specific internal energy"
h=Enthalpy(AirH2O,v=v,P=P,w=omega)      "dry air specific enthalpy"
s=Entropy(AirH2O,h=h,P=P,R=phi)         "dry air specific entropy"
```

Note that pressure is always required as an independent property and that there are some combinations of independent properties that are not accepted. For example, EES will not accept pressure, wet bulb, and relative humidity. The equation:

```
h2=Enthalpy(AirH2O,B=T_wb,P=P,R=phi)    "not accepted"
```

will result in the error message shown in Figure 4-11; this is an implementation limitation but it is not a major problem since additional equations can be used to obtain the required independent properties.

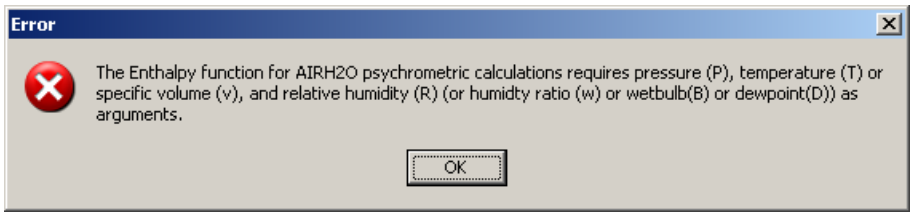


Figure 4-11: Error message resulting from an unaccepted combination of independent properties.

4.6 Property Plots

Property plots are commonly used in thermodynamic cycle models. For many years such plots were used to estimate properties graphically. Even if property information is obtained using other means, property plots are still very useful for visualizing thermodynamic cycles and processes. Cycle states and process trajectories can be understood more easily by overlaying them onto plots with property coordinates (e.g., *T*-*v* plots, *T*-*s* plots, and *P*-*h* plots).

The Property Plot Dialog

Different types of property plots can be automatically produced by EES by selecting Property Plot from the Plots menu. The Property Plot dialog that results is shown in Figure 4-12.

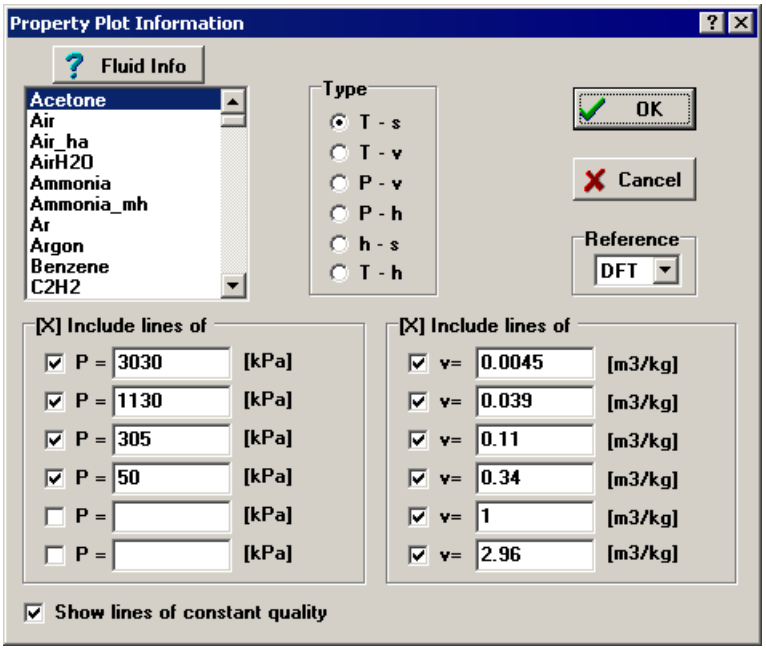


Figure 4-12: Property plot dialog window.

Property plots can be produced for any of the real fluids and built-in ideal gases. The fluid is selected from the list at the upper left of the dialog window. The type of plot is selected with the list in upper center of the dialog by selecting a radio button. The lower half of the dialog provides the opportunity to enter up to six values that will be used to generate lines of a constant property (e.g., isotherms, isobars, or isochors). The constant property choice depends on the type of property plot that has been selected. For example, lines of constant pressure and constant

specific volume can be optionally placed on a temperature-specific entropy plot (*T*-*s* plot) as shown in Figure 4-12. Suggested values for the constant properties are automatically generated, but they can be changed or deleted based on user preference. The constant property line is plotted only if the check box preceding the edit control showing the value is checked. To uncheck all of the values (i.e., to forego generating lines of constant property), de-select the Include lines of by clicking in the space between the brackets [ ] above the list.

Property Plots for Real Fluids

The property plots for real fluids will include the vapor dome. The check box at the bottom labeled Show lines of constant quality is visible only for real fluids. If the box at the bottom of the dialog is checked then lines of constant quality will be shown within the vapor dome. The temperature-specific entropy plot for acetone that is produced with the default values shown in Figure 4-12 appears in Figure 4-13.

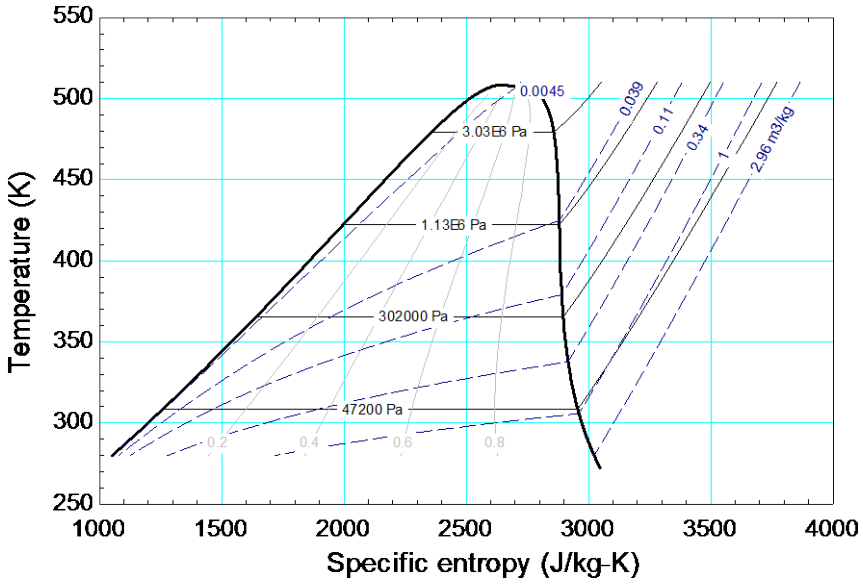


Figure 4-13: Temperature-specific entropy plot for Acetone.

Property Plots for Ideal Gases

If an ideal gas is selected then there will be no vapor dome included. For example, Figure 4-14 shows the temperature-specific entropy plot that is generated using the default settings for the fluid Air.

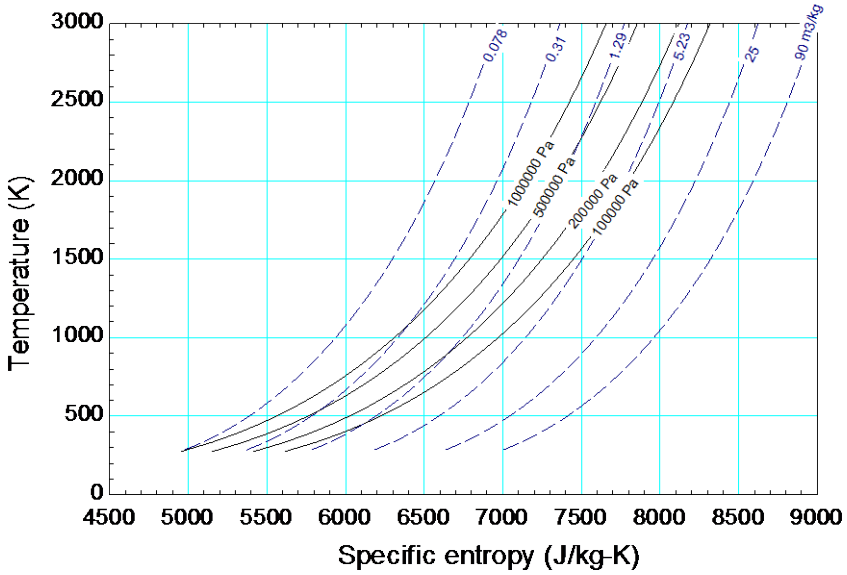


Figure 4-14: Temperature-specific entropy plot for Air.

Psychrometric Plots

If the fluid AirH2O is selected then the Property Plot dialog will appear as shown in Figure 4-15 in order to allow the generation of a psychrometric plot. The dialog allows the pressure and the dry-bulb temperature limits to be entered. Lines of constant wet-bulb temperature and specific volume can optionally be placed on the plot, as well as lines of constant relative humidity. The psychrometric plot produced with the information in Figure 4-15 is shown in Figure 4-16.

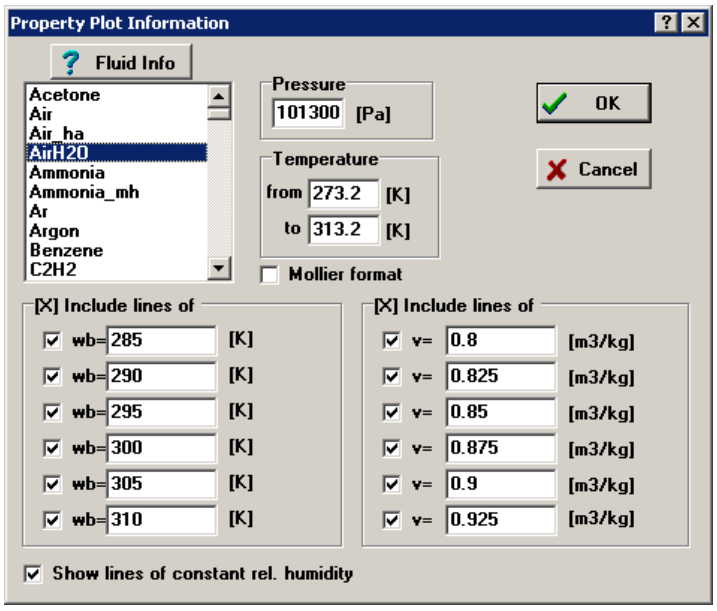


Figure 4-15: Property plot dialog window for a psychrometic plot.

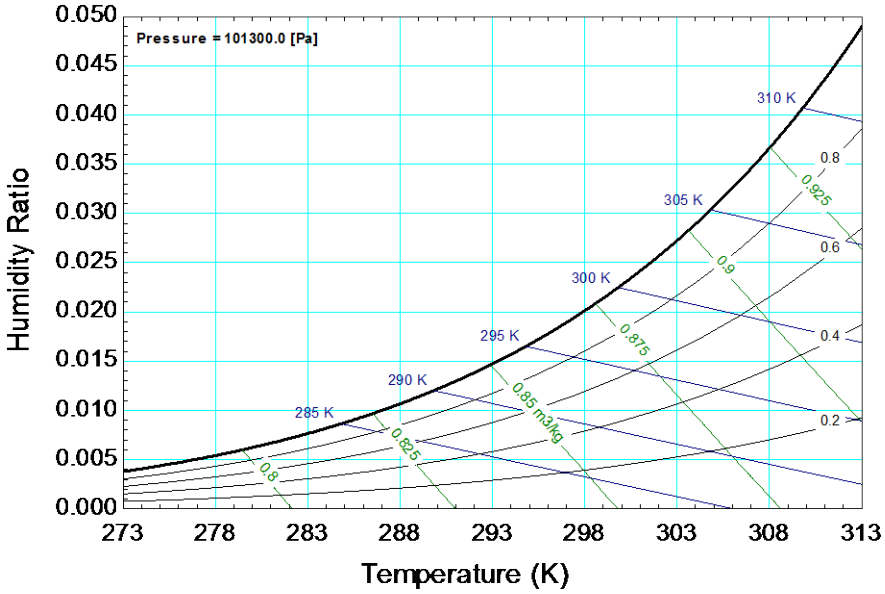


Figure 4-16: Psychrometric chart produced using the input shown in Figure 4-15.

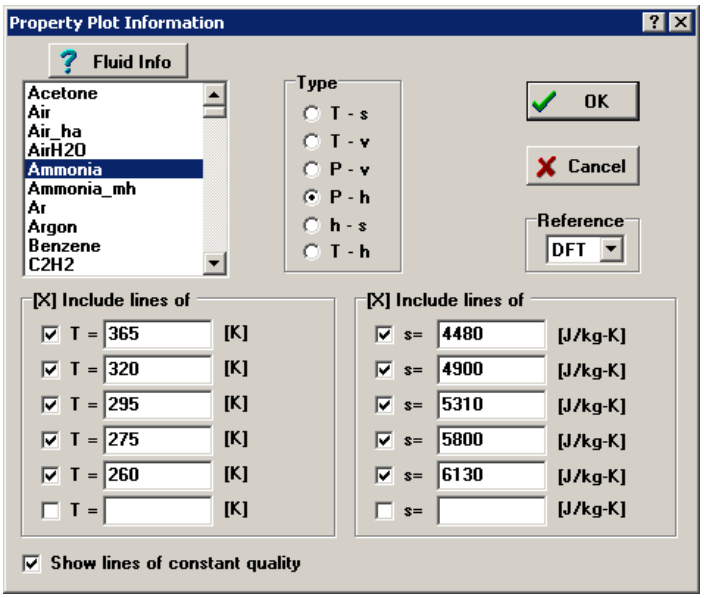
Overlaying States onto Property Plots

The property plots themselves are not of particular use as they cannot be read nearly as accurately as the property values that are returned from the property functions provided by EES. However, the plots become much more useful when calculated property information is overlayed onto the property plot using the Overlay Plot command. This process is most easily accomplished by using array variables for the property information at each state. The example provided in Section 4.3 is used to illustrate this capability. The code accomplishes an analysis of a vapor compression refrigeration cycle and is repeated below.

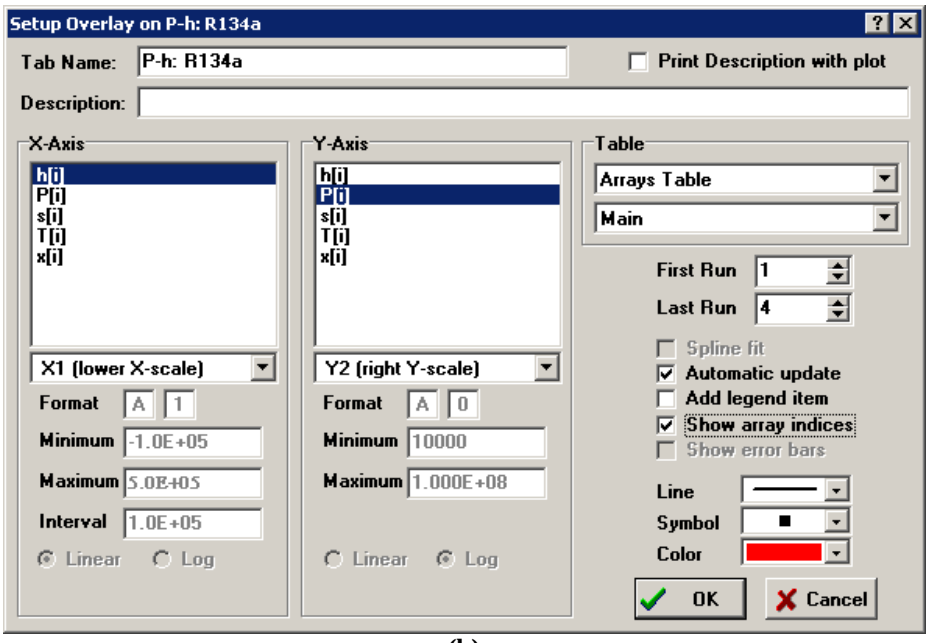
```
$UnitSystem SI Radian Mass J kg K
T_H=320 [K]                                  "high temperature reservoir"
T_C=260 [K]                                  "low temperature reservoir"
R$='Ammonia'                                 "refrigerant"

"State 4, Condenser exit/Throttle inlet"
T[4]=T_H                                     "temperature"
x[4]=0 [-]                                   "quality"
s[4]=entropy(R$,T=T[4],x=x[4])               "specific entropy"
P[4]=pressure(R$,T=T[4],x=x[4])              "pressure"
h[4]=enthalpy(R$,T=T[4],x=x[4])              "specific enthalpy"

"State 2, Evaporator exit/Compressor inlet"
T[2]=T_C                                     "temperature"
x[2]=1 [-]                                   "quality"
s[2]=entropy(R$,T=T[2],x=x[2])               "specific entropy"
P[2]=pressure(R$,T=T[2],x=x[2])              "pressure"
h[2]=enthalpy(R$,T=T[2],x=x[2])              "specific enthalpy"

"State 1, Throttle exit/Evaporator inlet"
h[1]=h[4]                                    "specific enthalpy"
P[1]=P[2]                                    "pressure"
s[1]=entropy(R$,h=h[1],P=P[1])               "specific entropy"
```

```
T[1]=temperature(R$,h=h[1],P=P[1])                "temperature"

"State 3, Compressor exit/Condenser inlet"
s[3]=s[2]                                         "specific entropy"
P[3]=P[4]                                         "pressure"
h[3]=enthalpy(R$,s=s[3],P=P[3])                   "specific enthalpy"
T[3]=temperature(R$,s=s[3],P=P[3])                "temperature"

Q_dot_cond\m_dot=h[3]-h[4]                        "condenser heat transfer per mass"
Q_dot_evap\m_dot=h[2]-h[1]                        "evaporator heat transfer per mass"
W_dot_comp\m_dot=h[3]-h[2]                        "compressor work per mass"
COP=Q_dot_evap\m_dot/W_dot_comp\m_dot             "Coefficient of Performance"
EER=COP*convert(-,Btu/hr-W)                       "energy efficiency rating"
```

After solving this problem, a pressure–specific enthalpy property plot is constructed using the default parameters for the fluid Ammonia. The Overlay Plot is then used to display the property information for all four states in the refrigeration cycle on the property plot. The options chosen in the Properties Plot dialog are shown in Figure 4-17(a) and the options in the Overlay Plot dialog are shown in Figure 4-17(b). Note that the Show array indices check box is selected. This option will place the number of each state next to its symbol on the property plot. Also notice that the Automatic update check box is selected so that any changes that are made to the refrigeration cycle model will be immediately reflected in the property plot. The completed plot is shown in Figure 4-18.



(a)



(b)

Figure 4-17: (a) Property Plot dialog and (b) Overlay Plot dialog used to generate Figure 4-18 for the refrigeration cycle example.

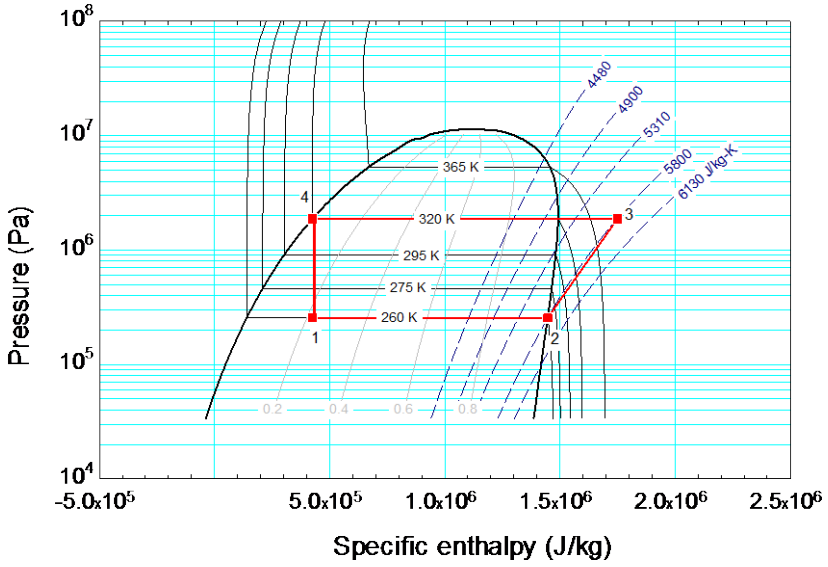


Figure 4-18: Pressure-specific enthalpy plot with refrigeration cycle property information overlaid.

4.7 Brine Properties

Brines are also called secondary refrigerants. These fluids are mixtures of water and another substance that result in a reduction of the freezing point. The properties of primary interest for brines are its specific heat capacity and its freezing point. The thermal conductivity and viscosity of brines are also of interest because they affect the heat transfer characteristics as well as the pressure drop and therefore the pumping costs. Brine property correlations have been published by the International Institute of Refrigeration (Melinder (2010)) and these correlations have been implemented in EES. All of the brine properties are functions of temperature and the concentration of the solute.

Brine Property Functions

The property functions applicable to brines and the units of the value that the function returns are summarized in Table 4-8. These functions are visible in the Function Information dialog window when the Brines button is selected, as shown in Figure 4-19.

Table 4-8: Brine property functions and their units.

Function Name	Returns	SI Units	English Units
Conductivity	thermal conductivity	W/m-K	Btu/hr-ft-R
Cp	specific heat capacity	J/kg-K, kJ/kg-K	Btu/lb_m-R
Density	density	kg/m^3	lb_m/ft^3
FreezingPt	freezing point temperature	°C, K	°F, R
Prandtl	Prandtl number	none	none
Viscosity	dynamic viscosity	Pa-s	lb_m/ft-hr
Volume	specific volume	m^3/kg	ft^3/lb_m

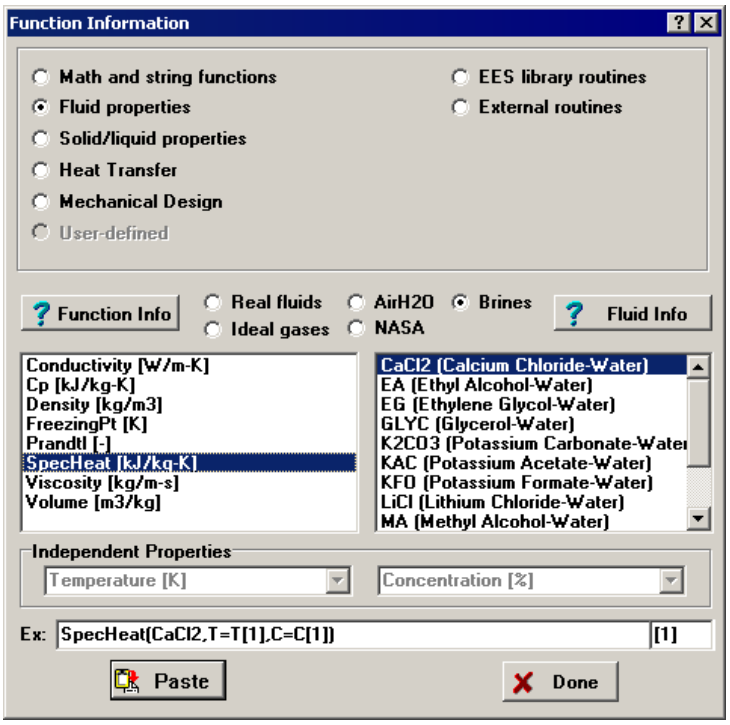


Figure 4-19: Function Information dialog that results when the Brines button is selected.

Brine Fluid Mixtures

Each of the brine property functions requires three parameters. The first parameter must be the short name of the mixture, which can be provided as a string constant or string variable. Quotes around the string constant are not required. The short version of the fluid mixture names for which data are currently provided are listed in Table 4-9 along with their complete names.

Table 4-9: Brine fluid mixture names.

Short Name	Complete Name
CACL2	Calcium Chloride-Water
EA	Ethyl Alcohol-Water
EG	Ethylene Glycol-Water
GLYC	Glycerol-Water
K2CO3	Potassium Carbonate-Water
KAC	Potassium Acetate-Water
KFO	Potassium Formate-Water
LICL	Lithium Chloride-Water
MA	Methyl Alcohol-Water
MGCL2	Magnesium Chloride-Water
NACL	Sodium Chloride-Water
NH3W	Ammonia-water
PG	Propylene Glycol-Water

Using the Brine Property Functions

The next two input parameters are the temperature and mass concentration of solute (expressed as a percentage) specified with the indicators T = and C =, respectively. These parameters can be provided in any order. The temperature must be provided in the units that are specified in Unit System dialog or by the $UnitSystem directive, as described in Section 4.1.

The following example determines property information for a 40% mass concentration of ethylene glycol in water ('EG') at 0°C (273.2 K).

```
$UnitSystem SI Mass K Pa J Rad
T=273.2 [K]
Conc=40 [%]
Brine$='EG'
rho=density(Brine$,T=T,C=Conc)              "density"
T_fp=FreezingPt(Brine$,T=T,C=Conc)          "freezing point"
T_fp_C=ConvertTemp(K,C,T_fp)                "in C"
cP=SpecHeat(Brine$,T=T,C=Conc)              "specific heat capacity"
mu=viscosity(Brine$,T=T,C=Conc)             "dynamic viscosity"
k=conductivity(Brine$,T=T,C=Conc)           "thermal conductivity"
```

Solving provides ρ = 1060 kg/m^3, T_{fp} = -23.87°C, c_P = 3434 J/kg-K, μ = 0.005799 Pa-s, and k = 0.4099 W/m-K.

The BrineProp2 External Procedure

The Brine property functions were implemented in EES version 8.785. Prior to this version, brine properties were not provided as internal property data and were instead provided with a call to the BrineProp2 external procedure. The brine property data used for the internal brine property data in EES is based on newer correlations than is used in the BrineProp2 procedure. However, the BrineProp2 external procedure is still provided with EES for backward compatibility. Information on the BrineProp2 external procedure is available in the Function Information dialog window by clicking on the EES library routines button and scrolling to the BrineProp2.lib folder. The format of the call to the BrineProp2 procedure is:

```
Call BrineProp2(Brine$,Conc,Temp:FreezingPt, Density, SpecHeat, ThermalC, DynVisc, Pr)
```

where Brine$ is the name of the mixture (as given in Table 4-9), Conc is the concentration (in mass %), Temp is the temperature (in °C), FreezingPt is the freezing temperature (in °C), Density is the density of the fluid (in kg/m^3), SpecHeat is the specific heat capacity (in kJ/kg-K), ThermalC is the thermal conductivity (in W/m-K), DynVisc is the dynamic viscosity (in Pa-s), and Pr is the Prandtl number. Note that the units of the inputs and outputs to this procedure must be as indicated above and summarized in Table 4-10, regardless of the settings of the unit system in EES.

Table 4-10: Required units for variables used with the BrineProp2 external procedure.

Variable	Property	Units
Conc	concentration in mass percent	none
Temp	temperature	°C
FreezingPt	freezing point	°C
Density	density	kg/m^3
SpecHeat	specific heat	kJ/kg-K
ThermalC	thermal conductivity	W/m-K
DynVisc	dynamic viscosity	Pa-s
Pr	Prandtl number	none

The following line of code, added to those from the previous section, provides the property information for a 40% mass concentration of ethylene glycol in water at 0°C using the BrineProp2 procedure.

```
Call BrineProp2(Brine$,Conc,converttemp(K,C,T):T_fp_2, rho_2, cP_2, k_2, mu_2, Pr)
                                                    "properties using BrineProp2 Procedure"
```

Solving provides ρ = 1061 kg/m^3, T_{fp} = -23.81°C, c_P = 3431 J/kg-K, μ = 0.005786 Pa-s, and k = 0.4096 W/m-K. Note the slight differences between these properties and those obtained in the previous section due to the fact that an older set of correlations are being used within the BrineProp2 procedure.

4.8 Solid/Liquid Properties

The Solid/Liquid property data library in EES provides the properties of solid and liquid materials (i.e., incompressible substances) as a function of temperature. Information and example function calls for the Solid/Liquid library are obtained by clicking the Solid/Liquid Property button in the Function Information dialog. The dialog window will appear as shown in Figure 4-20.

The data used to develop this library have been obtained from many sources. A list of references for the property data can be viewed by clicking the Property Info button in the Function Information dialog window shown in Figure 4-20.

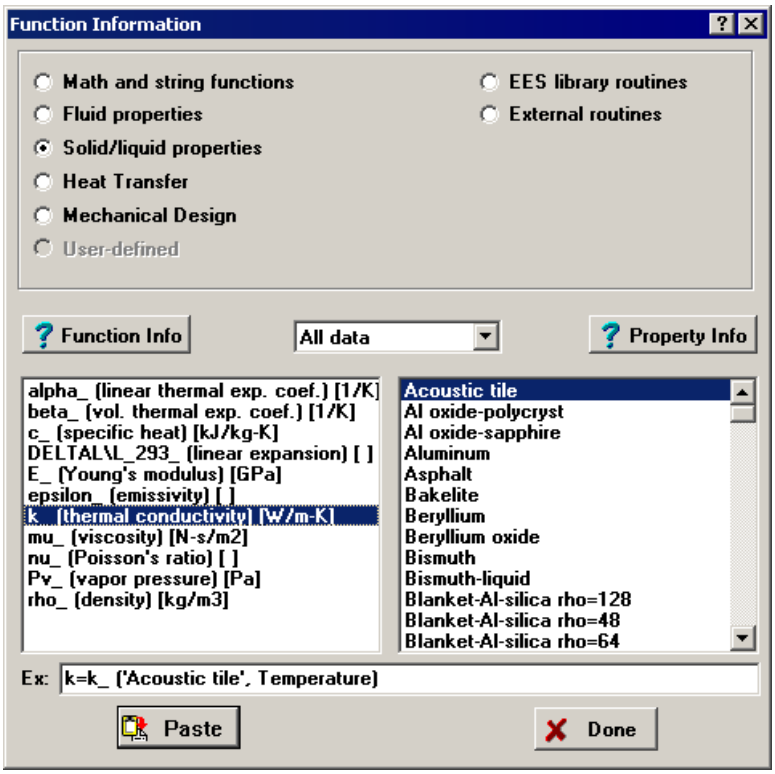


Figure 4-20: Function Information dialog window configured for Solid/Liquid properties.

Solid/Liquid Property Functions

As currently configured, the Solid/Liquid property library can provide the properties listed in Table 4-11 as a function of temperature. The name of all of the Solid/Liquid functions ends with an underscore character (_).

Table 4-11: Solid/liquid property functions and their units.

Function Name	Returns	SI Units	English Units
alpha_	linear coef. of thermal expansion	1/K	1/R
beta_	volumetric coef. of thermal expansion	1/K	1/R
c_	specific heat capacity	J/kg-K, kJ/kg-K	Btu/lb_m-R
DELTAL\L_293_	linear expansion[1]	none	none
E_	Young's modulus	GPa	psi
epsilon_	emissivity	none	none
k_	thermal conductivity	W/m-K	Btu/hr-ft-R
mu_	dynamic viscosity	Pa-s	lb_m/ft-hr
nu_	Poisson's ratio	none	none
Pv_	vapor pressure	kPa	psia
rho_	density	kg/m^3	lb_m/ft^3

1. Linear expansion refers to the change in length of a sample normalized by its length at 20ºC and corresponds to the linear coefficient of thermal expansion integrated from 20ºC to the temperature of interest.

Shown on the right side of the Function Information dialog (Figure 4-20) are the names of substances for which the selected Solid/Liquid function can be applied. Note that not all of the property functions are available for or apply to all of the substances. For example, the vapor pressure function (Pv_) is only applicable to liquid substances. The substance name must be provided as the first argument in the function call. It can be entered as a string constant (with optional quotes) or a string variable. The drop-down list box in the center allows a selection criterion to be applied to the list of substances. The selection criteria are: All Data, Metals, Liquid Metals, Building Materials, Insulation, Fluids, Molten Salts and Miscellaneous. Using the selection criterion shortens the list of substance names.

Using the Solid/Liquid Property Functions

The Solid/Liquid property functions require one numerical argument in addition to the substance name and that argument must be the temperature provided in the unit system that is specified as indicated in Section 4.1. Thus the calling protocol for a Solid/Liquid property function is:

```
X = FunctionName(Material$, Temperature)
```

where FunctionName is the name of the Solid/Liquid property function, Material$ is a string or string variable that indicates the material, and Temperature is the temperature. Note that the T = indicator is not required (although it is accepted) as temperature is the only possible independent variable that can be used to fix the state. An example function call that shows the proper format of the function appears in the Examples box at the bottom of the Function Information dialog, as seen in Figure 4-20. The argument Temperature can be a numerical value, an EES variable, or an algebraic expression that evaluates to the desired temperature in the selected temperature scale.

The following example provides the density, specific heat capacity, viscosity and thermal conductivity of a molten salt that consists of 60% sodium nitrate and 40% potassium nitrate (substance 'Salt (60% NaNO3, 40% KNO3)') at T = 550ºC.

```
$UnitSystem SI Mass K Pa J Mass
```

```
T=ConvertTemp(C,K,550 [C])          "temperature"
S$='Salt (60% NaNO3, 40% KNO3)'     "molten salt"
rho=rho_(S$, T)                     "density"
c=c_(S$, T)                         "specific heat capacity"
mu=mu_(S$,T)                        "viscosity"
k=k_(S$, T)                         "thermal conductivity"
```

The result is ρ = 1738 kg/m^3, c = 1543 J/kg-K, μ = 0.001188 Pa-s, and k = 0.5487 W/m-K.

Solid/Liquid Property Tables

The Solid/Liquid property functions are not truly built into EES but rather implemented in a user-accessible internal function library file named Solid-Liquid_Props.lib. The actual property data are stored in EES Lookup files having an .lkt filename extension. The library and data files are in located in the ../UserLib/EES_System/Solid-Liquid_Props folder. (If the Solid-Liquid_Props folder is renamed or moved out of the UserLib\EES_System folder then the Solid/Liquid properties button will be disabled.) A separate lookup (.lkt) table is provided for every substance. The library file uses the Interpolate command discussed in Section 2.3 to determine the property data values by interpolation from the data.

You can view, change, or add to the data that are used as the basis for the Solid/Liquid property routines by opening the associated Lookup table. For example, to examine the data for the molten salt 'Salt (60% NaNO3, 40% KNO3)' select Open Lookup Table from the Tables menu, navigate to the ../UserLib/EES_System/Solid-Liquid_Props/Molten Salts folder and select the file Salt (60% NaNO3, 40% KNO3).lkt. The Lookup Table is shown in Figure 4-21.

Lookup Table

Salt (60% NaNO3, 40% KNO3)

	1 T [K]	2 ρ [kg/m3]	3 C [kJ/kg-K]	4 μ [kg/m-s]	5 k [W/m-K]
Row 1	533.2	1924	1.491	0.004343	0.4922
Row 2	561.2	1906	1.499	0.003558	0.4976
Row 3	589.2	1888	1.503	0.002929	0.503
Row 4	616.2	1870	1.507	0.002437	0.5084
Row 5	644.2	1853	1.511	0.002062	0.5138
Row 6	672.2	1835	1.516	0.001786	0.5193
Row 7	700.2	1817	1.52	0.00159	0.5247
Row 8	727.2	1799	1.524	0.001454	0.5301
Row 9	755.2	1781	1.532	0.001361	0.5355
Row 10	783.2	1764	1.537	0.00129	0.541
Row 11	811.2	1746	1.541	0.001223	0.5464
Row 12	839.2	1728	1.545	0.001142	0.5518
Row 13	866.2	1710	1.549	0.001026	0.5572

Figure 4-21: The Lookup Table containing the data for the substance Salt (60% NaNO3, 40% KNO3).

Adding Solid/Liquid Property Data

You can add additional property data for a new substance by creating a new Lookup Table for that substance and copying it to the folder ../UserLib/EES_System/Solid-Liquid_Props or to a sub-folder within this folder. Optionally, add the name of the new substance to one of the text files contained in the folder (Building Materials.txt, Fluids.txt, Insulation.txt, Liquid Metals.txt, Miscellaneous.txt, or Molten Salts.txt). As an example, we will add properties for the metal manganin (an alloy of copper, manganese, and nickel). The specific heat capacity, thermal conductivity, and electrical resistivity at various temperatures are listed in Table 4-12.

Table 4-12: Properties of manganin at various temperatures.

Temperature (K)	Specific heat capacity (J/g-K)	Conductivity (W/cm-K)	Electrical Resistivity (μohm-m)
4	0.000246	0.00484	41.6
5	0.000313	0.00661	41.7
7	0.000524	0.0105	41.8
10	0.00112	0.0170	41.9
15	0.00319	0.0286	42.2
20	0.00742	0.0410	42.5
30	0.0258	0.0660	42.8
50	0.0930	0.101	43.7
70	0.171	0.120	44.3
90	0.234	0.135	44.8
120	0.279	0.148	45.5
150	0.328	0.157	46.1
180	0.354	0.166	46.6
210	0.370	0.176	47.0
250	0.385	0.195	47.3
300	0.400	0.220	47.5

The specific heat capacity and conductivity data as a function of temperature are shown in Figure 4-22.

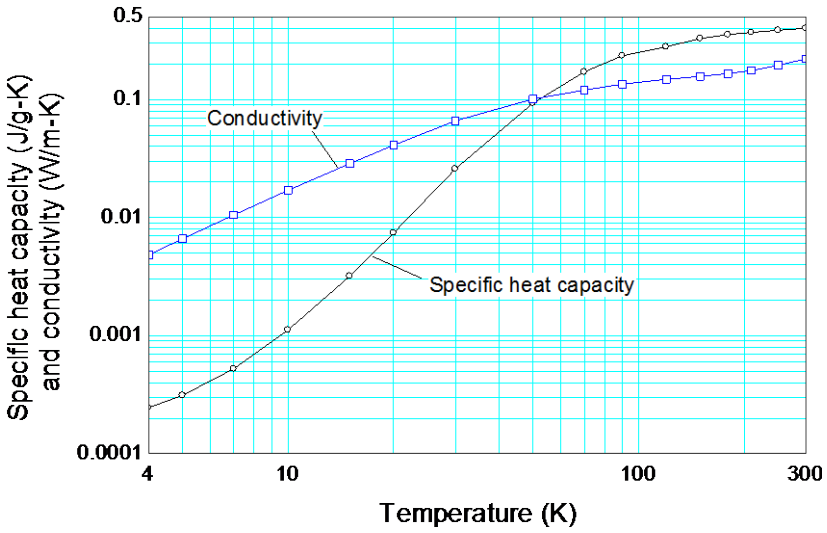


Figure 4-22: Specific heat capacity and conductivity of manganin as a function of temperature.

In order to have manganin appearing when the Metals is selected from the dropdown selection criteria, you must open the file Metals.txt in the Solid-Liquid_Props folder (using any text file editor) and add manganin to the bottom of the file, as shown in Figure 4-23.

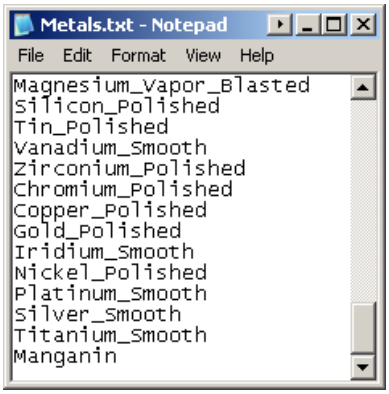


Figure 4-23: Adding Manganin to the Metals.txt file.

Next, generate a Lookup Table with one column for temperature (labeled T with units K) and a separate column for each property that is included; the column names must be the same as the associated function name (from Table 4-11) but without the final underscore. The names and units for each column must correspond to those shown in Table 4-13. Note that EES correctly converts units to the current unit system settings when returning a value from the Solid/Liquid properties; however, it assumes that the data are provided with the units shown in Table 4-13. Also note that not all properties must be provided for each substance (and the properties that are provided do not need to be provided at all of the temperatures that are included in the table).

Table 4-13: Column name and units for each property being added to a Solid/Liquid property table.

Property	Column Name	Unit
thermal conductivity	k	W/m-K
density	rho	kg/m^3
specific heat capacity	c	kJ/kg-K
viscosity	mu	Pa-s
volumetric thermal expansion coefficient	beta	1/K
modulus of elasticity	E	GPa
Poisson's ratio	nu	-
linear thermal expansion coefficient	alpha	1/K
vapor pressure	Pv	kPa
linear expansion[1]	DELTAL\L_293	-
emissivity	epsilon	-

1. Linear expansion refers to the change in length of a sample normalized by its length at 20ºC and corresponds to the linear coefficient of thermal expansion integrated from 20ºC to the temperature of interest.

The Lookup Table required to add the specific heat capacity and conductivity of manganin is shown in Figure 4-24.

Paste Special	1 T [K]	2 c [kJ/kg-K]	3 k [W/m-K]
Row 1	4	0.000246	0.484
Row 2	5	0.000313	0.661
Row 3	7	0.000524	1.05
Row 4	10	0.00112	1.7
Row 5	15	0.00319	2.86
Row 6	20	0.00742	4.1
Row 7	30	0.0258	6.6
Row 8	50	0.093	10.1
Row 9	70	0.171	12
Row 10	90	0.234	13.5
Row 11	120	0.279	14.8
Row 12	150	0.328	15.7
Row 13	180	0.354	16.6
Row 14	210	0.37	17.6
Row 15	250	0.385	19.5
Row 16	300	0.4	22

Figure 4-24: Lookup Table required to add the substance manganin.

Save the Lookup Table in EES Lookup File format (i.e., as a .lkt file) in the Solid-Liquid_Props folder. Make sure that the name of the table corresponds to the name of the substance, as entered in the *.txt file. For this example, the table should be saved as Manganin.lkt.

Start a new version of EES and you should find that the substance Manganin has been added to the Solid/Liquid library. Select Function Info from the Options menu and then select Solid/liquid properties. Navigate to the function k_ (or c_) and then select either All data (or Metals) and you will see that the substance Manganin appears in the list, as shown in Figure 4-25.

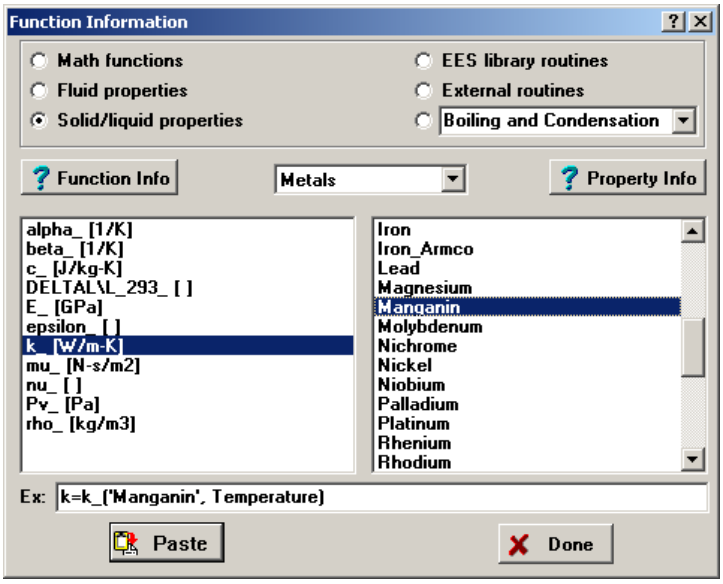


Figure 4-25: Function Information dialog showing the new substance Manganin.

Updating EES will not cause the new lookup file Manganin.lkt to be erased and therefore the substance Manganin will remain in the Solid/liquid database. However, if you make changes to or add data to the tables for existing substances then this information will be overwritten when you update EES. For example, you may have more refined or more accurate data for the molten metal substance Salt (60% NaNO3, 40% KNO3) and therefore it would be natural to make changes to the lookup table shown in Figure 4-21. However, this information will be lost during future updates of EES as the table Salt (60% NaNO3, 40% KNO3).lkt will be overwritten during the installation process. It is recommended that you make a backup copy of any Solid/Liquid property lookup table that you modify.

Adding Solid/Liquid Properties

You may want to add additional properties for either existing or new substances. For example, the data for manganin shown in Table 4-12 include electrical resistivity, which is not a built-in property in the Solid/Liquid database. Add the name of the new property function and the SI and English units associated with the property to the text file Solid-Liquid_Props.txt. This change is shown in Figure 4-26 for the property electrical resistivity, which is given the function name rho_e_.

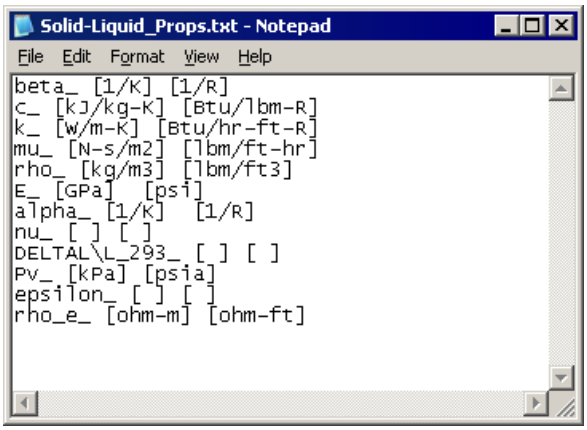


Figure 4-26: Adding the property function rho_e_ to the Solid-Liquid_Props.txt file.

The Lookup Table for any substance that includes electrical resistivity must have a column name corresponding to the function name without the final underscore. For manganin, the Lookup Table Manganin.lkt must be modified to appear as shown in Figure 4-27 and then saved in the Solid-Liquid_Props folder. Finally, the Solid-Liquid_Props.lib library file must be modified. Libraries are a collection of EES functions that can be automatically loaded and accessed from within EES programs, as discussed in Chapter 11. The Solid-Liquid_Props.lib file includes the collection of functions that implement the functions k_, c_, etc. by interpolating the data contained in the lookup tables.

Lookup Table

Manganin

	1 T [K]	2 c [kJ/kg-K]	3 k [W/m-K]	4 ρ_e [Ω-m]
Row 1	4	0.000246	0.484	4.160E-07
Row 2	5	0.000313	0.661	4.170E-07
Row 3	7	0.000524	1.05	4.180E-07
Row 4	10	0.00112	1.7	4.190E-07
Row 5	15	0.00319	2.86	4.220E-07
Row 6	20	0.00742	4.1	4.250E-07
Row 7	30	0.0258	6.6	4.280E-07
Row 8	50	0.093	10.1	4.370E-07
Row 9	70	0.171	12	4.430E-07
Row 10	90	0.234	13.5	4.480E-07
Row 11	120	0.279	14.8	4.550E-07
Row 12	150	0.328	15.7	4.610E-07
Row 13	180	0.354	16.6	4.660E-07
Row 14	210	0.37	17.6	4.700E-07
Row 15	250	0.385	19.5	4.730E-07

Figure 4-27: The Lookup Table Manganin.lkt modified to include electrical resistivity data.

The easiest way to add the new function to the library file (rho_e_) is to copy an existing function within the library file (e.g., k_), understand its functionality, and then modify it appropriately. The function k_ in the library file appears below:

```
Function k_(Substance$, T)
    File$=substance$
    LastRow=NLookupRows(File$)
    T_max=Lookup(File$, LastRow,'T')
    T_min=Lookup(File$, 1,'T')

    Call CheckT_(T_min, T_max, T : T_K, T$, U$, High$, Low$)

    If (T_K>T_max) or (T_K<T_min) then
        a$=Concat$('The temperature supplied to the thermal conductivity lookup table for ', substance$)
        b$=concat$(a$, ' is XXXF0' )
        c$=Concat$(b$, U$)
        d$=Concat$(c$, '.  The allowable range is ')
        e$= Concat$(d$, Low$)
        f$=Concat$(e$, ' to ')
        g$=Concat$(f$, High$)
        h$=Concat$(g$, U$)
        Call warning(h$, T)
    EndIf
    if (UnitSystem('SI')=1) then kU$='W/m-K' else kU$='Btu/hr-ft-R'
    k_=interpolate1(File$,'T', 'k', T=T_K)*&
        (UnitSystem('SI')+UnitSystem('Eng')*Convert(W/m-K, Btu/hr-ft-F))
        k_:=k_ "[kU$]"
end thermal conductivity
```

The name of the function must be changed from k_ to rho_e_ (the modified code is shown in bold and red below):

```
Function rho_e_(Substance$, T)
```

The next four lines determine the name of the lookup table file (based on the substance), the number of rows and the range of temperatures; these lines can not be altered.

```
    File$=substance$
    LastRow=NLookupRows(File$)
    T_max=Lookup(File$, LastRow,'T')
    T_min=Lookup(File$, 1,'T')
```

The next set of lines determines whether the temperature provided to the function is within the range of temperatures that are contained in the lookup table and, if not, calls the procedure Warning with the appropriate string. The procedure Warning is discussed in Section 3.7. The only part of this code that should be modified is the details of the warning string.

```
    Call CheckT_(T_min, T_max, T : T_K, T$, U$, High$, Low$)

    If (T_K>T_max) or (T_K<T_min) then
      a$=Concat$('The temperature supplied to the electrical resistivity lookup table for ', substance$)
      b$=concat$(a$, ' is XXXF0' )
      c$=Concat$(b$, U$)
      d$=Concat$(c$, '.  The allowable range is ')
      e$= Concat$(d$, Low$)
      f$=Concat$(e$, ' to ')
      g$=Concat$(f$, High$)
      h$=Concat$(g$, U$)
      Call warning(h$, T)
    EndIf
```

The next line sets the units of the output based on the current unit settings using the UnitSystem function (see Section 3.5):

```
    if (UnitSystem('SI')=1) then kU$='ohm-m' else kU$='ohm-ft'
```

The next line carries out the interpolation. The name of the column must be changed and the unit conversion must be adjusted. The changes are indicated in bold font.

```
    rho_e_=interpolate1(File$,'T', 'rho_e', T=T_K)*(UnitSystem('SI')+UnitSystem('Eng')*&
      Convert(ohm-m, ohm-ft))
    rho_e_:=rho_e_ "[kU$]"

end electrical resistivity
```

Restart EES and select Function Information from the Options menu. Select Solid/liquid properties and you should see that electrical resistivity (rho_e_) is now listed as an available

function. Select rho_e_ and you will see that Manganin is the only material that can be used with this function, as shown in Figure 4-28.

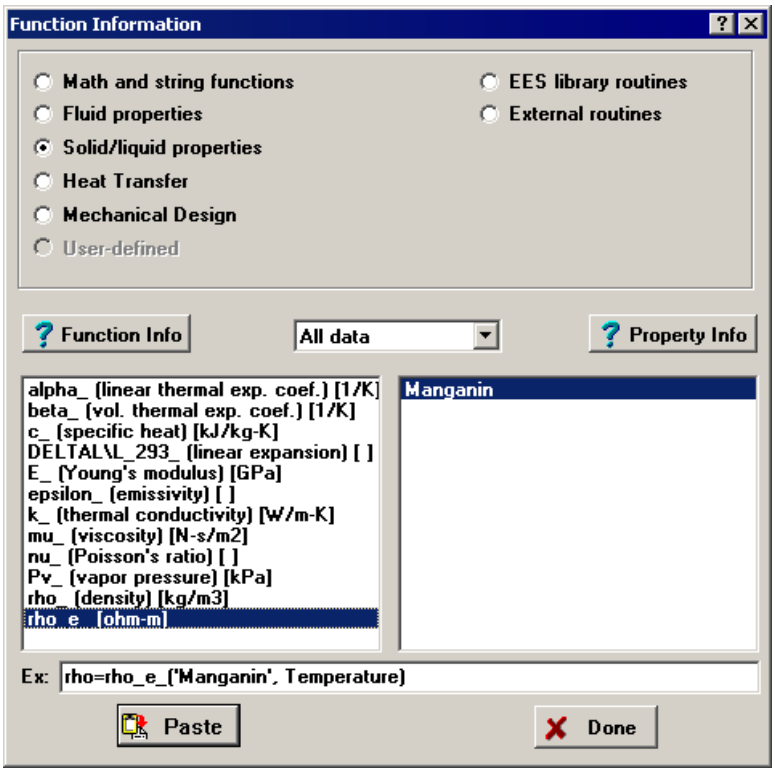


Figure 4-28: Function Information dialog showing the added Solid/liquid function rho_e_.

4.9 Property Data in External Procedures

Almost all of the property data that have been described in this chapter to this point can be considered to be built into EES. Property data can also be provided using external functions and procedures. External functions and procedures are programs that have been written and compiled independent of EES, as described in Chapter 19. When properly formulated, EES can interface with these external programs. This section describes several useful external procedures that have been developed and are available to provide property information to EES.

The NASA Library

The NASA Database is an EES external procedure that determines the specific heat capacity, specific enthalpy, and specific entropy as a function of temperature for more than 2000 ideal gases and condensed substances. The NASA program is provided with all versions of EES. The properties are determined by correlations and statistical thermodynamics by McBride et al. (2002). Note that the property information for the ideal gases has been incorporated into the EES property database, as described in Section 4.4. Consequently, it is not necessary to call the NASA external procedure in order to obtain property information for the ideal gases in the database, although it is permitted to do so. However, calling the NASA external procedure is the only way to obtain the property information for the condensed substances in the database. Lists

of the names of the ideal gas and condensed substances that the NASA external program will recognize can be viewed by selecting Function Info from the Options menu and then selecting the External routines button, as shown in Figure 4-29. Select the NASA program from the list, as shown, and then click the Function Info button.

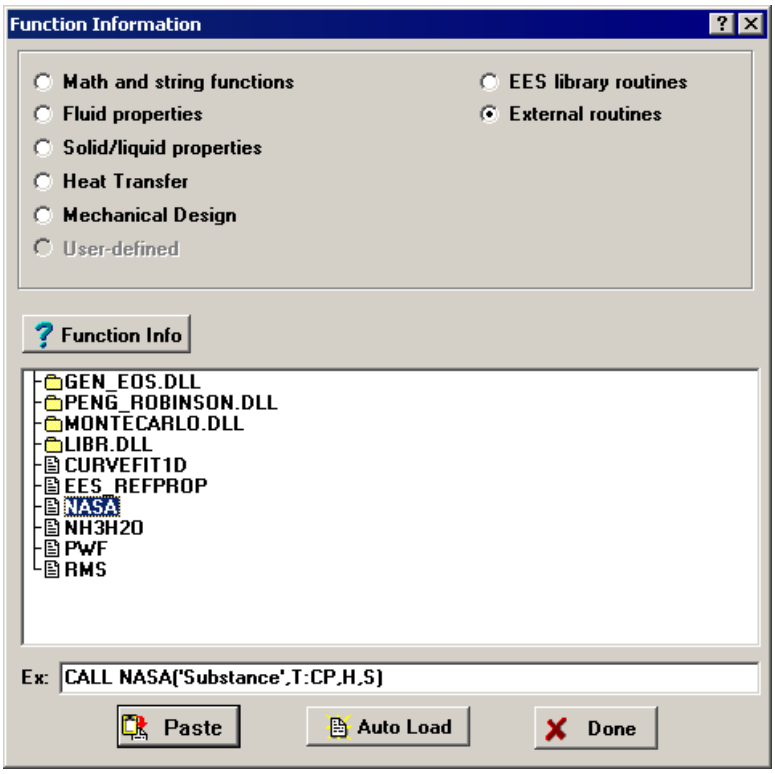


Figure 4-29: Function Information dialog window configured to show External routines.

The NASA external procedure is accessed from EES with the following calling format:

```
Call NASA(Substance$, T: cP, h, s)
```

The string Substance$ is the name of the ideal gas or condensed substance; the name must be either enclosed in single quotes (e.g., 'CO2') or defined with an EES string variable. The input variable T is the temperature, which must be provided in units K, regardless of the unit system settings in EES. The output variables cP, h, and s are the specific heat capacity (in kJ/kmol-K), specific enthalpy (in kJ/kmol), and specific entropy (in kJ/kmol-K) at the given temperature and a pressure of 1.0 bar. The specific enthalpy is referenced to zero for stable elements at 25°C (298.15 K) and the specific entropy is referenced to zero according to the Third Law of Thermodynamics. This choice of reference makes it easy to carry out calculations related to chemical reactions, as discussed in Section 4.4. Note that the variables cP, h, and s are returned in the units indicated above, regardless of the unit settings in EES.

Ideal gas carbon dioxide (CO2) is a built-in ideal gas. We can use the NASA external procedure to compare the property values built into EES with the values obtained from the NASA external procedure at a temperature of 1000 K:

```
$UnitSystem SI K kPa kJ Molar

G$='CO2'                                  "name of the gas in a string variable"
T=1000 [K]                                "temperature"
cP_EES=CP(G$,T=T)                         "specific heat capacity from EES built-in properties"
h_EES=enthalpy(G$,T=T)                    "specific enthalpy from EES built-in properties"
s_EES=entropy(G$,T=T,P=100 [kPa])         "specific entropy at 1 bar from EES built-in properties"

Call NASA(G$,T: cP_NASA, h_NASA, s_NASA)  "property information from the NASA program"
```

Note that the $UnitSystem directive sets EES to operate in the same unit system as the NASA external program so that the values obtained from EES and the NASA database can be directly compared. The Solution Window is shown in Figure 4-30 and indicates that the values differ, but by only a small amount. The NASA value is likely to be more accurate.

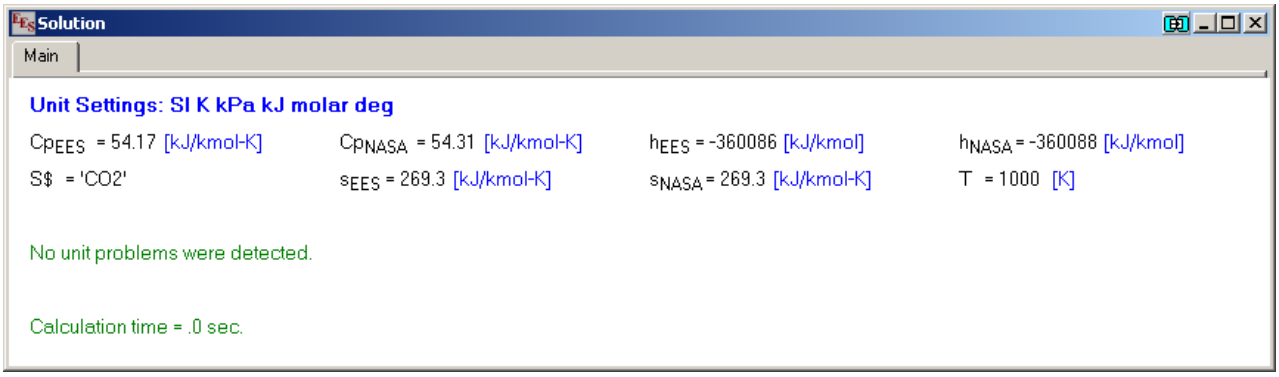


Figure 4-30: Solution Window for the NASA external procedure example.

The NASA external procedure is accessed for condensed substances in the same manner. For example, the thermodynamic properties of solid carbon, which is named 'C(gr)' in the NASA database, is found at 298.15 K using the following line of code.

```
Call NASA('C(gr)',298.15: c_C,h_C,s_C)    "property information for solid carbon"
```

As expected, the specific enthalpy of solid carbon at 298.15 K is 0 kJ/kmol because solid carbon is a stable element and 298.15 K is the standard reference temperature. The specific entropy at these conditions is 5.734 kJ/kmol-K.

Ammonia-Water Properties

Mixtures of ammonia and water are the working fluids in some power and refrigeration cycles. Approximate property data for ammonia-water mixtures based on correlations developed by Ibrahim and Klein (1993) are provided by the NH3H2O external procedure. The NH3H2O external procedure resides in the \Userlib\EES_System directory and it is provided with all versions of EES. Information on the use of the NH3H2O external procedure can be obtained from the Function Information dialog shown in Figure 4-29 by clicking on NH3H2O. Newer and more accurate properties for ammonia-water mixtures are available in the REFPROP program distributed as Standard Reference Database 23 by the National Institute of Standards and Technology (NIST). Information about an interface that allows the REFPROP program to be called from EES is provided in a subsequent section.

The NH3H2O external Procedure has the following calling format.

```
Call NH3H2O(Code, In1, In2, In3: T, P, x, h, s, u, v, q)
```

There are eight thermodynamic properties that are considered by the procedure NH3H2O. These properties are numbered and described in Table 4-14. The units of each property are also listed.

Table 4-14: Ammonia-water property functions and their units.

Property Number	Property Name	Property	Units
1	T	mixture temperature	K
2	P	mixture pressure	bar
3	x	ammonia mass fraction	-
4	h	specific enthalpy of mixture	kJ/kg
5	s	specific entropy of mixture	kJ/kg-K
6	u	specific internal energy of mixture	kJ/kg
7	v	specific volume of mixture	m^3/kg
8	q	vapor mass fraction	-

An ammonia-water mixture requires three properties to fix the state. The first input, Code, is a three digit integer indicating which three of the eight properties are provided as input values. The numerical values assigned to the properties are shown in Table 4-14 For example, a value 123 indicates that the values of properties 1 (temperature), 2 (pressure), and 3 (ammonia mass fraction) are provided by the inputs In1, In2, and In3, respectively. All eight properties are provided in the outputs; therefore, three of these outputs will have the same values as the corresponding three inputs, depending on the value of the input Code.

The inputs and outputs for the NH3H2O routine must be in the units designated in Table 4-14, regardless of the unit setting made in EES. The ammonia mass fraction lies between 0 and 1 for saturated states. Subcooled states are indicated with a mass fraction of x = -0.001 and superheated states with x = 1.001.

The NH3H2O procedure supports Code values of 123, 128, 137, 138, 148, 158, 168, 178, 234, 235, 238, 248, 258, 268, and 278. However, one (or more) output values can also be fixed in which case EES will attempt to determine the corresponding input values in an iterative manner

provided that the procedure NH3H2O is called from the Equations Window or a subprogram. In this way, EES can be used to determine the properties of ammonia-water given any three independent properties.

It is often most convenient to place the NH3H2O call within a function or procedure in the EES Equations Window, as demonstrated in the following example. Placing the call to NH3H2O in a procedure eliminates any problem that would otherwise be involved with using the same variable name for both an input and output in the Equations Window. For example, the procedure below returns the specific enthalpy and specific entropy given the temperature, pressure, and mass fraction of ammonia.

```
$UnitSystem SI Mass Rad kJ K bar
$TabStops 0.3 0.6 3

Procedure hs(T, P, x: h, s)
    call NH3H2O(123, T, P, x: T, P, x, h, s, u, v, q)
end
```

Note that within the procedure it is fine that three of the output variables (T, P, and x) have the same name as the three input variables. This would not be possible in the main body of the Equations Window. The following code uses the subroutine hs to determine the specific enthalpy and specific entropy of a 50% mass fraction ammonia-water mixture at 450 K and 10 bar:

```
T=450 [K]
P=10 [bar]
x=0.50 [-]
Call hs(T,P,x: h,s)
```

which leads to h = 2224 kJ/kg and s = 6.311 kJ/kg-K.

Sea Water Properties

An extensive library of property information for the properties of sea water has been developed by John Lienhard and his co-workers at Massachusetts Institute of Technology (MIT) and implemented in the library Seawater_EES.lib. This library provides many seawater property functions written in EES code. The library is provided (with permission) with all versions of EES. Information on how to use the functions and procedures in this library can be obtained from the Function Information dialog window by clicking on the EES library routines button and then on the Seawater_EES.lib list item as shown in Figure 4-31.

The library provides functions for the thermophysical properties of sea water that are listed in Table 4-15 as a function of the temperature (T, in ºC) and the salinity (S, in g/kg). More information on these properties can be found at http://web.mit.edu/seawater/.

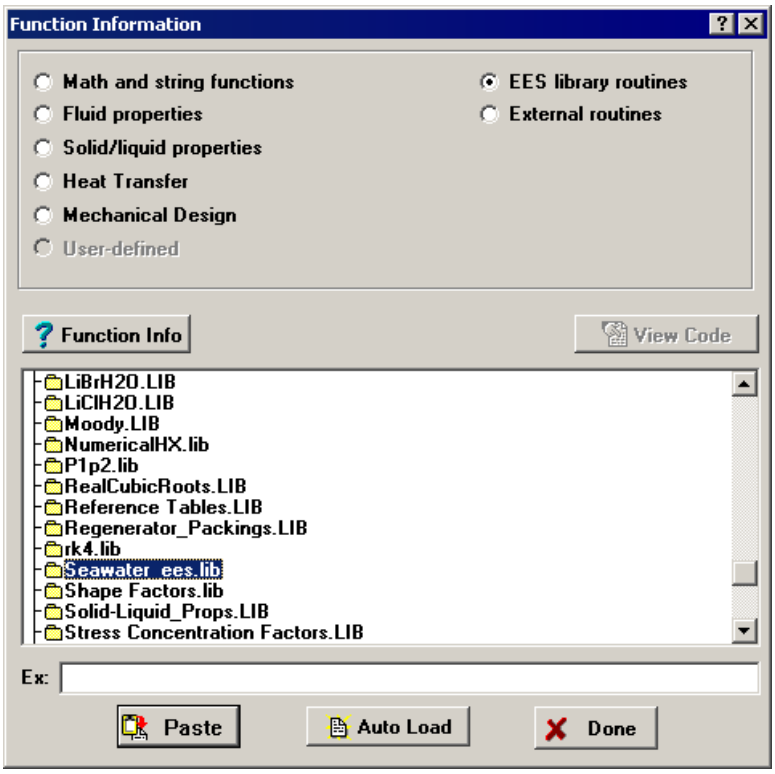


Figure 4-31: Function Information dialog window configured to show EES library routines.

Table 4-15: Sea water property functions, their units, and their range of applicability.

Function name	Returns	Units	Range of Temperature (°C)	Range of Salinity (g/kg)
SW_Density(T, S)	density	kg/m^3	0 - 180	0 - 160
SW_SpcHeat(T, S)	specific heat capacity	J/kg.K	0 - 180	0 - 180
SW_Conductivity(T, S)	thermal conductivity	W/m	0 - 180	0 - 160
SW_Viscosity(T, S)	dynamic viscosity	kg/m-s	0 - 180	0 - 150
SW_SurfaceTension(T, S)	surface tension	N/m	0 - 40	0 - 40
SW_Psat(T, S)	vapor pressure	kPa	0 - 200	0 - 240
SW_BPE(T, S)	boiling point elevation	K	0 - 200	0 - 120
SW_LatentHeat(T, S)	latent heat of vaporization	J/kg	0 - 200	0 - 240
SW_Enthalpy(T, S)	specific enthalpy	J/kg	10 - 120	0 - 120
SW_Entropy(T, S)	specific entropy	J/kg-K	10 - 120	0 - 120
SW_Gibbs(T, S)	specific Gibbs energy	J/kg	10 - 120	0 - 120
SW_Exergy(T, S, T0, S0)[1]	specific flow exergy	J/kg	10 - 120	0 - 120
SW_Osmotic(T, S)	osmotic coefficient	-	0 - 200	10 - 120
ChemPot_W(T, S)	chemical potential of water	J/kg	10 - 120	0 - 120
ChemPot_S(T, S)	chemical potential of salts	J/kg	10 - 120	0 - 120
SW_Diffusivity(T, S)	thermal diffusivity	m^2/s	0 - 180	0 - 160
SW_IntEnergy(T, S)	specific internal energy	J/kg	10 - 120	0 - 120
SW_KViscosity(T, S)	kinematic viscosity	m^2/s	0 - 180	0 - 150
SW_Density(T, S)	Prandtl number	-	0 - 180	0 - 150
SW_SpcHeat(T, S)	specific volume	m^3/kg	0 - 180	0 - 160

1. Note that the variables T0 and S0 correspond to the dead state temperature and salinity, respectively.

Lithium Bromide-Water and Lithium Chloride-Water Properties

Solutions composed of water and either lithium bromide or lithium chloride salts have desiccant properties and are used for drying and cooling air. The properties of these solutions depend on the solution temperature and the salt concentration. The property correlations for lithium bromide-water solutions that have been published by Patek and Klomfar (2006) are implemented in the LiBrH2O.lib EES library file. Property correlations for lithium chloride-water solutions have also been developed by Patek and Klomfar (2008) and these are implemented in the LiClH2O.lib EES library file. Both of these libraries are provided with all versions of EES. Information about these libraries can be obtained from the Function Information dialog window by clicking the EES library routines button at the top right of the screen, as shown in Figure 4-31. Next select the library file by clicking on its name and then click the Function Info button.

The property functions for lithium bromide-water solutions and the possible units for the properties that are returned are listed in Table 4-16. All of the property information provided to and returned by these functions must be in the selected unit system, as described in Section 4.1. Most of these property functions require temperature (T) and salt concentration (x) as input arguments. The salt concentration is taken to be the mass fraction if EES is configured to operate with specific properties expressed on a per unit mass bases. If the EES unit system is set to a molar basis, the salt concentration is taken to be the mole fraction.

Table 4-16: Property functions for lithium bromide-water solutions and their units.

Function Name	Returns	SI Units	English Units
Cond_LiBrH2O(T,x)	thermal conductivity	W/m-K	Btu/hr-ft-R
Cp_LiBrH2O(T,x)	specific heat capacity	J/kmol-K, kJ/kmol-K, J/kg-K, kJ/kmol-K	Btu/lbmol-R Btu/lbm-R
h_LiBrH2O(T,x)	specific enthalpy	J/kg, kJ/kg, J/kmol, kJ/kmol	Btu/lbm Btu/lbmol
massfraction_LiBrH2O(x)	mole fraction given mass fraction	none	none
molefraction_LiBrH2O(w)	mass fraction given mole fraction	none	none
P_LiBrH2O(T, x)	pressure	Pa, kPa, bar, MPa	psia, atm
Q_LiBrH2O(h,P,z :Q,T, x)	quality, temperature and composition	°C, K for T	°F, R for T
s_LiBrH2O(T,x)	specific entropy	J/kmol-K, kJ/kmol-K, J/kg-K, kJ/kmol-K	Btu/lbmol-R Btu/lbm-R
T_LiBrH2O(P, x)	temperature	°C, K	°F, R
x_LiBrH2O(T, P)	equilibrium composition	none	none
Visc_LiBrH2O(T, x)	viscosity	Pa-s	lb_m/ft-hr

As an example, the following code will return the specific heat capacity and viscosity of a 50% mass fraction lithium bromide-water solution at 50°C:

```
$UnitSystem SI C kPa kJ mass

T=50 [C]                          "temperature"
x=0.5 [-]                         "mass fraction"
C=Cp_LiBrH2O(T,x)                 "specific heat"
mu=Visc_LiBrH2O(T,x)              "viscosity"
```

Solving provides C = 2.183 kJ/kg-K and μ = 0.002095 Pa-s. The property correlations for lithium chloride-water solutions have the same form and units as those for lithium bromide-water solutions, with Br in each function replaced by Cl. However, thermal conductivity and viscosity functions for lithium chloride solutions are currently not provided.

The GENEOS Library

The GENEOS library uses a modification of the Redlich-Kwong equation of state proposed by Soave (1972) to provide approximate thermodynamic property information for any fluid for which critical properties are available. This library is included with all versions of EES. Information about the GENEOS library can be accessed using the Function Information dialog window by clicking on the External Routines button and then the GEN_EOS folder, as shown in Figure 4-29. The library consists of the four functions that are listed in Table 4-17. All of these functions require the reduced temperature (Tr) and reduced pressure (Pr) as inputs. Reduced temperature is the ratio of the temperature to the critical temperature and reduced pressure is the ratio of pressure to the critical pressure. The acentric factor (w) is an optional input to the functions in the GENEOS library. If the acentric factor is not provided, then a default value will be used; however, the accuracy of the results is likely to improve if the acentric factor for the fluid of interest is provided. The definition of the acentric factor, as well as more details about the use of the GENEOS library functions can be found in Klein and Nellis (2012).

Table 4-17: Functions in the GENEOS library.

Function Name	Returns	Units
Compress(Tr, Pr, w)	compressibility factor	none
EnthDep(Tr, Pr, w)	enthalpy departure	none
EntrDep(Tr, Pr, w)	entropy departure	none
FugCoef(Tr, Pr, w)	fugacity coefficient	none

The Peng-Robinson Library

The Peng-Robinson equation of state (1976) is widely used for calculating the thermodynamic properties of both pure fluids and fluid mixtures. The Peng-Robinson equation of state provides reasonably accurate estimates of liquid and vapor phase densities. These data can be used to calculate enthalpy and entropy departures. Although the Peng-Robinson equation of state is not as accurate the Fundamental Equation of State or Martin-Hou formulations used in the EES real fluid property data base, the Peng-Robinson equation offers generality since it requires as inputs only the critical temperature, the critical pressure, the acentric factor and the specific heat in the ideal gas state in order to provide estimates of the thermodynamics properties for any pure fluid. The Peng-Robinson equations can also be used for mixtures of different fluids.

The Peng-Robinson library is included with all versions of EES. Information on the functions in the Peng-Robinson library can be viewed from the Function Information dialog window by clicking on Peng_Robinson.DLL folder shown near the top of the list in Figure 4-29. Information on how to apply the Peng-Robinson library functions is provided by Klein and Nellis (2012).

The EES_REFPROP Interface

The National Institute of Standards and Technology (NIST) has developed the program REFPROP (which is also referred to as NIST Standard Reference Database 23). Information about the REFPROP database can be found at http://www.nist.gov/srd/nist23.cfm. The REFPROP program provides high accuracy property data for pure refrigerants and refrigerant mixtures as well as many other fluids and mixtures. REFPROP uses the most advanced methods for estimating property data and it provides the most accurate property information publicly available. Transport properties as well as thermodynamic properties are provided in all fluid regimes, including compressed liquid and near the critical state.

The EES_REFPROP interface has been developed for REFPROP versions 6 and newer in order to allow the NIST property database to be used with the equation-solving and other features of EES in a manner that is similar to the use of the EES built-in properties. Note that EES provides built-in high accuracy property data for many pure fluids, so this interface should only be of interest when property data for the pure fluid or mixture of interest are not available in EES. The EES_REFPROP interface is an extra cost option. Information about this interface can be found on the F-Chart Software website at http://fchart.com/ees/ees-refprop.php.

4.10 Adding Property Information

It is possible to add property information to EES in several ways. The most direct method is to provide tabular information and then interpolate these data using the methods described in Sections 2.3 and 2.4. Alternatively, the coefficients required to implement the property correlations associated with ideal gas and real fluids using the Martin-Hou equation of state can be entered with external files. These alternative ways for providing property information are described in this section. Fluids represented by the higher accuracy Fundamental Equation of State cannot be added by the user.

Providing Data in Lookup Tables

Property data are often available in tabular form. Tabular data can be entered into EES as Lookup tables or Lookup files. (See Section 1.8 for information on creating and using Lookup tables.) These property data can be interpolated in order to provide the property information needed in the EES equations. The most direct method of using these data is with the Interpolate (for 1-D) or Interpolate2D (for 2-D) functions described in Sections 2.3 and 2.4.

Providing Ideal Gas Property Data

To add ideal gas property information, the user must supply the necessary parameters for the thermodynamic and transport property correlations. The parameters are placed in a text file that must be located in the \UserLib subdirectory in the EES folder if it is to be automatically loaded when EES is started. Ideal gases that are entered in this manner will provide most of the same functional capability as the built-in ideal gases in EES.

Ideal gas property files must have an .idg filename extension. An equation of state is not needed since it is assumed that the fluid obeys the ideal gas equation of state. Correlations are required for the specific heat capacity at constant pressure, viscosity, and thermal conductivity as a function of temperature. Note that particular attention must be paid to the reference states if the gas is to be used in calculations involving chemical reactions. Therefore, the enthalpy of formation and the Third-Law entropy values at 298.2 K and 1 bar (or 1 atm) must be supplied.

The format of an ideal gas property file is listed below.

```
Name          {Name of ideal gas}
MW            {Molar mass of fluid}
Tn            {Tn, normalizing temperature in K}
T_low_cP      {Lower temperature limit of cP correlation in K}
T_high_cP     {Upper temperature limit of cP correlation in K}
a0 b0         {a0, b0  cP = sum(a[i]*(T/Tn)^b[i], i=0,9 in kJ/kmole-K}
a1 b1         {a1, b1}
a2 b2         {a2, b2}
a3 b3         {a3, b3}
a4 b4         {a4, b4}
a5 b5         {a5, b5}
a6 b6         {a6, b6}
a7 b7         {a7, b7}
a8 b8         {a8, b8}
a9 b9         {a9, b9}
T_ref         {T_ref in K}
P_ref         {P_ref in kPa}
hform         {hform - enthalpy of formation in kJ/kmol at T_ref}
s0            {s0 - Third law entropy in kJ/kmol-K at T_ref and P_ref}
0             {reserved - set to 0}
0             {reserved - set to 0}
T_low_visc    {Lower temperature limit of viscosity correlation in K}
T_high_visc   {Upper temperature limit of viscosity correlation in K}
v0            {v0  Viscosity = sum(v[i]*T^(i-1)) for i=0 to 5 in Pa-s}
v1            {v1}
v2            {v2}
v3            {v3}
v4            {v4}
v5            {v5}
T_low_k       {Lower temperature limit of thermal conductivity correlation in K}
T_high_k      {Upper temperature limit of thermal conductivity correlation in K}
t0            {t0  Thermal Conductivity = sum(t[i]*T^(i-1)) for i=0 to 5 in W/m-K}
t1            {t1}
t2            {t2}
t3            {t3}
t4            {t4}
t5            {t5}
0             {Terminator - set to 0}
```

The first line provides the name of the ideal gas in the variable Name. The second line provides the molar mass in the variable MW. The next lines provide the inputs for the correlation used to calculate the specific heat capacity at constant pressure. The form of the correlation is:

$$c_P(T) = \sum_{i=0}^{9} a_i \left(\frac{T}{T_n} \right)^{b_i} \tag{4-13}$$

where c_P is provided in kJ/kmol-K, T is the temperature (in K), and T_n is a normalizing temperature (also in K). The coefficients a_0 through a_9 and b_0 through b_9 are provided as well as a lower and upper bound on the range of validity for the correlation ($T_{low,cP}$ and $T_{upper,cP}$).

The next lines provide the reference temperature (T_{ref}) and pressure (P_{ref}) in K and kPa, respectively. The enthalpy of formation (h_{form}) and Third law entropy (s_0) in kJ/kmol and kJ/kmol-K, respectively, at the reference conditions must be provided. The remaining lines provide correlations for the viscosity and thermal conductivity. The form of these correlations are:

$$\mu(T) = \sum_{i=0}^{5} v_i T^{i-1} \tag{4-14}$$

and

$$k(T) = \sum_{i=0}^{5} t_i T^{i-1} \tag{4-15}$$

where μ is viscosity in Pa-s, k is thermal conductivity in W/m-K, and T is temperature in K. The coefficients v_0 through v_5 and t_0 through t_5 must be provided in addition to lower and upper limits on these correlations ($T_{low,visc}$, $T_{high,visc}$, $T_{low,k}$, and $T_{high,k}$).

An example file providing the parameters for ideal gas CO_2 is provided below. This file is, of course, not needed since EES provides built-in property data for carbon dioxide modeled as an ideal gas with the fluid CO2. The properties of a new ideal gas can be entered by editing this file appropriately. Note that the first row in the file causes the new gas to be named TestCO2; this is the name that will be recognized in EES.

```
TestCO2      {Name of ideal gas}
44.01        {Molar mass of fluid}
100.0        {Tn, normalizing temperature in K}
250          {Lower temperature limit of cP correlation in K}
1500         {Upper temperature limit of cP correlation in K}
-3.7357 0    {a0, b0  cP=sum(a[i]*(T/Tn)^b[i], i=0,9 in kJ/kmol-K}
30.529  0.5  {a1, b1}
-4.1034 1.0  {a2, b2}
0.02420 2.0  {a3, b3}
0 0          {a4, b4}
0 0          {a5, b5}
0 0          {a6, b6}
0 0          {a7, b7}
0 0          {a8, b8}
0 0          {a9, b9}
298.15       {T_ref in K}
100          {P_ref in kPa}
-393520      {hform - enthalpy of formation in kJ/kmol at T_ref}
213.685      {s0 - Third law entropy in kJ/kmol-K at T_ref and P_ref}
0            {reserved - set to 0}
0            {reserved - set to 0}
200          {Lower temperature limit of viscosity correlation in K}
1000         {Upper temperature limit of viscosity correlation in K}
```

```
-8.09519E-7   {v0  Viscosity = sum(v[i]*T^(i-1)) for i=0 to 5 in Pa-s}
6.039533E-8   {v1}
-2.8249E-11   {v2}
9.84378E-15   {v3}
-1.4732E-18   {v4}
0             {v5}
200           {Lower temperature limit of thermal conductivity correlation in K}
1000          {Upper temperature limit of thermal conductivity correlation in K}
-1.1582E-3    {t0  Thermal Conductivity = sum(t[i]*T^(i-1)) for i=0 to 5 in W/m-K}
3.9174E-5     {t1}
8.2396E-8     {t2}
-5.3105E-11   {t3}
3.1368E-16    {t4}
0             {t5}
0             {Terminator - set to 0}
```

Place the text file above in the \Userlib directory as TestCO2.idg and start EES. Select Function Info from the Options menu and select Ideal Gases. The gas TestCO2 is now available from the list of gases, as shown in Figure 4-32.

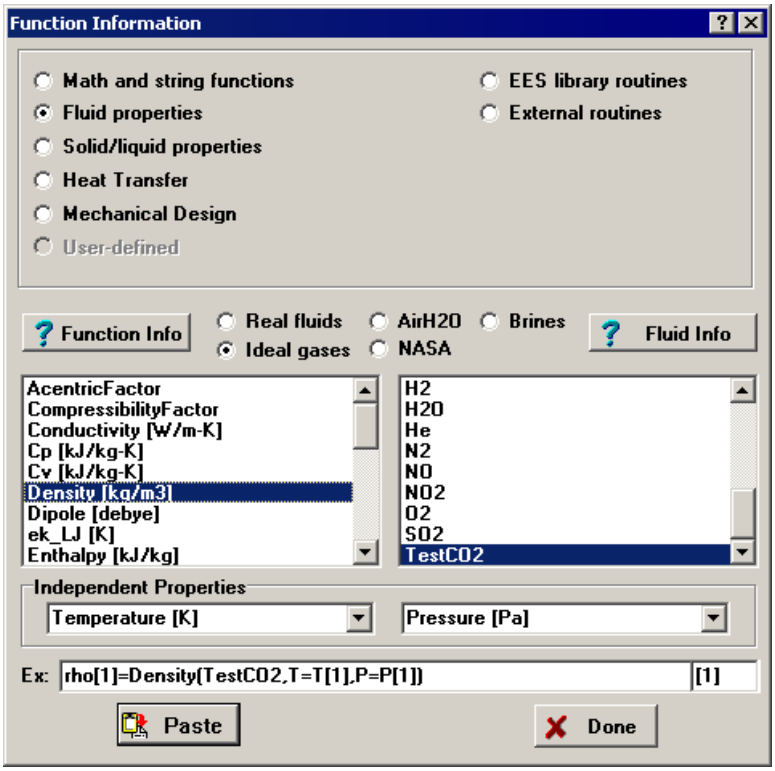


Figure 4-32: Function Information dialog showing the added ideal gas substance TestCO2.

Providing Real Fluid Property Data represented by the Martin-Hou Equation of State

Property information can be added for fluids that can be represented by the Martin-Hou (1955) equation of state. The input parameters for a real fluid are provided in a file that has an .mhe (for Martin-Hou Equation) filename extension. This file must be placed in the ..\UserLib subdirectory if it is to be automatically loaded when EES is started.

The format of Martin-Hou property file is listed below.

```
//Comment line 1
//Comment line 2
//additional comment lines as necessary
Name|Fluid Information message
MW              {molecular weight}
0               {not used}
ad              {ad Liquid density fit=ad+bd*Tz^(1/3)+cd*Tz^(2/3)+dd*Tz+ed*Tz^(4/3) }
bd              {bd         +fd*sqrt(Tz)+gd*(Tz)^2}
cd              {cd     where Tz=(1-T/Tc) and Liquid Density[=]lbm/ft3 }
dd              {dd}
ed              {ed}
fd              {fd}
gd              {gd}
ap              {ap  Vapor pressure fit: lnP=ap/T+bp+cp*T+dp*(1-T/Tc)^1.5+ep*T^2}
bp              {bp     where T[=] R and P[=]psia}
cp              {cp}
dp              {dp}
ep              {ep}
0               {not used}
R               {Gas constant in psia-ft3/lbm-R}
b               {b  Constants for Martin-Hou EOS/English_units
A2              {A2    where P[=] psia, T[=]R and v[=]ft^3/lbm}
B2              {B2}
C2              {C2}
A3              {A3}
B3              {B3}
C3              {C3}
A4              {A4}
B4              {B4}
C4              {C4}
A5              {A5}
B5              {B5}
C5              {C5}
A6              {A6}
B6              {B6}
C6              {C6}
Bexp            {Bexp - Martin-Hou exponential constant}
alpha           {alpha}
C'              {C'}
ac              {ac  Cv0 fit Cv(0 pressure) = ac + bc*T + cc*T^2 + dc*T^3 + ec/T^2 }
bc              {bc           where T[=]R and Cv[=]Btu/lbm-R }
cc              {cc}
dc              {dc}
ed              {ec}
href            {href offset set to 0 for sat'd liquid at -40F}
sref            {sref offset}
Pc              {Pc [=] psia}
Tc              {Tc [=] R}
vc              {vc [=] ft3/lbm}
0               {not used}
0               {not used}
VCT             {Viscosity correlation type: 2 poly gas&liq 2.1 poly gas/log liq}
Tlowgv          {Lower limit of gas viscosity correlation in K}
Thighgv         {Upper limit of gas viscosity correlation in K}
Agv             {Agv    GasViscosity*1E12=Agv+Bgv*T+Cgv*T^2+Dgv*T^3}
Bgv             {Bgv    where T[=]K and GasViscosity[=]N-s/m2 }
Cgv             {Cgv}
Dgv             {Dgv}
Tlowlv          {Lower limit of liquid viscosity correlation in K}
Thighgv         {Upper limit of liquid viscosity correlation in K}
Alv             {Alv   log10(Liquid Viscosity)=Alv+Blv/T+Clv*T+Dlv*T^2}
Blv             {Blv    where T[=]K and Liquid Viscosity[=]N-s/m2}
```

```
Clv             {Clv}
Dlv             {Dlv}
kCT             {Conductivity correlation type: set to 2 poly gas&liq: do not change}
Tlowgk          {Lower limit of gas conductivity correlation in K}
Thighgk         {Upper limit of gas conductivity correlation in K}
Agk             {Agk   GasConductivity=Agk+Bgk*T+Cgk*T^2+Dgk*T^3}
Bgk             {Bgk    where T[=]K and GasConductivity[=]W/m-K}
Cgk             {Cgk}
Dgk             {Dgk}
Tlowlk          {Lower limit of liquid conductivity correlation in K}
Thighlk         {Upper limit of liquid conductivity correlation in K}
Alk             {Alk   LiquidConductivity=Alk+Blk*T+Clk*T^2+Dlk*T^3}
Blk             {Blk    where T[=]K and LiquidConductivity[=]W/m-K}
Clk             {Clk}
Dlk             {Dlk}
0               {not used: terminator}
```

The file consists of 75 lines after one or more comment lines that begin with the // characters. The first line after the comments provides the name of the fluid that EES will recognize in the property function statements (the variable Name). An optional comment can be placed on this line after the | character; this comment will be displayed when the Fluid Info button in the Function Information dialog is pressed while this fluid is selected. The fluid itself will appear in alphabetical order with the other fluid names in the Function Information dialog.

The next 74 lines each contain one number. A comment in curly braces follows on the same line (after one or more spaces) to identify the number. The molecular weight, MW, must be provided followed by the coefficients for the correlation for liquid density:

$$\rho_l(T) = a_d + b_d T_z^{1/3} + c_d T_z^{2/3} + d_d T_z + e_d T_z^{4/3} + f_d T_z^{1/2} + g_d T_z^2 \tag{4-16}$$

where T_z is defined as:

$$T_z = \left(1 - \frac{T}{T_c}\right) \tag{4-17}$$

where T_c is the critical temperature. Note that each of the coefficients must be provided in units that are consistent with ρ_l in units of lb_m/ft^3 and T in units of R. The coefficients for the correlation for vapor pressure:

$$\ln(P_v) = \frac{a_p}{T} + b_p + c_p T + d_p \left(1 - \frac{T}{T_c}\right)^{1.5} + e_p T_z^2 \tag{4-18}$$

must be provided in units that are consistent with P_v in psia and T in units of R.

The details of the equation of state are provided next. Pressure, volume and temperature are related by the Martin-Hou equation of state in the following form:

$$P = \frac{RT}{(v-b)} + \frac{A_2 + B_2 T + C_2 \exp\left(-\frac{\beta T}{T_c}\right)}{(v-b)^2} + \frac{A_3 + B_3 T + C_3 \exp\left(-\frac{\beta T}{T_c}\right)}{(v-b)^3} + \frac{A_4 + B_4 T + C_4 \exp\left(-\frac{\beta T}{T_c}\right)}{(v-b)^4} + \frac{A_5 + B_5 T + C_5 \exp\left(-\frac{\beta T}{T_c}\right)}{(v-b)^5} + \frac{A_6 + B_6 T + C_6 \exp\left(-\frac{\beta T}{T_c}\right)}{\exp(\alpha v)\left[1 + C' \exp(\alpha v)\right]} \quad (4\text{-}19)$$

The coefficients must be provided in units that are consistent with P in psia, T in R, and v in ft^3/lb_m. A method for obtaining the coefficients is described by Martin and Hou (1955).

A correlation for the ideal gas specific heat capacity at constant volume (c_{v0}) must be provided in the following form:

$$c_{v0}(T) = a_c + b_c T + c_c T^2 + d_c T^3 + \frac{e_c}{T^2} \quad (4\text{-}20)$$

where the coefficients should provided in units that are consistent with c_{v0} in Btu/lb_m-R and T in R. The values of h_{ref} and s_{ref} are the specific enthalpy and specific entropy at reference conditions (saturated liquid at -40°F).

Finally, coefficients for the gas and liquid phase viscosity and thermal conductivity correlations must be provided. The first parameter (VCT) provides the type of correlation for viscosity; currently there are two choices, VCT = 2 indicates polynomial for gas and liquid viscosity while VCT = 2.1 indicates polynomial for gas viscosity and a logarithmic function for liquid viscosity. A polynomial correlation for gas phase viscosity is provided by:

$$\mu_g \text{x} 10^{12} = A_{gv} + B_{gv} T + C_{gv} T^2 + D_{gv} T^3 \quad (4\text{-}21)$$

and for liquid phase viscosity is:

$$\mu_l \text{x} 10^{12} = A_{lv} + B_{lv} T + C_{lv} T^2 + D_{lv} T^3 \quad (4\text{-}22)$$

The alternative logarithmic correlation for liquid phase viscosity is:

$$\log_{10}(\mu_l) = A_{lv} + \frac{B_{lv}}{T} + C_{lv} T + D_{lv} T^2 \quad (4\text{-}23)$$

The polynomial forms of the conductivity correlations are similar. Note that the coefficients must be provided in units that are consistent with μ in Pa-s, k in W/m-K, and T in K.

You may need to curve fit tabular property data or data obtained from a correlation in a different form to obtain the appropriate parameters. The file below includes the information for the fluid isopropanol.

```
//isopropanol (2-propanol) C3H8O
//Correlation fit based on data provided in Yaws (1999) and N.B. Vargaftik (1975)
isopropanol|Data for isopropanol (MW=60.096) from Yaws (1999) and Vargaftik (1975).
60.096          {molecular weight of Isobutane}
0               {not used}
16.7518011      {ad Liquid density=ad+bd*Tz^(1/3)+cd*Tz^(2/3)+dd*Tz+ed*Tz^(4/3) }
42.2156075      {bd         +fd*sqrt(Tz)+gd*(Tz)^2}
-8.85376801     {cd     where Tz=(1-T/Tc) and Liquid Density[=]lbm/ft3 }
13.4145785      {dd}
0               {ed}
0               {fd}
0               {gd}
-1.288927E+04   {ap  Vapor pressure fit: lnP=a/T+b+cT+d(1-T/Tc)^1.5+eT^2}
3.156751E+01    {bp     where T[=] R and P[=]psia}
-1.763007E-02   {cp}
0               {dp}
6.209092E-06    {ep}
0               {not used}
0.178564        {R Gas constant in psia-ft3/lbm-R}
9.28443E-03     {b  Constants for Martin-Hou EOS/English_units}
-22.07494       {A2}
9.56524E-03     {B2}
-520.92446      {C2}
1.64008E+00     {A3}
-1.11778E-03    {B3}
28.73835        {C3}
-1.76126E-02    {A4}
0               {B4}
0               {C4}
0               {A5}
2.18204E-07     {B5}
-7.35278E-03    {C5}
0               {A6}
0               {B6}
0               {C6}
5.47500         {Bexp - Martin-Hou exponential constant}
0               {alpha}
0               {C'}
3.764019E-02    {ac   cv(0 pressure) = ac + bc T + cc T^2 + dc T^3 + ec/T^2 }
6.168131E-04    {bc           where T[=]R and Cv[=]Btu/lb-R }
-1.411459E-07   {cc}
7.379913E-12    {dc}
0               {ec}
273.062322      {href offset set to 0 for sat'd liquid at -40F}
0.0459364       {sref offset}
690.961         {Pc [=] psia}
914.958         {Tc [=] R}
0.05408         {vc [=] ft3/lbm}
0               {not used}
0               {not used}
2.1             {Viscosity correlation type: 2.1 indicates log type for liquid}
200             {Lower limit of gas viscosity correlation in K}
1000            {Upper limit of gas viscosity correlation in K}
-10.859E5       {Agv    GasViscosity*1E12=Agv+Bgv*T+Cgv*T^2+Dgv*T^3}
3.0873e4        {Bgv    where T[=]K and GasViscosity[=]N-s/m2 }
-4.8098         {Cgv}
0               {Dgv}
185             {Lower limit of liquid viscosity correlation in K}
```

```
508             {Upper limit of liquid viscosity correlation in K}
-0.7009         {Alv   log10(Liquid Viscosity)=Alv+Blv/T+Clv*T+Dlv*T^2}
8.415e2         {Blv    where T[=]K and Liquid Viscosity[=]N-s/m2}
-8.6068e-3      {Clv}
8.2964e-6       {Dlv}
2               {Conductivity correlation type: set to 2: do not change}
350             {Lower limit of gas conductivity correlation in K}
560             {Upper limit of gas conductivity correlation in K}
0.0165723       {Agk}   GasConductivity=Agk+Bgk*T+Cgk*T^2+Dgk*T^3
-4.93183e-5     {Bgk    where T[=]K and GasConductivity[=]W/m-K}
1.80816e-7      {Cgk}
0               {Dgk}
185             {Lower limit of liquid conductivity correlation in K}
483             {Upper limit of liquid conductivity correlation in K}
2.209607E-01    {Alk   LiquidConductivity=A+B*T+C*T^2+D*T^3 (from 3M data)}
-4.746872E-04   {Blk    where T[=]K and LiquidConductivity[=]W/m-K}
1.090827E-06    {Clk}
-1.456185E-09   {Dlk}
0               {not used: terminator}
```

Placing the file isopropanol.mhe in the ...\UserLib subdirectory and EES will automatically add the real fluid isopropanol to the list of fluids, as shown in Figure 4-33.

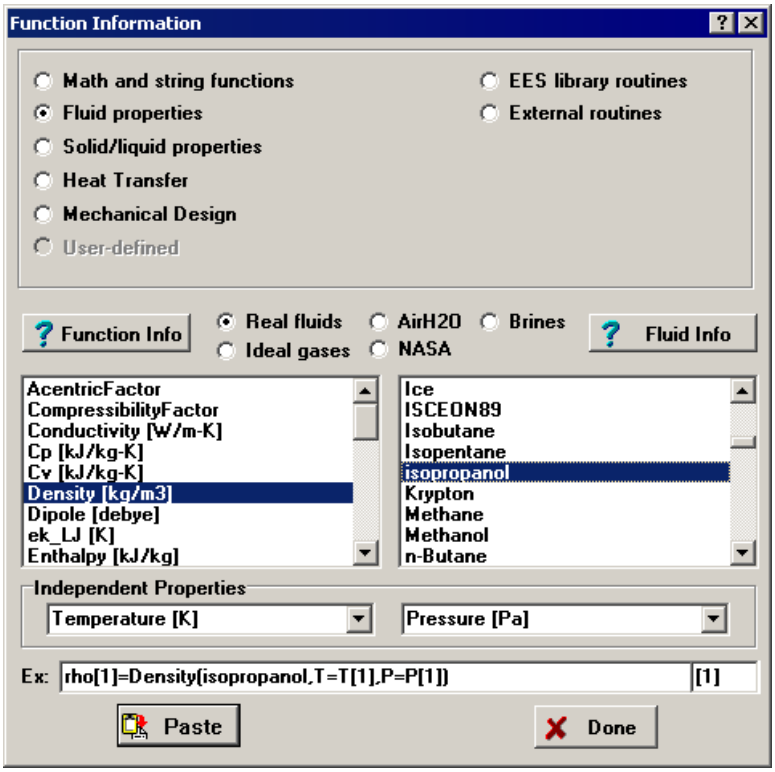


Figure 4-33: Function Information dialog showing the added real fluid isopropanol.

The fluid properties for the additional fluid are obtained just as any for any of the built-in properties in EES. For example, the enthalpy for this substance would be obtained as follows:

```
$UnitSystem SI Mass Rad J K Pa
h=Enthalpy(isopropanol,T=350 [K],P=250000 [Pa])
```

References

Harr, L. Gallagher, J.S., and Kell, G.S (Hemisphere, 1984). NBS/NRC Steam Tables, Hemisphere Publishing Company, Washington, (1984).

Ibrahim, O.M., Klein, S.A.,"Thermodynamic Properties of Ammonia-Water Mixtures," ASHRAE Trans.: Symposia, 21, 2, 1495 (1993).

Klein, S.A. and Nellis, G.F., *Thermodynamics*, Cambridge University Press, New York, (2012).

Martin, J.J. and Hou, Y.C., "Development of an Equation of State for Gases," A.I.Ch.E Journal, 1:142, (1955)

McBride, B.J., Zehe, M.J., and Gordon, S "NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species", NASA/TP-2002-211556, Sept. (2002), http://gltrs.grc.nasa.gov/reports/2002/TP-2002-211556.pdf

Melinder, Å *Properties of Secondary Working Fluids for Indirect Systems*, IIF/IIR, 2010, http://www.iifiir.org/en/details.php?id=1177

National Institute of Standards and Technology, NIST Standard Reference Database 23, NIST Reference Fluid Thermodynamic and Transport Properties Database (REFPROP): Version 9.0, http://www.nist.gov/srd/nist23.cfm.

Patek, J. and Klomfar, J., "A computationally effective formulation of the thermodynamic properties of LiBr-H2O from 273 to 500 K over full composition range", Int. J. of Refrigeration, Vol 29, pp. 566-578, (2006)

Patek, J. and Klomfar, J., "Thermodynamic properties of the LiCl-H2O system at vapor-liquid equilibrium from 273 K to 400 K", Int. J. of Refrigeration, Vol. 31, pp. 287-303, (2008)

Peng, D and Robinson, D.B., "A New Two-Constant Equation of State", Ind. Eng. Chem. Fundam., Vol. 15, No. 1, pp. 59-64,(1976)

Saul, A. and Wagner, W., "International Equations for the Saturation Properties of Ordinary Water Substance," J. Phys. Chem. Ref. Data, 16, 893 (1987)

Soave, G., "Equilibrium Constants from a Modified Redlich-Kwong Equation of State," Chemical Engineering Science, Vol. 27, pp. 1197-1203, (1972)

Span, R., Multiparameter Equations of State, Springer, ISBN 3-540-67311-3, (2000)

Wagner, W., and Pruss, A. "International Equations for the Saturation Properties of Ordinary Water Substance. Revised According to the International Temperature Scale of 1990," J. Phys. Chem. Ref. Data, 22, 783, (1993)

5 CONVERGENCE AND DEBUGGING

EES is not a programming language like C, Pascal, FORTRAN or MATLAB. Programming languages use assignments; that is, they require the user to enter statements in which the variable that is to be determined (i.e., assigned a value) is on the left of the equal sign. All variables appear on the right of the equal sign and they must have pre-determined values.

EES is very different because it is an equation-solver. EES uses equations that are simply relationships between variables. These equations can be entered in any way that you wish. There is no requirement that the variables on the right side of an equation have pre-determined values. The equations can be entered in any order that you wish. The equations can be linear or non-linear. Non-linear equations may have multiple solutions and necessarily require iterative solution methods. Unlike any programming language, EES automatically recognizes when iteration is needed and it will internally apply a variety of powerful numerical methods to solve the system of equations that have been entered into the Equation Window. However, it is not always possible to solve sets of non-linear equations without some user input. The most important user input required for this purpose is a set of guess values that can be used as a starting point in the iterative solution process. This chapter reviews the basic solution methods that EES employs to solve sets of non-linear equations in order to demonstrate the need for guess values. It also describes debugging techniques that are available in EES that can be used to ensure that the equation set converges to the solution that you intended.

5.1 Solution Methodology Used in EES

The various techniques that EES uses to solve sets of equations are discussed in this section in order to clarify how EES, an equation solver, differs from programming languages. Familiarity with these solution methods also makes many of the terms that are used to describe the debugging features in EES more clear.

Numerical Solution of One Non-Linear Equation

Consider the following non-linear equation in one unknown:

$$x^{2.5} - 3.5\,x^2 + 2\,x = 10 \tag{5-1}$$

This equation does not have an analytic solution. It must be solved numerically. The most convenient way to solve one or more non-linear equations is to rewrite the equation(s) in terms of a residual by subtracting the right side of the equation from the left side. Equation (5-1) can be rewritten in terms of the residual, $f(x)$:

$$f(x) = x^{2.5} - 3.5\,x^2 + 2\,x - 10 \tag{5-2}$$

A solution to this equation is a value of x that causes the residual, $f(x)$, to be zero. Non-linear equations may have multiple solutions. This is one reason that guess values and bounds may be needed during the solution process in order to ensure that the solution that is obtained is the desired solution, as discussed in Section 2.2.

A plot of $f(x)$ for values of x between 0 and 15 is shown in Figure 5-1. There is only one real solution in this range and that solution occurs at $x = 11.5726$.

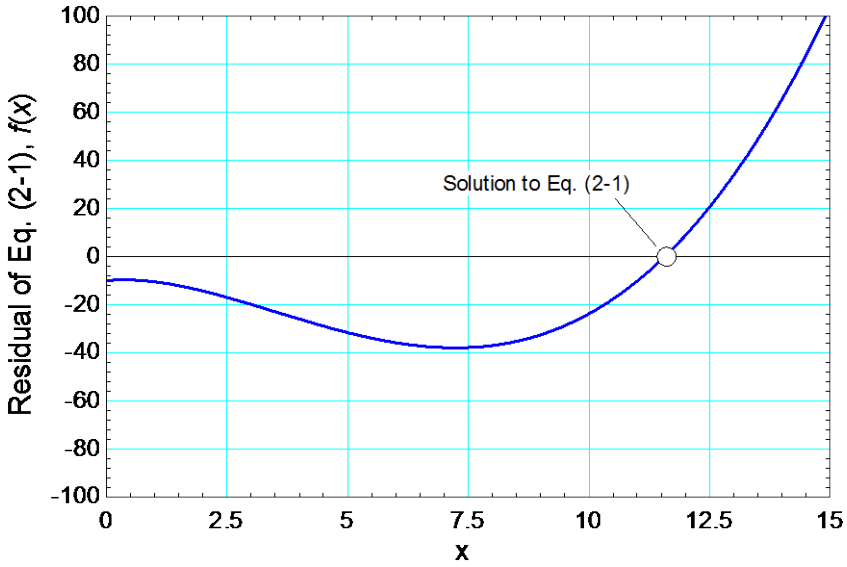


Figure 5-1: Residual of Eq. (5-1), $f(x)$, as a function of x.

One way to numerically identify this solution is to use a Taylor's series expansion of Eq. (5-2), as shown in Eq. (5-3); note that only the first through third order terms are shown for clarity. In Eq. (5-3), x_i is the current (or guessed) value of x and x_s is the solution; i.e., the value of x that results in the residual $f(x_s)$ being equal to zero.

$$f(x_s) = f(x_i) + \frac{df(x_i)}{dx}(x_s - x_i) + \left(\frac{1}{2!}\right)\frac{d^2 f(x_i)}{dx^2}(x_s - x_i)^2 + \left(\frac{1}{3!}\right)\frac{d^3 f(x_i)}{dx^3}(x_s - x_i)^3 \ldots = 0 \tag{5-3}$$

Equation (5-3) provides the exact solution to the problem but is not solvable due to the infinite nature of the series. The numerical iteration technique ignores the second order and higher terms, allowing Eq. (5-3) to be written as:

$$f(x_{i+1}) \approx f(x_i) + J(x_i)(x_{i+1} - x_i) \tag{5-4}$$

where

$$J(x_i) = \frac{df(x_i)}{dx} \tag{5-5}$$

and x_{i+1} is an improved estimate for x, i.e., a value that hopefully results in a value of $f(x_{i+1})$ that is closer to zero but not exactly equal to zero since Eq. (5-4) is an approximation. We are trying to find a value of x_{i+1} that causes $f(x_{i+1})$ to be closer to zero. Therefore, the left side of Eq. (5-4) can be set to zero in order to select our best value of x_{i+1}:

$$f(x_i) + J(x_i)(x_{i+1} - x_i) = 0 \tag{5-6}$$

Solving Eq. (5-6) for x_{i+1} provides:

$$x_{i+1} = x_i - \frac{f(x_i)}{J(x_i)} \tag{5-7}$$

Equation (5-7) provides a method in which improved estimates of x can be obtained iteratively. This solution technique is called Newton's method and it is very efficient at finding a solution, when it works.

To solve the equation, Newton's method proceeds using the following steps.

1. An initial guess is made for x_1 (e.g., 10).
2. The residual, $f(x_1)$, is evaluated from Eq. (5-2).

$$f(x_1) = (10)^{2.5} - 3.5\ (10)^2 + 2\ (10) - 10 = -23.77 \tag{5-8}$$

3. The derivative, $J(x_1)$, is evaluated (typically numerically but in this case analytically):

$$J(x_1) = \frac{df(x_1)}{dx} = 2.5\,x_1^{1.5} - 3.5(2)\ x_1 + 2 = 2.5(10)^{1.5} - 3.5(2)(10) + 2 = 11.06 \tag{5-9}$$

4. The next estimate for x, x_2, is calculated using Eq. (5-7).

$$x_2 = x_1 - \frac{f(x_1)}{J(x_1)} = 10 - \frac{(-23.77)}{11.06} = 12.15 \tag{5-10}$$

Steps 2 to 4 are repeated using the most recent value of x in order to evaluate the residual, f, and its derivative, J. Each iteration (hopefully) provides an improved estimate for x.

The EES program below automatically carries out 5 of these iterations using the Duplicate command and arrays, discussed in Section 1.7.

```
x[1]=10                                  "guess value"
duplicate i=1,5                          "carry out iterations"
    f[i]=x[i]^2.5-3.5*x[i]^2+2*x[i]-10   "residual"
    J[i]=2.5*x[i]^1.5-3.5*2*x[i]+2       "derivative"
    x[i+1]=x[i]-f[i]/J[i]                "improved estimate"
end
```

The Arrays Table in Figure 5-2 shows the results of the iterative process.

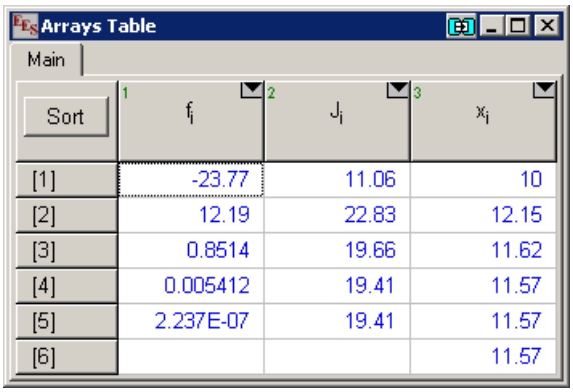


Figure 5-2: Arrays Table showing iteration results for 5 iterations.

Note that Newton's method converged to a solution of 11.57 (which is the solution that was graphically identified in Figure 5-1) after just a few iterations. When Newton's method works, it works well.

Stopping Criteria

Some type of criterion is needed to stop the iteration process. The criterion could be the number of iterations (as was used in the EES program above, which was terminated after 5 iterations). Alternatively, we could monitor the absolute value of the residual, $f(x)$, or the absolute value of the change in the independent variable between iterations, $x_{i+1} - x_i$, and stop the iteration process when these indicators become smaller than a specified tolerance. These are the criteria that can be selected in the Stop Crit tab of the Preferences dialog (selected from the Options menu). The stop criteria set in Figure 5-3 will cause the iteration process to stop when the number of iterations reaches 250, or the relative residual (i.e., the residual normalized by the value of the left side of the equation, assuming it is not zero) is reduced below $1x10^{-6}$, or the change in variables between iterations is reduced below $1x10^{-9}$. In addition, the iterations will stop if the elapsed time reaches 3600 s.

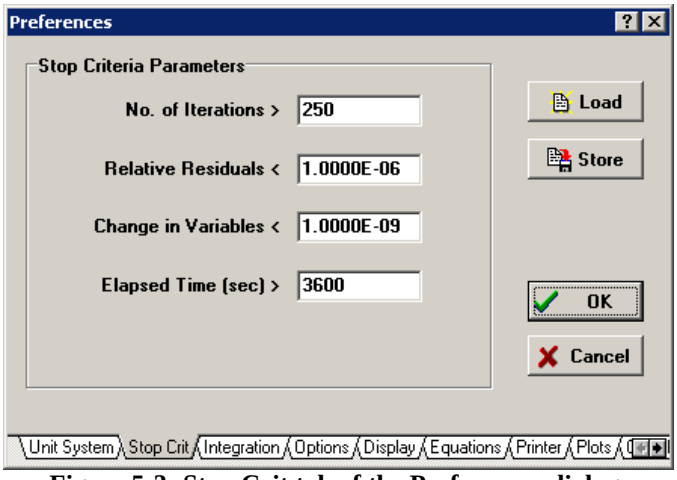


Figure 5-3: Stop Crit tab of the Preferences dialog.

The stop criteria used by EES can also be specified using the $StopCriteria directive. For example, the statement:

```
$StopCriteria  Iterations = 250  Residuals = 1e-6  Variables = 1e-9  Time = 3600
```

placed at the top of the Equations Window will result in the same stop criteria shown in Figure 5-3. Stop criteria specified with the $StopCriteria directive override those set in the Stop Crit tab of the Preferences dialog.

Unfortunately, Newton's method (or any other iterative method for solving non-linear sets of equations) does not always work. A poor initial guess can cause the method to diverge or to encounter a numerical problem (e.g., raising a negative number to a non-integer power). Try, for example, running the EES program with an initial guess of 5 instead of 10 and see what happens. (You will not be able to obtain a solution using Newton's method.) The first estimate of the unknown quantity (i.e., x_1) is called a guess value. As you can see from this simple example, the choice of guess value is very important. Guess values and lower and upper bounds for each variable in EES can be specified by selecting Variable Info from the Options menu, as discussed in Section 1.2.

Numerical Solution of Simultaneous Non-Linear Equations

Newton's method can be extended to solve a set of simultaneous non-linear equations. In this case, the derivative, $J(x)$ in Eqs. (5-7) and (5-5), must be replaced with a matrix called the "Jacobian matrix", $\underline{\underline{J}}$. Consider the following two simultaneous non-linear equations in two unknowns:

$$\begin{aligned} x^2 + y^2 &= 18 \\ x^3 &= \frac{2}{y} - 2 \end{aligned} \tag{5-11}$$

These equations can be rewritten in terms of a residuals vector, $\underline{f}$:

$$\underline{f} = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} \tag{5-12}$$

where

$$\begin{aligned} f_1(x,y) &= x^2 + y^2 - 18 \\ f_2(x,y) &= x^3 - \frac{2}{y} + 2 \end{aligned} \tag{5-13}$$

The Jacobian for a system of two equations in two unknowns is a 2x2 matrix. The first column of the first row contains the partial derivative of the first equation with respect to the first variable:

$$J_{1,1} = \frac{\partial f_1}{\partial x} = 2x \tag{5-14}$$

The second column of the first row contains the partial derivative of the first equation with respect to the second variable:

$$J_{1,2} = \frac{\partial f_1}{\partial y} = 2y \tag{5-15}$$

The first column of the second row contains the partial derivative of the second equation with respect to the first variable:

$$J_{2,1} = \frac{\partial f_2}{\partial x} = 3x^2 \tag{5-16}$$

The second column of the second row contains the partial derivative of the second equation with respect to the second variable:

$$J_{2,2} = \frac{\partial f_2}{\partial y} = \frac{2}{y^2} \tag{5-17}$$

In general $J_{i,j}$ is the partial derivative of equation i with respect to variable j. For this example, the Jacobian matrix is:

$$\underline{\underline{J}} = \begin{bmatrix} 2x & 2y \\ 3x^2 & \dfrac{2}{y^2} \end{bmatrix} \tag{5-18}$$

Newton's method, as presented in Eq. (5-7), can be extended for this system of two equations and two unknowns according to:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \underline{\underline{J}}^{-1}\,\underline{f} \tag{5-19}$$

where $\underline{\underline{J}}^{-1}$ is the inverse of the Jacobian matrix. For a 2x2 matrix, the inverse can be written as:

$$\underline{\underline{J}}^{-1} = \frac{1}{\left(J_{1,1}J_{2,2} - J_{1,2}J_{2,1}\right)} \begin{bmatrix} J_{2,2} & -J_{1,2} \\ -J_{2,1} & J_{1,1} \end{bmatrix} \tag{5-20}$$

Substituting Eq. (5-20) into Eq. (5-19) leads to:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \frac{1}{\left(J_{1,1}\,J_{2,2} - J_{1,2}\,J_{2,1}\right)} \begin{bmatrix} J_{2,2} & -J_{1,2} \\ -J_{2,1} & J_{1,1} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{5-21}$$

or, carrying out the matrix multiplication:

$$\begin{aligned} x_{i+1} &= x_i - \frac{\left(J_{2,2}\,f_1 - J_{1,2}\,f_2\right)}{\left(J_{1,1}\,J_{2,2} - J_{1,2}\,J_{2,1}\right)} \\ y_{i+1} &= y_i - \frac{\left(-J_{2,1}\,f_1 + J_{1,1}\,f_2\right)}{\left(J_{1,1}\,J_{2,2} - J_{1,2}\,J_{2,1}\right)} \end{aligned} \tag{5-22}$$

We can illustrate this solution process in EES. Initial guesses for x (x_1) and y (y_1) are needed to start the iterative process. Consider the following initial set of guesses:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \tag{5-23}$$

```
"initial guesses"
x[1]=2
y[1]=3
```

The iterative process is carried out 10 times using a Duplicate loop. The residual is computed according to Eq. (5-13).

```
Duplicate i=1,10                                    "carry out iterations"
    "calculate residual"
    f1[i]=x[i]^2+y[i]^2-18
    f2[i]=x[i]^3-2/y[i]+2
```

The Jacobian is calculated according to Eqs. (5-14) through (5-17):

```
    "calculate Jacobian"
    J11[i]=2*x[i]
    J12[i]=2*y[i]
    J21[i]=3*x[i]^2
    J22[i]=2/y[i]^2
```

Newton's method is applied according to Eq. (5-22):

```
    "apply Newton's method"
    x[i+1]=x[i]-(J22[i]*f1[i]-J12[i]*f2[i])/(J11[i]*J22[i]-J12[i]*J21[i])
    y[i+1]=y[i]-(-J21[i]*f1[i]+J11[i]*f2[i])/(J11[i]*J22[i]-J12[i]*J21[i])
end
```

Run the EES code to obtain the Arrays Table shown in Figure 5-4.

Arrays Table

Main

Sort	1 $f1_i$	2 $f2_i$	3 $J11_i$	4 $J12_i$	5 $J21_i$	6 $J22_i$	7 x_i	8 y_i
[1]	-5	9.333	4	6	12	0.2222	2	3
[2]	2.518	3.257	2.394	8.738	4.298	0.1048	1.197	4.369
[3]	0.5779	1.619	0.8821	8.575	0.5835	0.1088	0.441	4.288
[4]	7.982	-11.85	-4.751	9.02	16.93	0.09833	-2.375	4.51
[5]	0.7589	-3.18	-3.345	7.99	8.392	0.1253	-1.673	3.995
[6]	0.1468	-0.6628	-2.589	8.117	5.028	0.1214	-1.295	4.058
[7]	0.01779	-0.06465	-2.327	8.164	4.061	0.12	-1.163	4.082
[8]	0.0002567	-0.0008732	-2.295	8.169	3.951	0.1199	-1.148	4.085
[9]	4.937E-08	-1.668E-07	-2.295	8.169	3.949	0.1199	-1.147	4.085
[10]	1.801E-15	-6.090E-15	-2.295	8.169	3.949	0.1199	-1.147	4.085
[11]							-1.147	4.085

Figure 5-4: Arrays Table showing iteration results for 10 iterations of Newton's method.

The values of the variables x and y converge to -1.147 and 4.085, respectively, after 8 iterations; this is the correct solution to this problem. Of course, it is not necessary to do the iterations explicitly to solve this system of equations in EES. The solution could have been obtained by entering the following two equations directly into the Equations Window:

```
x^2+y^2=18                                   "solve the equations directly"
x^3=2/y-2
```

Solving will show the same result as obtained for the last iteration in Figure 5-4. However, to obtain these results, EES internally had to use a numerical technique that is similar to Newton's method.

Newton's method applies to both linear and non-linear sets of equations. If the equations are linear then convergence will generally occur after one iteration, regardless of the guess values that are provided for the unknown variables. For a set of non-linear equations the number of iterations required depends on the guess values as well as on the stopping criteria. The stopping criteria are set from the Stop Crit tab in the Preferences dialog (selected from the Options menu).

The Jacobian matrix plays a key role in the solution of algebraic equations. It is also used to block the set of equations, as explained in the next subsection. EES evaluates the Jacobian matrix numerically, which necessarily introduces some error. The accuracy of the Jacobian may change the number of iterations required to solve the equations, but it does not necessarily affect the accuracy of the solution. Because EES does all calculations with 80 bit precision (about 20 decimal places), numeric evaluation of the Jacobian rarely results in numerical problems from loss of precision.

In most equation sets, many of the elements of the Jacobian matrix are zero. It is wasteful to store these zero elements and even more wasteful to use them in multiplication processes. A matrix with many zero elements is called a sparse matrix. Special techniques are available to efficiently work with sparse matrices; EES automatically uses sparse matrix routines as needed. Without the use of sparse matrix techniques, the number of simultaneous equations that could be solved in a reasonable time using EES would be substantially reduced. Certainly it would not be possible to solve problems with 6,000 equations, which is the current limit. (12,000 equations can be used in the Professional version.)

Newton's method does not always work. This is particularly true if the guess values are not reasonably close to the desired solution. The value of the residuals of the equations should become closer to zero after each iteration. When EES is unable to reduce the residuals using Newton's method, it will attempt to use several alternative backup methods. One method is to reduce the step size, according to:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \underline{\underline{J}}^{-1} \underline{f} \varepsilon \tag{5-24}$$

where ε is a factor between 0 and 1. Smaller values of ε improve the chances of having a difficult set of equations converge, but using any value of ε less than one increases the number of iterations required to obtain a solution. For example, when ε is 0.5, approximately 18 iterations are required to achieve the result that was obtained in Figure 5-4 with 8 iterations. EES will control the value of ε during the calculations in order to cause the residuals to be closer to zero during each iteration. If this method is also unsuccessful then EES will attempt a few other methods and finally quit if it is unable to solve the equations.

Blocking and Reordering Equation Sets

Even though you may have what looks like a set of simultaneous equations, it is often possible to solve these equations in groups (or blocks) rather than solving all of the equations simultaneously. In some cases, a block may only contain a single equation. Solving equations in blocks makes Newton's method work more reliably because it reduces the number of variables that must be determined at any one time. Blocking also reduces the required computation time. EES automatically organizes the equations into blocks before attempting to solve the equations; this reorganization occurs regardless of the order in which the user has entered the equations in the Equations Window.

To illustrate this method, consider the set of eight equations in eight unknowns shown below:

$$x_3 + x_8 = 11 \tag{5-25}$$

$$x_4 = x_7 + 4 \tag{5-26}$$

$$x_5 - x_6 - x_7 = -8 \tag{5-27}$$

$$x_1 + x_4 - x_6^2 = -1 \tag{5-28}$$

$$x_2 + x_8 = 10 \tag{5-29}$$

$$x_3 - x_5 + x_7 + \sqrt{x_8} = 5 \tag{5-30}$$

$$x_7 = 7 \tag{5-31}$$

$$x_1 + x_6 + x_7 = 14 \tag{5-32}$$

These equations have been entered into the Equations Window:

```
x_3+x_8=11
x_4=x_7+4
x_5-x_6-x_7=-8
x_1+x_4-x_6^2=-1
x_2+x_8=10
x_3-x_5+x_7+sqrt(x_8)=5
x_7=7
x_1+x_6+x_7=14
```

and used to obtain the solution shown in Figure 5-5.

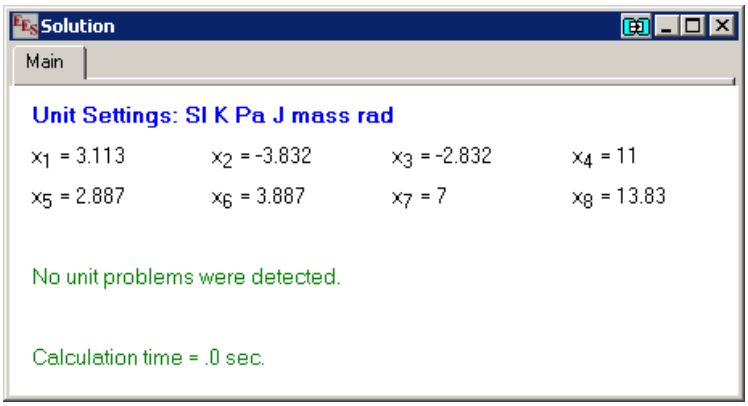


Figure 5-5: Solution Window.

These equations could be solved as one simultaneous set. However, they can be more easily solved if they are reordered and blocked; EES accomplishes this automatically. For example, the process of solving Eqs. (5-25) through (5-32) can be simplified substantially if one recognizes that Eq. (5-31) and Eq. (5-26):

$$\underline{\text{Block 0}}$$
$$x_7 = 7$$
$$x_4 = x_7 + 4 \tag{5-33}$$

can be solved immediately and alone in order to provide x_7 = 7 and x_4 = 11. With x_7 and x_4 known, Eq. (5-28) and Eq. (5-32):

$$\underline{\text{Block 1}}$$
$$x_1 + x_4 - x_6^2 = -1$$
$$x_1 + x_6 + x_7 = 14 \tag{5-34}$$

can be solved simultaneously to determine x_1 = 3.113 and x_6 = 3.887. With x_1, x_4, x_6, and x_7 known, Eq. (5-27):

$$\underline{\text{Block 2}}$$
$$x_5 - x_6 - x_7 = -8 \tag{5-35}$$

can be solved for x_5 = 2.887. Equation (5-25) and Eq. (5-30):

$$\underline{\text{Block 3}}$$
$$\begin{aligned} x_3 + x_8 &= 11 \\ x_3 - x_5 + x_7 + \sqrt{x_8} &= 5 \end{aligned} \tag{5-36}$$

can be solved for x_3 = -2.832 and x_8 = 13.83. Finally, Eq. (5-29):

$$\underline{\text{Block 4}}$$
$$x_2 + x_8 = 10 \tag{5-37}$$

can be solved for x_2 = -3.832. At this point, all of the equations have been solved and, in this case, a maximum of two equations needed to be solved simultaneously.

Blocking and reordering the equations is useful even when all of the equations are linear; however, in this case it is not essential. When one or more of the equations are nonlinear, as in this example, then blocking of equations is indispensable as it reduces the required number of iterations needed to solve the problem and improves the likelihood of finding a solution. It is must easier to solve several small sets of equations than one larger set. EES is able to recognize groups of equations prior to solution by inspecting the Jacobian matrix using the Tarjan (1972) algorithm. The result of the blocking and reordering process in EES is most evident in the Residuals Window, which is discussed in the following section.

5.2 The Residuals Window

Section 5.1 summarized the solution methods that EES employs to solve equations. EES modifies each equation so that it is represented as a residual by subtracting the right side of the equation from the left side. EES will then reorder and block the equations in order to optimize the computational efficiency of the solution process. Consequently, the equations that you enter may be solved in an order that is different from the order that you entered them. Knowing the order in which EES solves the equations can be very helpful to understanding the nature of your system of equations. It is also often helpful for debugging purposes, as discussed in this section. The order that the equations are solved, the values of the residuals, and other information are provided in the Residuals Window.

Blocks

Figure 5-6 shows the Residuals Window (select Residuals from the Windows menu) for the equation set associated with Eqs. (5-25) through (5-32). The first column shows the block number which indicates the reordering that EES has done to solve the equations; note that this blocking results in the same result that was identified in Eqs. (5-33) through (5-37). Equations that have a single unknown and can be solved one at a time are assigned to Block 0, as shown in Eq. (5-33). The order that the equations appear in block zero is the order in which EES attempts to solve them. For example, note that EES has chosen to solve Eq. (5-31) first, even though it was the seventh equation in order of appearance in the Equations Window. With x_7 known, x_4 can be found by solving Eq. (5-26). That completes Block 0 for this example because the next two equations must be solved simultaneously in Block 1 in order to determine x_1 and x_6. Note that the variables x_1 and x_6 are shown in bold in the rows for Block 1 because these are the unknown variables in this block. Blocks 0 through 4 are solved sequentially, with all of the equations in the same block (except block 0) being solved simultaneously.

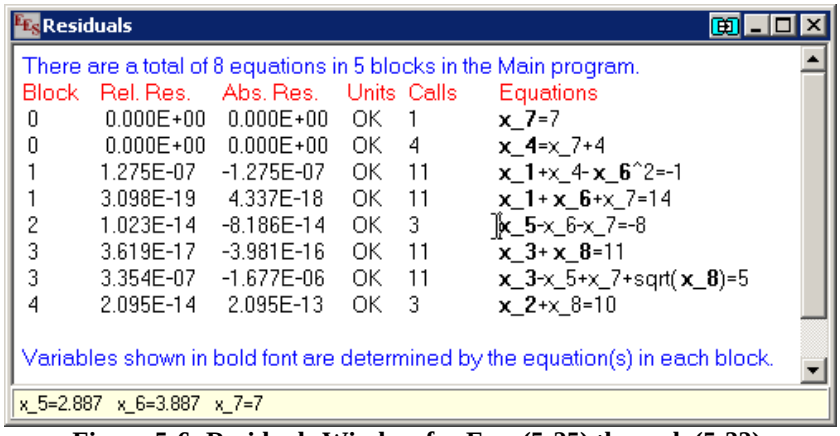


Figure 5-6: Residuals Window for Eqs. (5-25) through (5-32).

Residuals

The Residuals Window displays the relative and absolute residuals for every equation in the main program, as well as in subprograms and modules (discussed in Chapter 10). The absolute residual of an equation is the difference between the values on the left and right hand sides of the equation. The relative residual is the absolute value of the residual divided by the value of magnitude of the left side of the equation. If the value of the left hand side of an equation is zero (or close to zero), then the relative residual is assigned to be equal to the absolute residual. EES monitors the value of the relative residuals during iterative calculations to determine when the equations have been solved to the accuracy specified in the Stopping Crit tab of the Preferences dialog or the $StopCriteria directive. Using the relative residuals instead of the absolute residuals in the stopping criteria makes it possible to specify a tolerance that is independent of the scale of each equation.

Units

The Units column in the Residuals Window indicates the status of the unit checking; OK appears if EES did not find any unit checking problem. Figure 5-6 indicates that there are no unit conversion problems since units were not entered for any of the variables. If, for example, we set the units of variable x_7 to ft and resolve this problem then the Residuals Window would appear as shown in Figure 5-7.

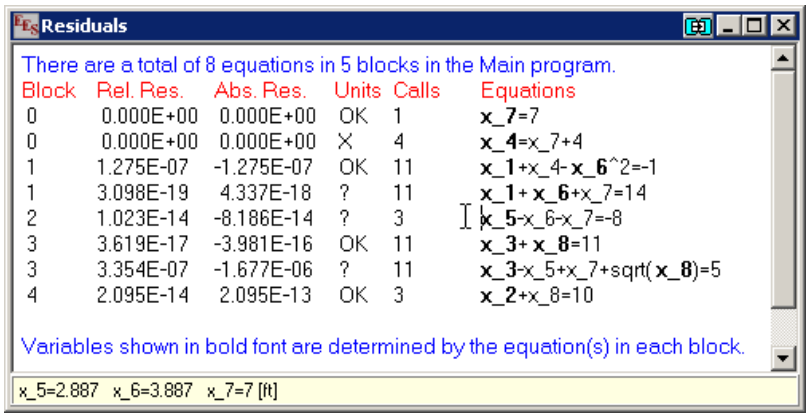


Figure 5-7: Residuals Window showing unit checking warnings.

In Figure 5-7, the equations that failed the unit checking process are indicated by a ? in the Units column. Clicking the right mouse button on the ? will cause the pop-up menu shown in Figure 5-8 to appear.

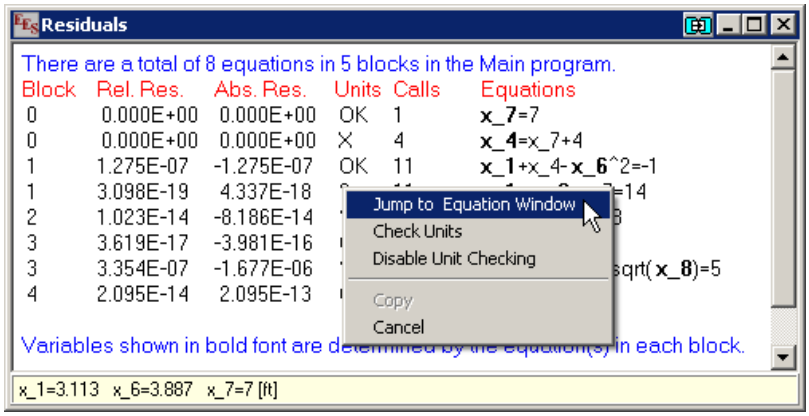


Figure 5-8: Pop-up menu that appears after right-clicking in the Units column.

The pop-up menu in Figure 5-8 has provides several options for addressing the unit problem. One option is to disable the unit checking for the selected equation. If unit checking for an equation is disabled, then the Units column will show an X, as seen for the second equation in Figure 5-8. The Check Units option will provide a more detailed description of the unit inconsistency.

Calls

The number of times that the equation needed to be evaluated in the process of obtaining a solution is shown in the Calls column. This number includes the calls needed to evaluate numerical derivatives. EES solves all equations (except assignment statements) numerically which explains why 4 calls were needed to solve the second equation. The number of calls will necessarily increase with the number of variables that are being determined simultaneously. Numerical derivatives are required for two variables in the third and fourth equations, which is why 11 calls were required. A very large number of calls for an equation usually indicates that the guess values for the variables involved in that equation could be improved.

Clicking on any row in the Residuals Window will display the variables that are in the equation with their units in the status bar at the bottom of the window, as seen for the fifth equation in Figure 5-7. The height of the status bar will automatically adjust so allow all of the variables in the selected equation to be displayed. The Doubling-clicking the left mouse button (or clicking the right mouse button) on a row in the Residuals Window will cause the Equations Window to be brought to the front with the selected equation highlighted. The Find command from the Search menu can be used to help locate the equations in large problems. The entire contents of the Residuals window can be copied as tab-delimited text to the Clipboard. To do this, select the Select All command from the Edit menu and then the Copy command from the Edit menu while the Residuals window is foremost.

Procedures in the Residuals Window

Procedures are compartmentalized code segments that can be used to return multiple outputs. In Section 3.3, we built the simple procedure called Test that computes the product, ratio, sum, and difference of two inputs X and Y:

```
Procedure Test(X,Y: M,D,A,S)
    M:=X*Y                                  "multiply"
    D:=X/Y                                  "divide"
    A:=X+Y                                  "add"
    S:=X-Y                                  "subtract"
end
```

Internally, EES treats a call to a procedure as *M* separate function calls, where *M* is the number of output parameters in the procedure. Therefore, the call to the procedure Test corresponds to four function references. For example, the code:

```
Call Test(33,44: Product, Ratio, Sum, Difference)        "Call to the Test Procedure"
```

will lead to the Residuals Window shown in Figure 5-9(a). Even though the Equations Window contained only one call to the procedure Test, the Residuals Window shows four separate functions corresponding to each of the outputs of this procedure. The first row is Product = Test(33,44,1) which should be read as an equation setting the value of the variable Product by calling the procedure Test with inputs X = 33 and Y = 44. The 1 indicates that Product occupies the first output argument position. The remaining three rows correspond to the other outputs. Because the inputs are specified, the four equations are blocked sequentially.

Procedures can also be called with one or more of the outputs set in order to determine one or more of the inputs. For example, the procedure Test could be called according to:

```
Call Test(X, Y: Product, Ratio, 88, 32)          "Alternative call to the Test Procedure"
```

which would lead to the Residuals Window shown in Figure 5-9(b). The Residuals Window shows that EES used the equations corresponding to the known third and fourth outputs in order to determine variables X and Y simultaneously. Thus these two equations were placed in Block 1. Then, the variables Product and Ratio were determined. EES saves the input and output values for each procedure call and reuses this information, if possible, in order to reduce unnecessary computation.

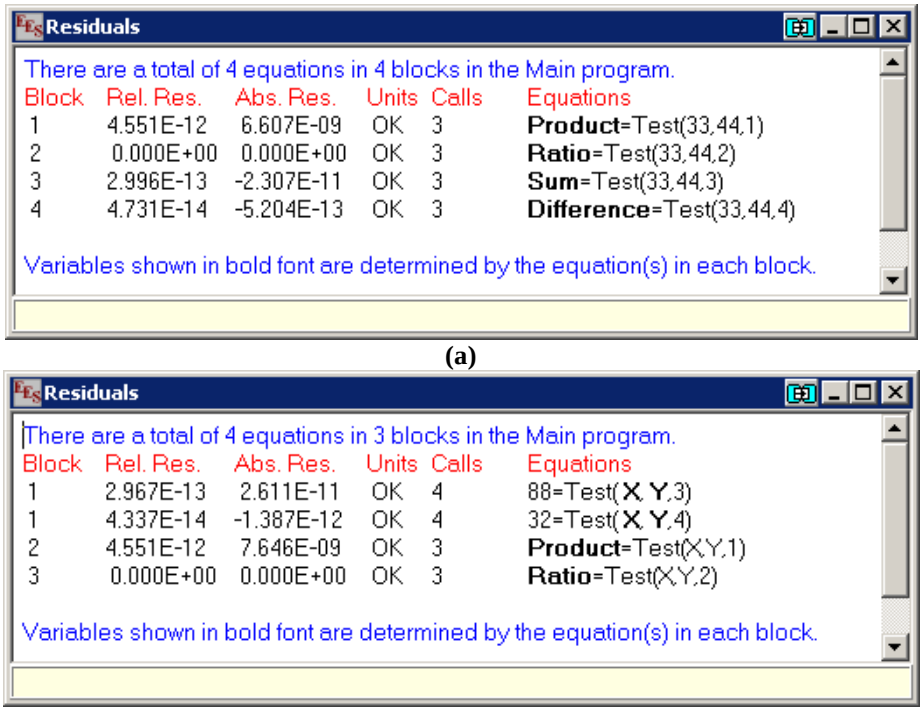


Figure 5-9: Residuals Window for the call to the procedure Test with (a) inputs set and outputs calculated and (b) two outputs set and the inputs calculated.

5.3 Setting Guess Values and Limits

The iterative solution methodology used in EES requires guess values for the variables appearing in the equations, as explained in Section 5.1. Non-linear equations may have more than one solution. Lower and upper bounds are sometimes needed to ensure that the desired solution is obtained. The specification of these bounds can also prevent numerical problems, such as division by zero or taking the square root of a negative number.

By default, the guess values of all variables in EES are 1.0 and the lower and upper bounds are negative and positive infinity, respectively. The guess value, limits, display format, units and

other information for each variable in an EES program can be specified using the Variable Information command in the Options menu, as noted in Section 1.2. The Variable Information command provides some additional capabilities that are discussed here.

The Variable Information Window

Consider the simple set of equations shown below:

```
Z=37
X[1]^2+X[2]^2=Z
X[1]/X[2]=1.2345
```

Entering the Solve command (or pressing F2) will provide a solution for these equations: X[1] = 4.727 and X[2] = 3.829. However, this is not the only possible solution. Another solution to these equations is: X[1] = -4.727 and X[2] = -3.829. EES only converges to and displays one solution, even when multiple solutions exist. The solution it displays depends on the guess values and possibly on the variable bounds. To demonstrate this dependence, select the Variable Information command from the Options menu in order to display the Variable Information Window, shown in Figure 5-10.

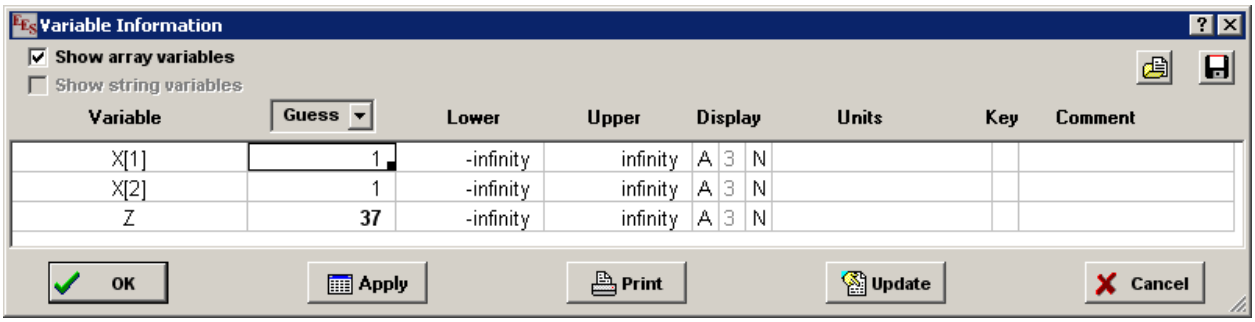


Figure 5-10: Variable Information Window with default information.

The Variable Information Window shows that the guess values for X[1] and X[2] are both 1; this is the default value for all variables. EES shows the guess value for Z to be 37 and displays this value in bold font. However, the guess value for Z is not 37. EES is displaying this value because it thinks that this is the value of variable (and it is correct in this case). Even though a value is provided, you can edit it if you wish. If you click in the edit box containing the 37, it will change to 1 (the real guess value) and remove the bold font. Change the guess values of X[1] and X[2] to be -1. Also set the upper bounds for these variables to be 0, as shown in Figure 5-11. Then click the OK button to dismiss the dialog.

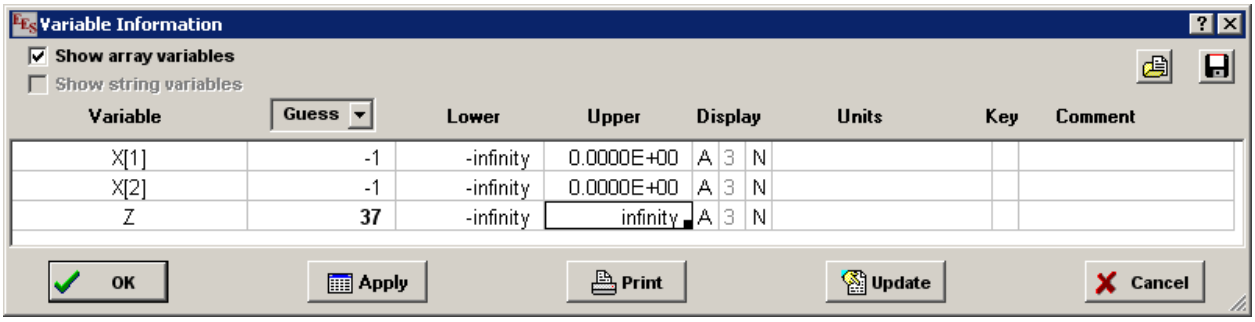


Figure 5-11: Variable Information dialog window with altered guess values and bounds.

Solve the equations (press F2) and you will see that EES has converged to the alternate solution: X[1] = -4.727 and X[2] = -3.829. The solution that EES converges to is primarily controlled by the guess values and, to a lesser extent, the bounds specified for each variable. Some equations sets cannot be solved at all unless good guess values are provided.

Setting Guess Values and Limits using Variables

The guess values and bounds are normally entered as numerical values, but they also can be provided with the value of another variable. As an example, the variable X_g is entered in the Equation Window and specified to be -1:

```
Z=37
X[1]^2+X[2]^2=Z
X[1]/X[2]=1.2345
X_g=-1
```

Bring up the Variable Information Window and set the guess value for X[1] and X[2] to be X_g, as shown in Figure 5-12. Click the OK button.

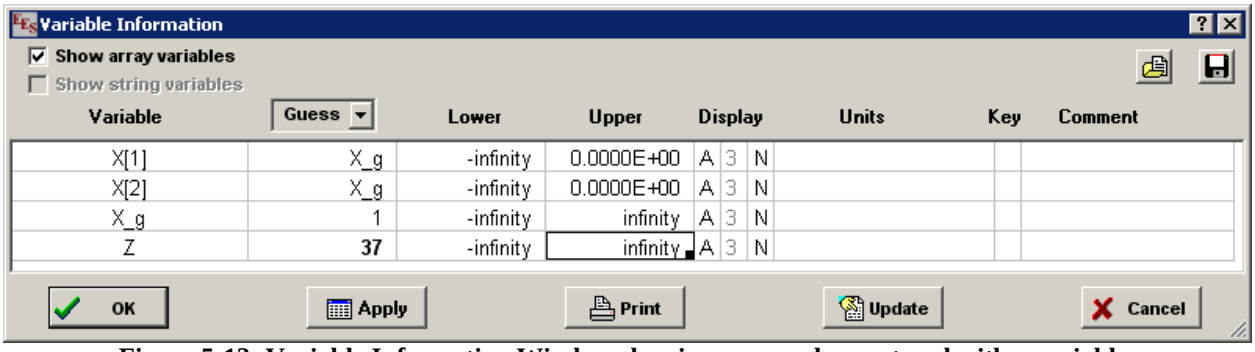


Figure 5-12: Variable Information Window showing guess values entered with a variable.

Solve the equations and you should find that EES has converged to a solution in which both X[1] and X[2] are negative. Change X_g to 1 in the Equations Window and solve again. EES will converge to values of X[1] and X[2] that are both positive. The guess values and lower and upper limits can also be entered using variables that are specified in the Diagram Window (discussed in Chapter 15) or in a Parametric Table. This capability is very useful in some problems where the guess values that you need to obtain the desired solution change depending on other information in the problem. For example, the equation that provides the Mach number in a converging-diverging nozzle given its cross-sectional area has two roots. In the converging section, the correct root leads to a Mach number that is less than one. In the diverging section, the other root should be selected, which leads to a Mach number that is greater than one. In order to control the solution process, a guess for Mach number that is less than one in the converging section and greater than one in the diverging section could be entered in a column of a Parametric Table and then used to control which root is calculated by EES.

The Professional version of EES extends this capability by allowing guess values and limits to be specified with equations that involve variables defined in the Equations Window and numerical

constants. As a trivial example, the guess values for X[1] and X[2] could have been entered as X_g/2+0.01.

Arrays in the Variable Information Window

The simple example used in this discussion only involves two array variables, X[1] and X[2]. However, an EES program may often involve many array variables. Each array variable is a separate entity and each may have its own guess value and limits. It is often the case that the guess value and limits should be the same for all variables in the array. De-selecting the Show array variables control at the upper left of the Variable Information dialog window will toggle the collapsed display of the array variables. In the collapsed form, each array is displayed on one row of the Variable Information dialog, as shown in Figure 5-13.

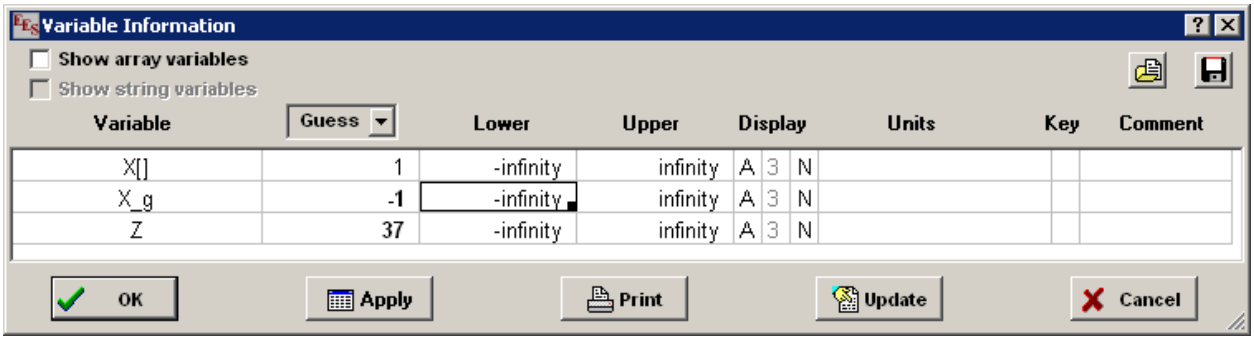


Figure 5-13: Variable Information Window showing collapsed view for array variables.

One advantage of the collapsed array display option is that it reduces the number of rows in the table so that other, non-array variables can be more easily located. However, another advantage is that it facilitates changing all of the guess values or limits in the array to have the same value. For example, if the guess value for X[] is changed from 1 to -1 then the guess values of all variables in the X array (X[1] and X[2], in this case) will be set to -1. You can confirm this change by selecting the Show array variable control so that all of the variables in the X[] array are again visible. Note that the change only occurs if the displayed value is *changed* while in the collapsed view. If the displayed value already shows the desired value, but not all of the array elements are set to this value, then it is necessary to change the displayed value to some other value and then change it back to its original value. This action will ensure that all of the array variables are changed to have the same guess value. The lower and upper bounds, display format, and units can all be changed for array variables in the same manner.

Changing Variable Names

A useful capability of the Variable Information dialog is that it can be used to change the name of a variable or an array. For example, suppose you wish to change the name of every array element X[i] to Y[i]. This change can be most easily accomplished by changing X[ ] in Figure 5-13 to Y[ ]. Perhaps you also wish to change the name of X_g to Y_g. Make the changes and then click the OK button. You will next see the confirmation dialog in Figure 5-14.

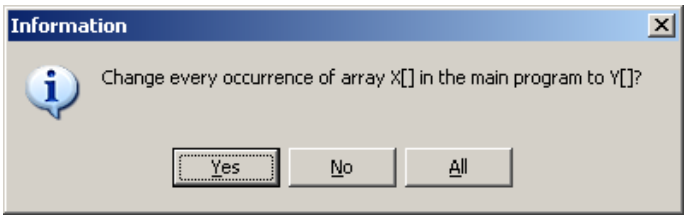


Figure 5-14: Conformation dialog for changing variable names.

If you click the Yes button, EES will make the indicated change in the variable name and proceed to display the next requested variable name change. If you click the All button, EES will make all of the changes that you requested with no further confirmations. All variable information, including the guess value, lower and upper bounds, and units, is assigned to the new variable name. Also, if the variable appeared in a Parametric Table then its name will be changed there as well. You could of course use the Find/Replace commands in the Search menu to change variable names. For example, you could change every occurrence of X[to Y[. However, EES would then assume then that Y[i] array elements are new variables. The guess values, limits, units and other information that were assigned to the X[i] array elements would not be assigned the Y[i] variables. Also, no changes will be made to any Parametric Table (i.e., if X[1] was a variable in a Parametric Table then it would still appear in the Parametric Table after the Find/Replace process is completed).

Variable Information Files

The Professional version of EES provides the ability to save and read Variable Information (.var) files using the two buttons at the upper right of the Variable Information dialog window, as shown in Figure 5-15.

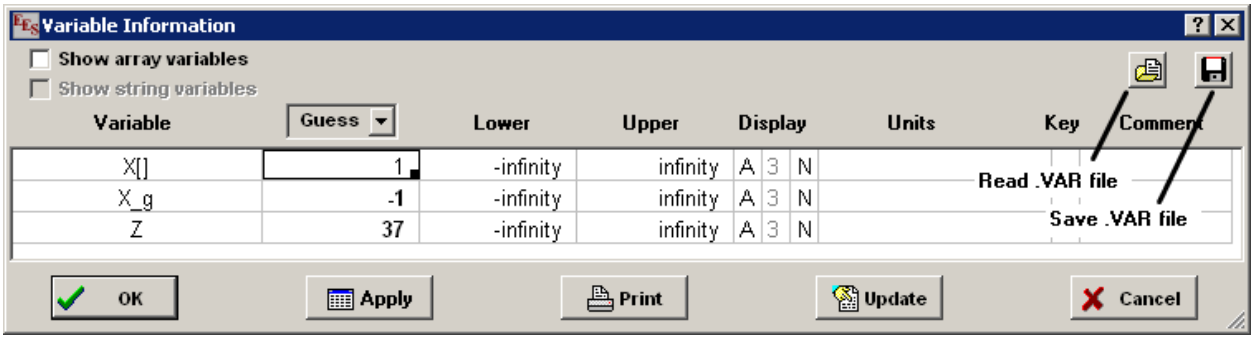


Figure 5-15: Variable Information Window showing the buttons to read and save .VAR files.

Clicking the Save button will bring up the Save Variable Information dialog window where you can specify a file name having a .var file name extension. After confirmation, EES will write all of the information for each variable that is shown in the Variable Information dialog window to the specified file. The .var file is a text file that can be opened, viewed and edited in any editor, such as Notepad. For example, the .var file created from the Variables Information Window in Figure 5-15 is shown in Figure 5-16. Units were not assigned to any of the variables in Figure 5-15, but if they were, the units would also appear in the .var file. The same is true for the comments.

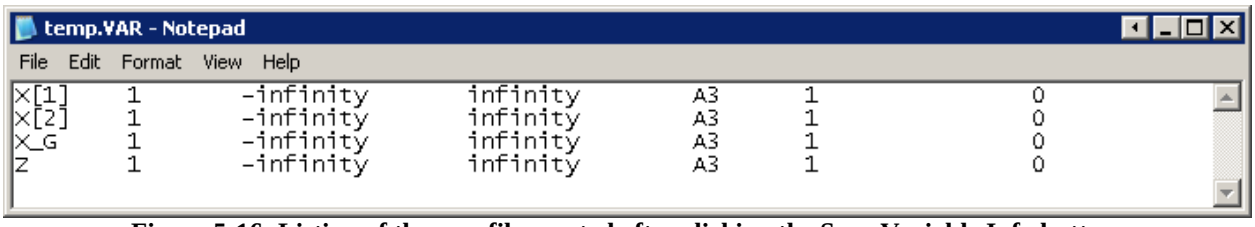


Figure 5-16: Listing of the .var file created after clicking the Save Variable Info button.

Selecting the Read button will bring up a standard file open dialog that allows navigation to a .var file that has been previously saved. Reading a .var file will assign all of the attributes (i.e., guess value, bounds, units, etc.) to every variable that has a name that exists in the .var file, including variables that are in functions, procedures subprograms and modules as well as in the main program. Variable Information (.var) files provide a more general and more powerful way of saving default information for variables than is possible with the Default Variable dialog.

Using Arrays as Guess Values and Limits

It is possible to assign different values of the guess values or bounds to each element in an array using a different array that is setup just for this purpose. This capability can be useful if you want to force EES to converge to different solutions to an equation or set of equations that have multiple roots. For example, consider the equation:

$$\tan(\zeta_i) = \frac{Bi}{\zeta_i} \tag{5-38}$$

Equation (5-38) is the eigencondition for the separation of variables problem related to transient conduction in a plane wall, as discussed in Nellis and Klein (2009). The symbol *Bi* indicates the Biot number and ζ_i indicates the i^{th} eigenvalue. Equation (5-38) has multiple roots and each successive root is referred to as an eigenvalue; in order to solve the problem, it is necessary to identify an arbitrary number of sequential roots.

The EES code below evaluates both the left (LHS) and right (RHS) sides of Eq. (5-38):

```
Bi=1                    "Biot number"
LHS=tan(zeta)           "left hand side of eigencondition"
RHS=Bi/zeta             "right hand side of eigencondition"
```

Create a Parametric Table in which the variable zeta is varied from just above 0 (to avoid a division by zero error when calculating the value of RHS) to 14.137 (approximately $9\pi/2$). Generate the plot shown in Figure 5-17. Notice that the left and right sides of the eigencondition are equal (i.e., the equation is satisfied) at multiple values of ζ. Further examination of Figure 5-17 shows that these intersections occur in well-defined intervals. The first root of Eq. (5-38) lies in the range $0 < \zeta_1 < \pi/2$, the second root lies in the range $\pi < \zeta_2 < 3\pi/2$, and so on. In general, the i^{th} eigenvalue must lie in the range:

$$\pi(i-1) < \zeta_i < \pi(i-1) + \frac{\pi}{2} \tag{5-39}$$

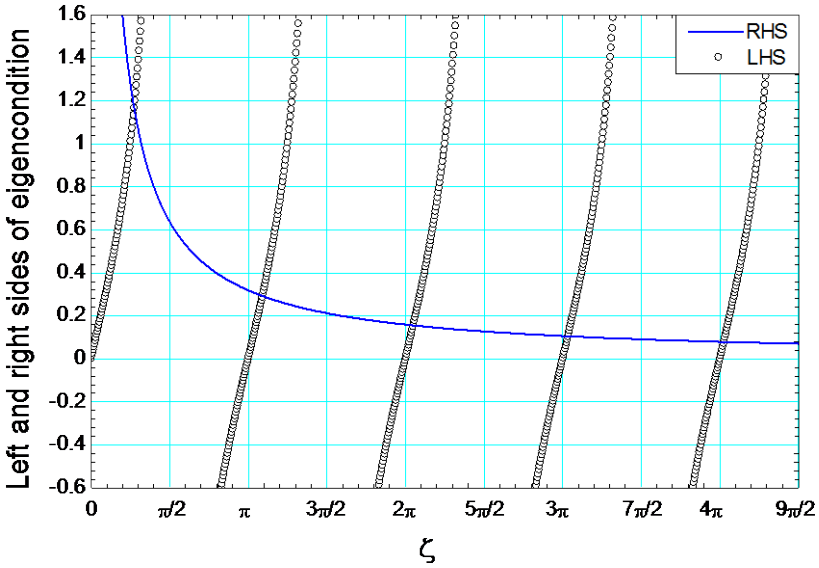


Figure 5-17: Right and left hand sides of the eigencondition, Eq. (5-38).

EES can be used to generate an array of eigenvalues by controlling the guess values and limits of each of the elements using arrays that are defined just for this purpose. Use a Duplicate statement to generate an array of lower and upper bounds appropriate for each eigenvalue.

```
Bi=1                                  "Biot number"
{LHS=tan(zeta)                        "left hand side of eigencondition"
RHS=Bi/zeta                           "right hand side of eigencondition"}
N=10                                  "number of eigenvalues to identify"
duplicate i=1,N
    lowerbound[i]=pi*(i-1)            "lower bound"
    upperbound[i]=pi*(i-1)+pi/2       "upper bound"
end
```

Generate an array that contains reasonable guess values for each eigenvalue; here, the guess is taken to be the average of the upper and lower bounds.

```
duplicate i=1,N
    guess[i]=pi*(i-1)+pi/4            "guess"
end
```

Use a Duplicate statement to solve Eq. (5-39) multiple (N) times.

```
duplicate i=1,N
    tan(zeta[i])=Bi/zeta[i]           "eigencondition"
end
```

Solving the EES code leads to the Arrays Table shown in Figure 5-18(a); note that only the first eigenvalue was identified for every value of the array zeta[]. In order to control the solution process, open the Variable Information Window and de-select the Show array variables option. Enter the arrays guess[], lowerbound[], and upperbound[] in the appropriate columns for the array

zeta[] as shown in Figure 5-19. Select OK and solve the EES code again in order to obtain the Arrays Table shown in Figure 5-18(b). Notice that each value of the array zeta[] now corresponds to each successive eigenvalue.

Arrays Table

Main

Sort	1 $guess_i$	2 $lowerbound_i$	3 $upperbound_i$	4 ζ_i
[1]	0.7854	0	1.571	0.8603
[2]	3.927	3.142	4.712	0.8603
[3]	7.069	6.283	7.854	0.8603
[4]	10.21	9.425	11	0.8603
[5]	13.35	12.57	14.14	0.8603
[6]	16.49	15.71	17.28	0.8603
[7]	19.63	18.85	20.42	0.8603
[8]	22.78	21.99	23.56	0.8603
[9]	25.92	25.13	26.7	0.8603
[10]	29.06	28.27	29.85	0.8603

(a)

Arrays Table

Main

Sort	1 $guess_i$	2 $lowerbound_i$	3 $upperbound_i$	4 ζ_i
[1]	0.7854	0	1.571	0.8603
[2]	3.927	3.142	4.712	3.426
[3]	7.069	6.283	7.854	6.437
[4]	10.21	9.425	11	9.529
[5]	13.35	12.57	14.14	12.65
[6]	16.49	15.71	17.28	15.77
[7]	19.63	18.85	20.42	18.9
[8]	22.78	21.99	23.56	22.04
[9]	25.92	25.13	26.7	25.17
[10]	29.06	28.27	29.85	28.31

(b)

Figure 5-18: Arrays Table (a) without controlling the guess values and limits of the array zeta[] and (b) after using the arrays guess[], lowerbound[], and upperbound[] to control the solutions to the array zeta[].

Variable Information

Show array variables

Show string variables

Variable	Guess	Lower	Upper	Display	Units	Key	Comment
Bi	1	-infinity	infinity	A 3 N			
guess[]	0.7854	-infinity	infinity	A 3 N			
lowerbound[]	0	-infinity	infinity	A 3 N			
N	10	-infinity	infinity	A 3 N			
upperbound[]	1.571	-infinity	infinity	A 3 N			
zeta[]	guess[]	lowerbound[]	upperbound[]	A 3 N			

OK Apply Print Update Cancel

Figure 5-19: Variable Information Window showing the use of arrays to control the solution process.

Default Variable Information

In many engineering problems, variables that start with a certain letter tend to correspond to the same physical quantity. For example, variables that start with the letter T may all refer to temperatures and those that start with the letter P refer to pressures. With this in mind, EES provides a method of assigning a common set of characteristics to variables that start with the same letter. Select Default Info from the Options menu to access the Default Variable Information dialog, shown in Figure 5-20.

The Default Variable Information dialog allows you to set a common guess value, limits, and units for all variables that begin with the same letter. For example, if all variables that begin with the letter T represent temperatures then you may want to set the units of these variables automatically to K and provide more reasonable guess values (e.g., 300 K) and limits (e.g., 10 K to 1000 K), as shown in Figure 5-20. These settings can be stored in the preferences file by selecting Store so that they remain in effect for future EES sessions.

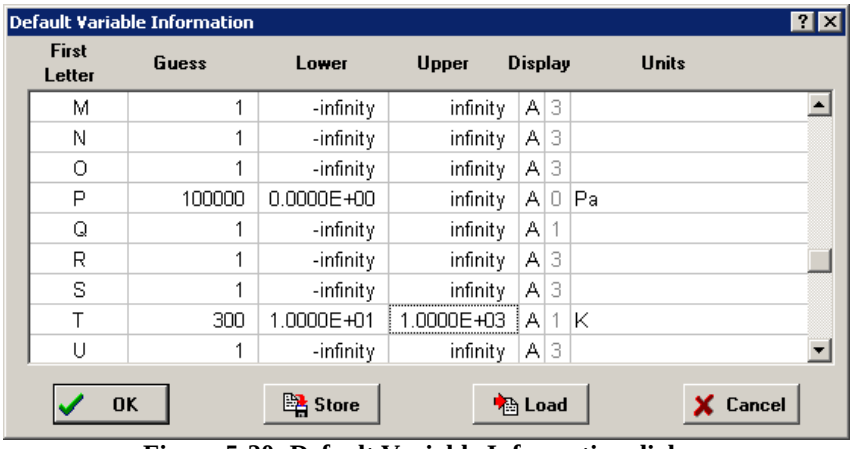


Figure 5-20: Default Variable Information dialog.

5.4 Debugging Techniques

The numerical methods built into EES are robust and powerful. Nevertheless, you will likely encounter situations where your set of equations does not solve. The most common reason for the equations not solving is that there is some error in the equations that you have entered. It could be that there is a unit conversion problem or a sign error. Even one small problem may be sufficient to prevent EES from being able to solve the equations that have been entered. On the other hand, sometimes even a properly formulated set of equations may not solve if one or more of the equations are non-linear so that they depend on the guess values that you supply

EES is not a programming language. It is an equation-solving application. It does not solve the equations in the order in which they are entered into the Equations Window, but rather reorders them as it sees fit in an attempt to solve the equations in an optimal manner, as explained in Section 5.1. When EES is able to solve a set of equations, it does so efficiently. However, it is relatively difficult to identify the cause of a problem when EES does not solve your equation set because you do not directly control the solution process. This section provides a number of techniques that should help you coax a stubborn set of equations to a solution in EES.

Effective Use of EES

Although EES allows equations to be entered in any order, there are good reasons to enter the equations in a logical order. Whenever possible, equations should be entered in a sequential manner so that an intermediate solution can be obtained after each equation (or at least every few equations) is entered. This methodology is similar to the way you would solve the problem by hand or using a conventional programming language. In a sequential solution, each new equation only uses information that was stated or determined from preceding equations. In a large, coupled set of equations it may help to enter a temporary equation that effectively provides a guess for a variable that has yet to be determined. The temporary equation is later removed when additional information is entered.

There are several reasons for entering equations in a sequential order. First, it allows your program to be debugged as it is being written. It is a frustrating to enter a large set of equations and then finally hit Solve only to see EES fail to converge. It will be difficult to debug such a large EES program. It is much easier to enter equations one at a time and verify that EES can solve the set of equations that results after each equation is entered. When problems are encountered, they can immediately be isolated to the last equation that was entered. This method is good engineering practice. When something goes wrong with an experiment or device, an experienced engineer will immediately try to isolate the problem by testing each sub-system of the device alone.

There is another advantage to using the sequential approach discussed in the previous paragraph. EES solves a set of non-linear equations using an iterative approach; the approach starts from an initial point that is characterized by a set of "guess" values for each variable and iteratively improves the solution. The solution process is easier if the guess values for each variable are close to the final solution. Obtaining a series of intermediate solutions usually allows the user to easily obtain a very good set of guess values.

The Update Guesses Command and Directive

The process of updating guess values as you proceed through the solution is facilitated by the use of the Update Guesses command in the Options menus. This command can also be initiated using the Update Guesses speed button () on the speed button bar that is displayed below the main menu bar.

Selecting the Update Guesses command will cause the guess values of all variables to be set to their most recently calculated values. Updating the guess values can also be accomplished by pressing the Update button in the Variable Information Window or by placing the directive $UpdateGuesses in the Equations Window. The $UpdateGuesses directive differs slightly from the Update Guesses menu command. The Update Guesses menu command updates the guess values throughout the entire EES program (i.e., in the main body and all subprograms) whereas the $UpdateGuesses directive only operates within the scope (i.e., the main program or subprogram) in which it is located. Subprograms are discussed in Chapter 10. In any case, the guess values can only be updated if the current set of equations was successfully solved. If the equation set has not been successfully solved then the Update Guesses command and speed button will be disabled.

The following example illustrates the sequential solution technique and the use of the Update Guesses command. The problem is to determine the temperature, T, of a surface that is experiencing radiation heat transfer with two surfaces, one at temperature T_H = 1000 K and the other at T_L = 300 K. The areas exposed to the high and low temperature surfaces are A_H = 1 m^2 and A_L = 0.5 m^2, respectively. The rate of radiation heat transfer to the surface from T_H is given by:

$$\dot{Q}_H = A_H \, \sigma \left(T_H^4 - T^4 \right) \tag{5-40}$$

where σ is the Stefan-Boltzmann constant (5.67×10^{-8} W/m^2-K^4); note that this is a built-in constant in EES (sigma#), as discussed in Section 1.9. The rate of radiation heat transfer from the surface to T_L is given by:

$$\dot{Q}_L = A_L \, \sigma \left(T^4 - T_L^4 \right) \tag{5-41}$$

The surface is at steady state, therefore an energy balance provides:

$$\dot{Q}_L = \dot{Q}_H \tag{5-42}$$

Proceeding in a sequential manner, we first enter the known information.

```
$UnitSystem SI K Pa J

"known information"
sigma=sigma#                    "Stefan-Boltzmann constant"
A_L=0.5 [m^2]                   "area of low temperature surface"
T_L=300 [K]                     "temperature of low temperature surface"
A_H=1.0 [m^2]                   "area of high temperature surface"
T_H=1000 [K]                    "temperature of high temperature surface"
```

It would be good practice to solve the equations entered up to this point in order to ensure that there are no typographical errors. We could next try to directly enter and solve the set of equations in Eqs. (5-40) through (5-42).

```
Q_dot_H=A_H*sigma*(T_H^4-T^4)    "radiation from high temperature surface"
Q_dot_L=A_L*sigma*(T^4-T_L^4)    "radiation to low temperature surface"
Q_dot_H=Q_dot_L                  "steady-state requirement"
```

EES will not converge when you try to solve these and instead it will display the error message shown in Figure 5-21.

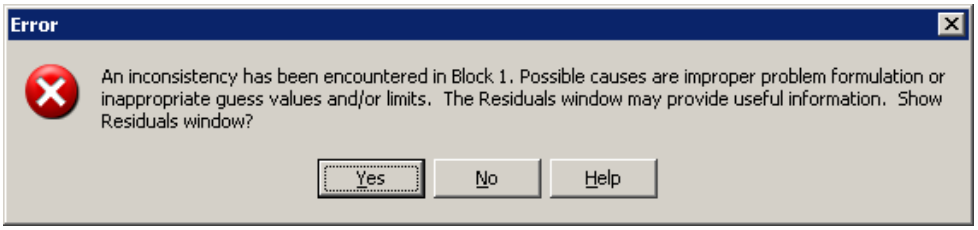


Figure 5-21: Error message displayed by EES when you try to solve Eqs. (5-40) through (5-42).

The error message directs us to view the Residuals Window. Clicking the Yes button will cause the Residuals Window to display, as shown in Figure 5-22.

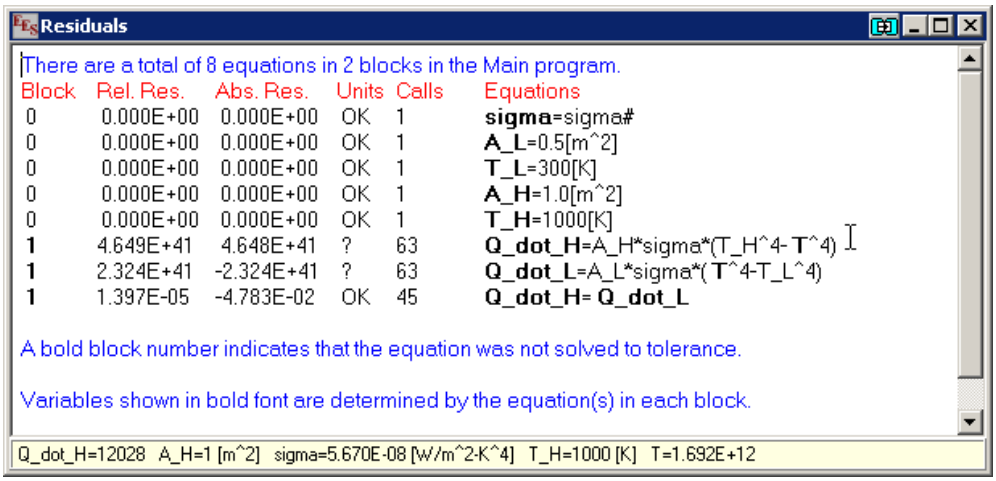


Figure 5-22: Residuals window showing Block 1 did not converge.

The Residuals Window shows that the convergence problem occurs in the three equations that together form Block 1. The block number is shown in bold font, indicating that the equations were not solved to the specified tolerance. Also, the values of the relative residuals for the equations in block 1 are much greater than the tolerance value specified in the Stop Criteria dialog (Options menu). In a larger problem, this information may be useful as it isolates the location of the problem. However, in this case, we already knew that the last three equations we entered were the source of the convergence problem.

Entering Eqs. (5-40) through (5-42) simultaneously caused this convergence problem. Let's take an alternative approach where the equations are entered one at a time. Delete the last three equations in the Equations Window. In order to take a sequential (rather than simultaneous) approach, we enter Eq. (5-40) and then solve it. Next, enter Eq. (5-41) and solve. Finally, enter Eq. (5-42) and solve. However, this approach appears to fail for this example because Eq. (5-40) involves two unknowns, $\dot{Q}_H$ and T. In order to carry on, it is best to enter a temporary equation that provides a guess for the unknown T. We know that the temperature of the surface must lie between T_L and T_H; a reasonable guess is therefore the average.

```
T=(T_H+T_L)/2                    "a temporary equation"
```

This equation is clearly temporary and it must be removed eventually in order to obtain the correct solution; this can be emphasized by highlighting the equation, as shown above. (To highlight the equation, select it and then click the right mouse button.) With a temporary value of T specified, it becomes possible to enter Eq. (5-40) and solve the resulting set of equations:

```
Q_dot_H=A_H*sigma*(T_H^4-T^4)    "radiation from high temperature surface"
```

The units for the variables can be entered and checked and the solution debugged if necessary. Equation (5-41) can be entered next and the new equation set solved.

```
Q_dot_L=A_L*sigma*(T^4-T_L^4)    "radiation to low temperature surface"
```

Although the equations solved, the solution is not correct because we entered an arbitrary value for *T*. However, the solution does serve a purpose. The current values for the variables Q_dot_L, Q_dot_H, and T are very good guess values; much better than the default guess values that would otherwise be used. Select the Update Guesses command from the Options menu or the speed button tool bar. After the confirmation, EES will assign the current values of all variables to be the guess values for those values. For example, the variable T initially had a guess value of 1 K. Now its guess value will be 650 K. Next, delete (or comment out) the temporary equation that specified T and enter the steady-state requirement given by Eq. (5-42).

```
{T=(T_L+T_H)/2          "temporary equation"}
Q_dot_H=Q_dot_L         "steady-state requirement"
```

Now, when you try to solve the equation set there will be no convergence problems. The correct value of *T* is 904.5 K, which is still pretty far from the guess value but close enough to allow the equations to converge.

This example illustrates the basic method of entering equations sequentially, using temporary equations as needed to supply guesses, and applying the Update Guesses command. There are other ways to make EES solve these equations, such as by adjusting guess values at the start of the solution process. However, the sequential method is, in the opinion of the authors, the best way to ensure that you are in control of the solving process with EES.

The Residuals Window as a Debugging Tool

The information in the Residuals Window is useful in coaxing a stubborn set of equations to converge. An examination of the residuals will indicate which equations have been solved by EES and which have not. For example, suppose we had not followed the sequential approach recommended in the previous section and instead entered Eqs. (5-40) to (5-42) simultaneously, as well as an additional equation that converts the surface temperature to Celsius.

Edit the Equations Window so the following equations appear. Select the Reset Guesses command in the Options menu to set the guess values of all variables back to their default values.

```
$UnitSystem SI K Pa J

"known information"
sigma=sigma#                        "Stefan-Boltzmann constant"
A_L=0.5 [m^2]                       "area of low temperature surface"
T_L=300 [K]                         "temperature of low temperature surface"
A_H=1.0 [m^2]                       "area of high temperature surface"
T_H=1000 [K]                        "temperature of high temperature surface"

Q_dot_H=A_H*sigma*(T_H^4-T^4)       "radiation from high temperature surface"
Q_dot_L=A_L*sigma*(T^4-T_L^4)       "radiation to low temperature surface"
Q_dot_H=Q_dot_L                     "steady-state requirement"

T_C=converttemp(K,C,T)              "temperature, in C"
```

These equations will not solve with the default guess values. The Residuals Window will appear as shown in Figure 5-23.

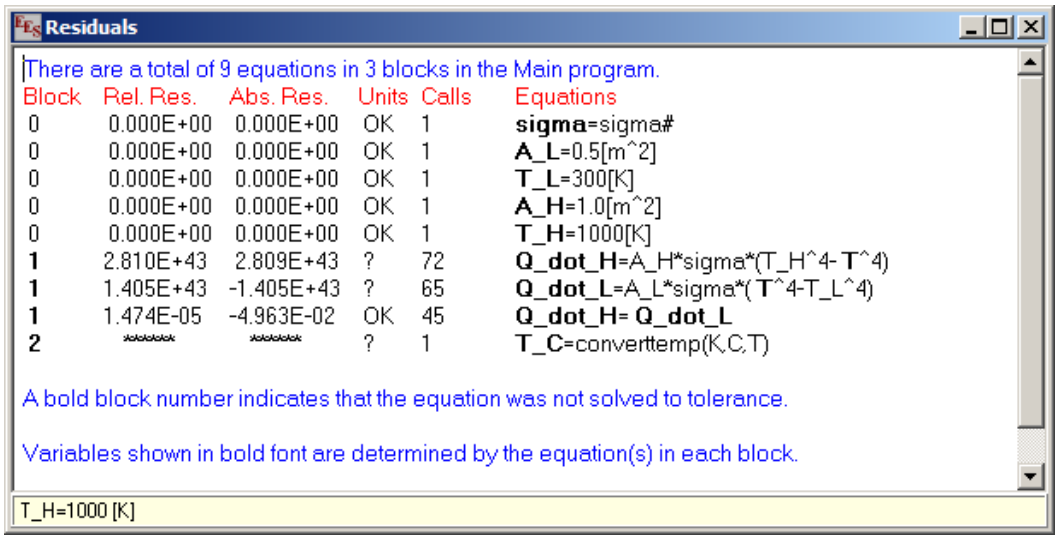


Figure 5-23: Residuals window showing **** for blocks in which calculations were not attempted.**

Note the ****** in the residuals columns for Block 2; these stars indicate that calculations for this block (and any following blocks) were not attempted. The reason that the calculations were not attempted is that calculations failed in the preceding block, Block 1, for the reasons explained in the previous section. The variables that EES was trying to solve for in Block 1 are shown in bold font in the equations for this block. This information quickly indicates the source of the problem within what might be a large set of equations. Debugging efforts can be directed to the equations in Block 1. For example, the equations in the block that did not solve should be checked to ensure that they are properly entered. It may be necessary to adjust the guess values for the variables appearing in bold font. Note that Figure 5-23 also shows that there are unit conversion problems with some of the equations in block 1. These unit conversion problems should be eliminated before further attempts are made to solve the equation set, as they could result from errors in the equations that may be the source of the problem. In this example, the unit conversion errors occurred because the units for the variables Q_dot_H and Q_dot_H were not specified.

Normally, the block numbers appear in sequential order. When one or more equations are missing, EES will skip a block number at the point in which it encounters this problem. The equations in the following blocks should be carefully reviewed to determine whether they are correctly and completely entered. There is clearly a problem somewhere in the equation set if EES skips a block number.

Common Problems

The most common reason that a set of equations will not solve is that there is an incorrect equation (which can often be identified by inconsistent units) or inappropriate guess values. These problems are usually detected early in the process of entering the equation provided that the sequential process recommended earlier is followed. However, there are some common problems that do not show up in the Residuals Window and may not be easy to identify.

One common problem that prevents EES from converging to a solution is the presence of a redundant equation. An example of a redundant equation is demonstrated in the following simple example.

```
z=50
x+y=0.022
z*x+z*y=0.022*z
```

EES may or may not obtain a solution to this equation set, depending on stopping criteria. However, even if a solution is obtained it will not be unique. The equation set consists of three equations in three unknowns (x, y, and z) and therefore it would appear that the equations are solvable. However, closer inspection shows that second and third equations are actually identical; the third equation is equal to the second equation multiplied by the variable z. The third equation provides no new information and is redundant. When this equation is removed, another relation between x and y must be provided if the equation set is to be solved.

A somewhat more subtle example of a redundant equation can occur when using the built-in property functions, discussed in Chapter 4. This situation is demonstrated in the following equations.

```
$UnitSystem Mass SI K Pa J

P=100000 [Pa]
h=Enthalpy(Steam,T=T, P=P)
T=Temperature(Steam, h=h, P=P)
```

EES will not solve this equation set, although it appears to be simple and complete; there are three equations in the three unknown variables P, h, and T. However, note that the last two equations both rely on fundamentally the same relationship between h, T, and P. The equations in the Enthalpy function that return the specific enthalpy as a function of temperature and pressure are the same as the equations in the Temperature function that return temperature given specific enthalpy and pressure. Therefore, one of the last two equations is redundant. EES may return a solution, depending on the unit system and stopping criteria, but the result will not be correct. Some other equation relating specific enthalpy and temperature must be entered in place of one of the last two equations.

Another common problem is related to the fact that the properties of pure fluids can not be obtained from temperature and pressure in the two-phase regime. Consider the following example:

```
$UnitSystem SI K Pa J mass

h=1000000 [J/kg]
P=100000 [Pa]
h=Enthalpy(Steam,T=T,P=P)
```

EES will not be able to solve these equations, although they seem simple enough. The error message shown in Figure 5-24 results when you try to solve these equations.

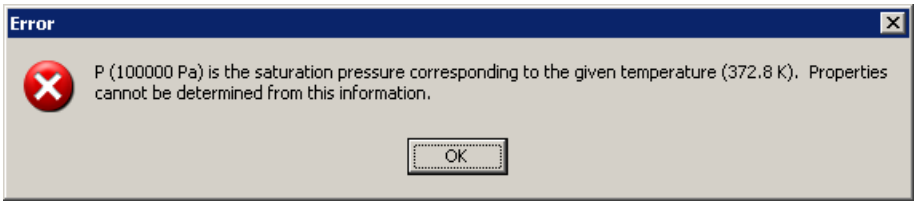


Figure 5-24: Error message that results when trying to determine properties of two-phase state using temperature and pressure.

The problem in this case is that the state defined by the specific enthalpy and pressure is in the two-phase liquid-vapor regime. In this regime, temperature and pressure are not independent. Consequently, it is not possible to determine the temperature with the equation that uses the Enthalpy function called with pressure and temperature used as inputs. The problem would solve if the last equation were replaced with one that uses the Temperature function, as shown below.

```
{h=Enthalpy(Steam,T=T,P=P)}
T=Temperature(Steam, h=h, P=P)
```

The resulting temperature is 372.8 K.

The $Trace Directive

The $Trace directive is a powerful, but rarely needed, debugging tool. It produces a table of all of the intermediate values of specified variables in the Equation Window that occur during the iteration process. In some cases, this information can be useful to identify why a set of equations is not converging.

The format of the $Trace directive is:

```
$Trace /# X, Y, Z
```

The /# parameter in the $Trace directive is optional. The number following the / character (#) indicates the maximum number of iterations that will be recorded in the Trace Table. By default, this number is 1000. The variables X, Y, and Z are the variables in the main section of the EES program that will appear in the Trace Table. There is no specific limit on the number of variables that can be traced; however, it is best to confine the list to those variables that are suspected of causing a convergence problem. Variables that are involved in functions, procedures, modules, or subprograms can not be traced.

The $Trace directive will place the intermediate values of the specified variables in a Lookup Table having the name TRACE. If a Lookup Table having the name TRACE already exists, then it will be deleted at the start of the calculations. If you wish to keep the previous results of a Lookup Table named TRACE then the table must be renamed before running the program.

The $Trace directive will be created when the Solve command is issued. It will have no effect when used with Solve Table or other commands in the Calculate menu. It is best to place the $Trace directive at the top of the program so that it is processed before any of the specified variables appear in equations.

As an example, the $Trace directive is used to debug the radiation problem that did not converge with the default guess values.

```
$Trace /5000 T, Q_dot_H, Q_dot_L

sigma=sigma#                          "Stefan-Boltzmann constant"
A_L=0.5 [m^2]                         "area of low temperature surface"
T_L=300 [K]                           "temperature of low temperature surface"
A_H=1.0 [m^2]                         "area of high temperature surface"
T_H=1000 [K]                          "temperature of high temperature surface"
Q_dot_H=A_H*sigma*(T_H^4-T^4)         "radiation from high temperature surface"
Q_dot_L=A_L*sigma*(T^4-T_L^4)         "radiation to low temperature surface"
Q_dot_H=Q_dot_L                       "steady-state requirement"
```

Select Solve and you will again find that the program does not converge. However, the $Trace directive has resulted in the generation of a Lookup Table called TRACE. The first 10 rows of the Lookup Table are shown in Figure 5-25. The trace process clearly shows the wild variations in the temperature that occur with each iteration.

Lookup Table

TRACE

Paste Special	1 Iteration	2 T [Blk 1]	3 $\dot{Q}_H$ [Blk 1]	4 $\dot{Q}_L$ [Blk 1]
Row 1	1	1	1	1
Row 2	2	1.672E+11	18733	18733
Row 3	3	8.361E+10	9367	9367
Row 4	4	1.254E+11	14050	14050
Row 5	5	1.045E+11	11708	11708
Row 6	6	1.149E+11	12879	12879
Row 7	7	1.097E+11	12294	12294
Row 8	8	1.123E+11	12586	12586
Row 9	9	1.110E+11	12440	12440
Row 10	10	0.6607	0.2758	0.6707

Figure 5-25: Excerpt of the Lookup Table produced by the $Trace directive.

Equations are solved in blocks and the block structure can be examined in the Residuals Window. The block for each variable in the Trace Table is shown below the variable name (see Figure 5-25). Note that iterations for variables that are in different blocks cannot be directly compared as they are not occurring at the same time.

References

Tarjan, R. “Depth-First Search and Linear Graph Algorithms,” *SIAM J. Comput.*, 1, 146-160, (1972)

6 OPTIMIZATION

To this point we have talked about using EES to solve problems that involve fixed inputs and result in a single solution. In Section 1.3, the use of Parametric Tables was presented as a method for carrying out a parametric study or one or more variables. Optimization is a process in which one or more variables are varied in order to minimize or maximize an objective function. Optimization is often the purpose of developing a model of a physical system as it allows identification of an optimal physical design or operating condition. This chapter discusses the powerful algorithms that are available in EES for accomplishing single- and multi-dimensional optimization. Optimization in EES is also useful for solving particularly stiff sets of nonlinear equations and for developing advanced curve fits.

6.1 One-Dimensional Optimization

Consider the energy recovery heat exchanger shown in Figure 6-1. Cold air from outdoors is pre-heated before it enters a furnace using an exhaust flow of hot air in a counterflow heat exchanger. The optimal size of the energy recovery heat exchanger balances the capital cost of the heat exchanger against the value of the energy saved.

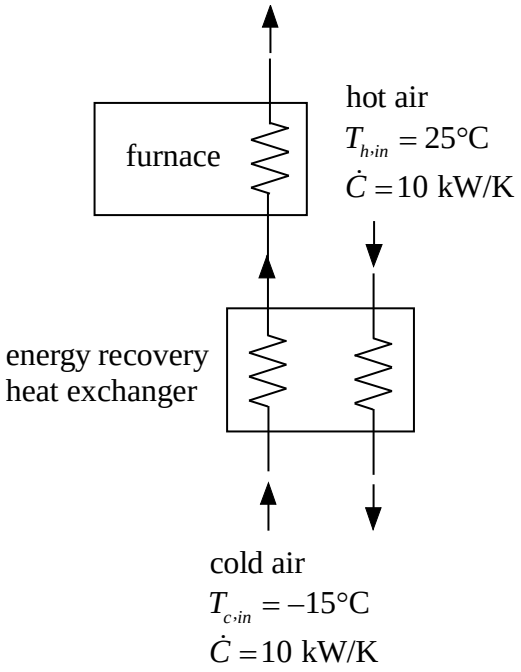


Figure 6-1: Energy recovery heat exchanger.

The capacitance rate (i.e., the product of mass flow rate and specific heat capacity) of the cold and hot air streams are the same, $\dot{C}$ = 10 kW/K. The cold air enters at $T_{c,in}$ = -15°C and the hot air enters at $T_{h,in}$ = 25°C. We would like to determine the conductance (*UA*) of the heat exchanger that is most economical, assuming that the system operates for 100 days/year for 5

years. The value of the energy saved is 0.73$/therm (a therm is an energy unit commonly used for natural gas that is equivalent to 105.5 MJ). The cost of the heat exchanger is given by:

$$HX_{Cost} = 2.5\left[\frac{\$\text{-K}}{\text{W}}\right]UA \tag{6-1}$$

The inputs are entered in EES:

```
$UnitSystem SI Mass J Pa K Rad

C_dot=10 [kW/K]*convert(kW/K,W/K)      "capacitance rate of both fluids"
T_c_in=ConvertTemp(C,K,-15 [C])        "cold inlet temperature"
T_h_in=ConvertTemp(C,K,25 [C])         "hot inlet temperature"
```

Implementing an Optimization Problem

The optimum conductance of the heat exchanger (*UA*) is to be determined in this optimization problem; the value of *UA* will be adjusted in order to maximize the amount of money saved by the system. However, it is best to initially specify the reasonable values of optimization variables in order to develop the model. These specifications allow the code to be debugged, units checked, etc., before adding another layer of complexity associated with the optimization process. In this case, a possible value of the conductance is specified:

```
UA_kWpK=15 [kW/K]                      "conductance, in kW/K - will be varied"
UA=UA_kWpK*convert(kW/K,W/K)           "conductance"
```

The number of transfer units for the heat exchanger is:

$$NTU = \frac{UA}{\dot{C}} \tag{6-2}$$

The effectiveness of a balanced counterflow heat exchanger is:

$$\varepsilon = \frac{NTU}{1+NTU} \tag{6-3}$$

(Note that this effectiveness relation in Eq. (6-3) could have been accessed using the Heat Transfer application library in EES, as discussed in Chapter 12.) The effectiveness is the ratio of the actual rate of heat transfer to the maximum possible rate of heat transfer:

$$\varepsilon = \frac{\dot{Q}}{\dot{Q}_{max}} \tag{6-4}$$

where

$$\dot{Q}_{max} = \dot{C}\left(T_{h,in} - T_{c,in}\right) \tag{6-5}$$

```
NTU=UA/C_dot                                "number of transfer units"
eff=NTU/(1+NTU)                             "effectiveness"
Q_dot_max=C_dot*(T_h_in-T_c_in)             "maximum possible heat transfer rate"
Q_dot=eff*Q_dot_max                         "actual heat transfer rate"
```

The heat exchanger cost is calculated according to Eq. (6-1). The total amount of energy saved is the product of the rate of heat transfer and the operating time. The value of this energy (neglecting the time value of money) is the product of the energy and the cost of the natural gas. The net savings (*NetSavings*) is the fuel cost savings less the heat exchanger cost.

```
HX_cost=2.5 [$-K/W]*UA                                "cost of heat exchanger"
time=5 [year]*100 [day/yr]*convert(day,s)             "operating time"
Energy=Q_dot*time                                     "energy savings"
Savings=0.73 [$/therm]*convert($/therm,$/J)*Energy    "money saved"
Net_savings=Savings-HX_cost                           "net savings"
```

Solving provides a net savings of $34,240 for the value of *UA* that was selected to develop the model. A Parametric Table is created that includes the variables UA_kWpK, HX_cost, Savings, and Net_savings. Figure 6-2 illustrates the results of the parametric study and shows that there is an optimal value of *UA* that maximizes net savings.

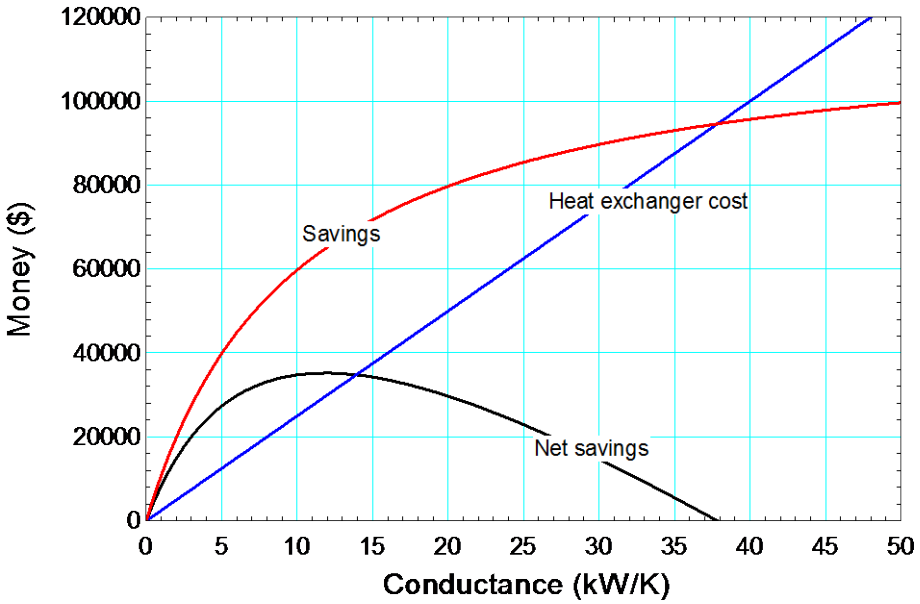


Figure 6-2: Savings, heat exchanger cost, and net savings as a function of the heat exchanger conductance.

Degrees of Freedom

The optimal value of *UA* can be determined using the built-in optimization algorithms in EES. In order to carry out optimization, EES requires at least one free parameter that can be varied. With *UA* set, the problem is completely specified. Therefore, if you select Min/Max from the Calculate menu in order to initiate the optimization, then the error message shown in Figure 6-3 will be displayed.

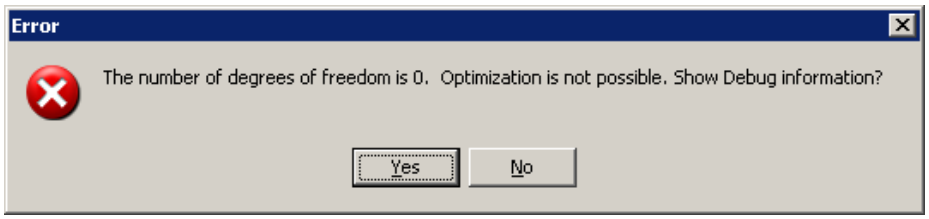


Figure 6-3: Error message.

It is necessary to comment out the equations that specify values of the optimization variables to proceed with the optimization. In this case, the equation that specifies *UA* is commented out.

```
{UA_kWpK=15 [kW/K]}              "conductance of heat exchanger, in kW/K - will be varied"
```

The Min/Max Dialog

Select Min/Max from the Calculate menu again to access the Find Minimum or Maximum dialog, shown in Figure 6-4.

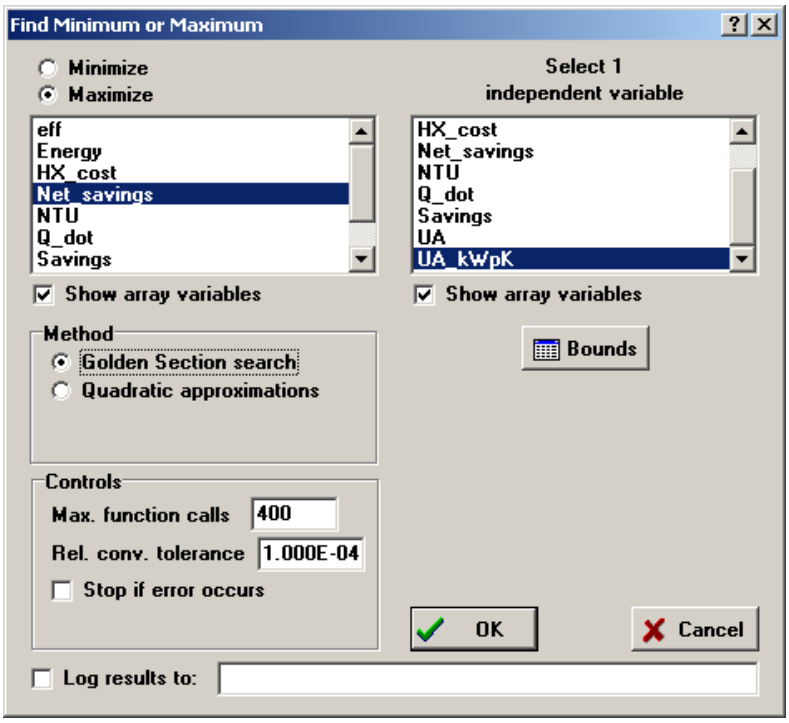


Figure 6-4: Find Minimum or Maximum dialog for one-dimensional optimization.

The top left portion of the window requires that you select the optimization target (i.e., the objective function) from the list of variables and specify whether the optimization target is to be minimized or maximized. In this case we wish to maximize the value of the variable Net_savings, as shown. The top right window allows you to select the independent (i.e., the optimization) variable(s). EES determines the number of free parameters associated with the equations and requires that this number of independent parameters be selected. For this problem (with *UA* commented out) there is only one free parameter, which we have chosen to be variable UA_kWpK.

The Professional version of EES provides an option to log results to a file. Clicking the Log results will provide the option of either logging all points or logging the best points as they are identified to a file selected by the user.

Stopping Criteria

The Controls box in the Find Minimum or Maximum dialog shown in Figure 6-4 allows the termination criteria for the optimization process to be specified. The process will terminate when either the maximum number of calculations are reached or the relative convergence tolerance is achieved. The relative convergence tolerance refers to the change in the optimization target normalized by its value between successive iterations. If the Stop if error occurs box is unchecked then EES will not terminate the optimization process if it attempts to solve the equations for a value of the optimization variable that results in an error. Rather, it will consider this value of the independent variable to be non-viable (and therefore not optimal) and move on to other values.

Guess Value and Bounds

EES requires that you set both upper and lower bounds and the guess value for each of the independent variables. To set these bounds, select Bounds from the Find Minimum or Maximum dialog in order to access the Variable Information window for the independent variable(s). For this problem, there is only one independent variable, UA_kWpK, and so the dialog appears as shown in Figure 6-5 with reasonable limits and guess values applied.

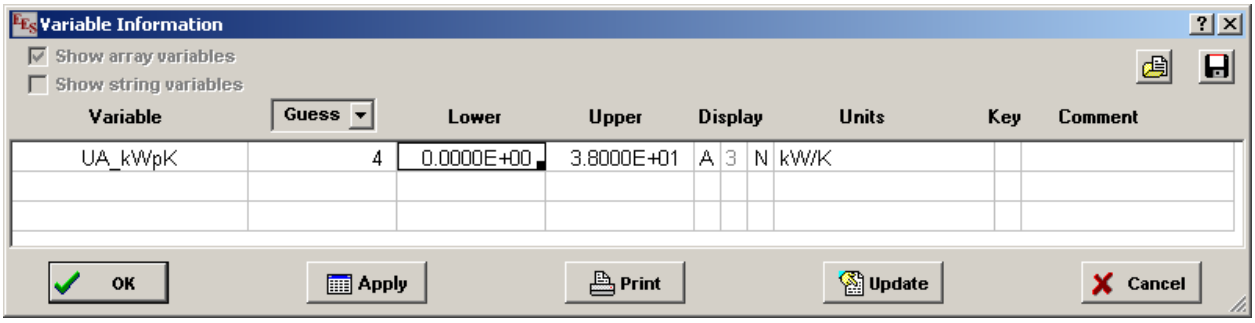


Figure 6-5: Guess values and bounds for variable UA_kWpK.

Select OK in order to return to the Find Minimum or Maximum dialog. All that remains is to select the optimization method. In this problem there is only one free parameter, therefore a one-dimensional optimization will be accomplished using either of the two methods shown under the heading Method: Golden Section search or Quadratic approximations. The methods required for a multi-dimensional optimization are different and are discussed in Section 6.2.

The Golden Section Search

The Golden Section search operates by successively narrowing the range in which the extremum (minimum or maximum value) is known to exist. The process starts by evaluating the objective function (*NetSavings*) at the lower and upper limits of the independent variable. For this problem, these limits are UA_1 = 0 W/K and UA_2 = 38 W/K. The objective function is found to be near zero at these two points.

Next, two test points are located within the range. The first test point, identified as point 3 in Figure 6-6, is located a distance of g (UA_2 - UA_1) from the left edge, where g is the Golden Ratio equal to 0.6182. The range of *UA* is initially 38 W/K so point 3 is located at *UA* = 23.5 W/K. The second test point is located at a value of *UA* that is g (UA_2 - UA_1) from the right edge which is *UA* = 14.5 W/K. The objective function (*NetSavings*) is evaluated at points 3 and 4. Point 4 exhibits a larger value of *NetSavings*. The result is that the range of *UA* containing the maximum value of *NetSavings* can be narrowed by eliminating all points to the right of point 3. The process is repeated by locating two points within the reduced range between point 1 and point 3. However point 4 is already a located a distance of g (UA_3 - UA_1) from the left edge. This, in fact, is how the value of g was determined. Therefore, it is not necessary to reevaluate *NetSavings* at this point. Point 5 is located a distance of g (UA_3 - UA_1) from the right edge and *NetSavings* is evaluated at this point. Since *NetSavings* is higher at point 4 then at point 5, all values to the left of point 5 are eliminated and the reduced range is between points 3 and 5. This process continues until the stopping criteria selected for the optimization are achieved.

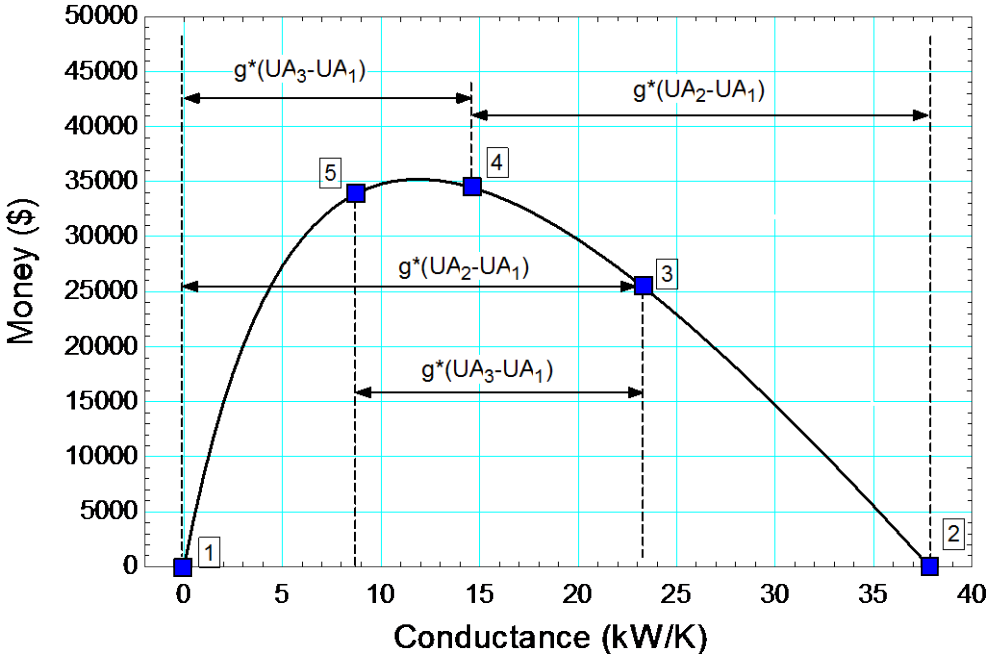


Figure 6-6: Progression of the golden section search.

Select the Golden Section search and then hit OK to initiate the optimization. The result is shown in Figure 6-7. The optimal value of *UA* has been identified to be 11.87 W/K resulting in a net savings of $35,217. The optimization required 30 iterations to find this result.

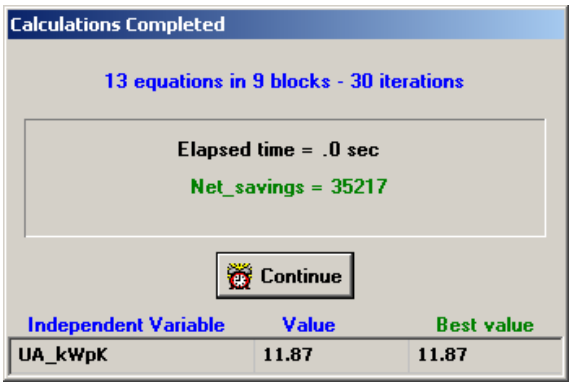


Figure 6-7: Calculations Completed dialog for golden section search.

The solution at the optimal value of *UA* is contained in the Solution Window, Figure 6-8.

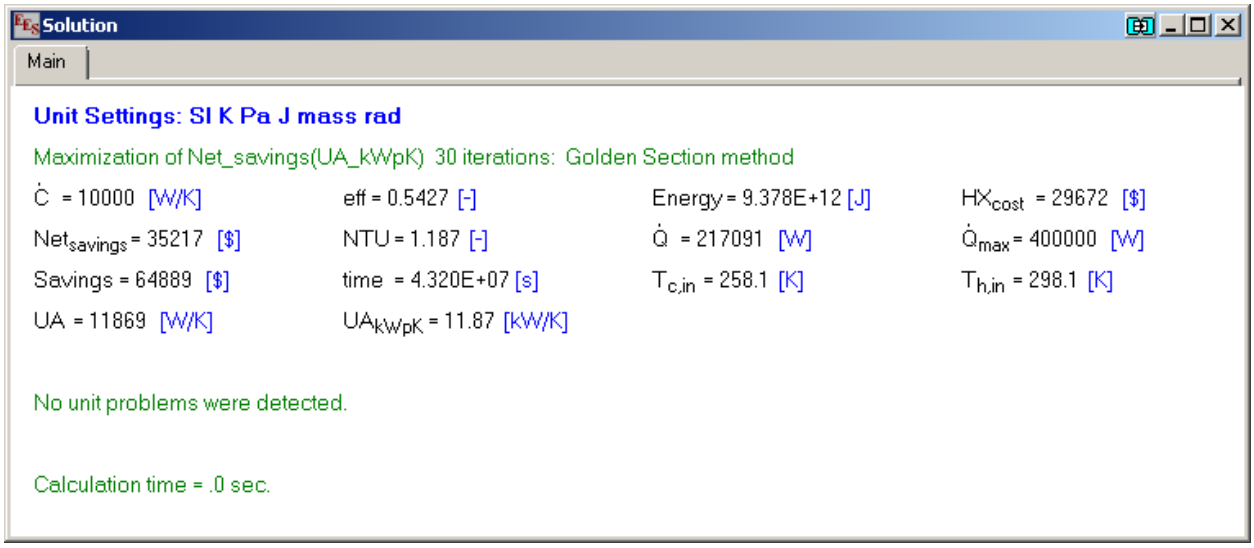


Figure 6-8: Solution Window.

The Quadratic Approximations Optimization Method

An alternative method for one-dimensional optimization is Quadratic Approximations. Quadratic Approximations proceeds three points at a time. For the first iteration, these three points correspond to the bounds of the problem (points 1 and 2 in Figure 6-9) and one point within the range (point 3).

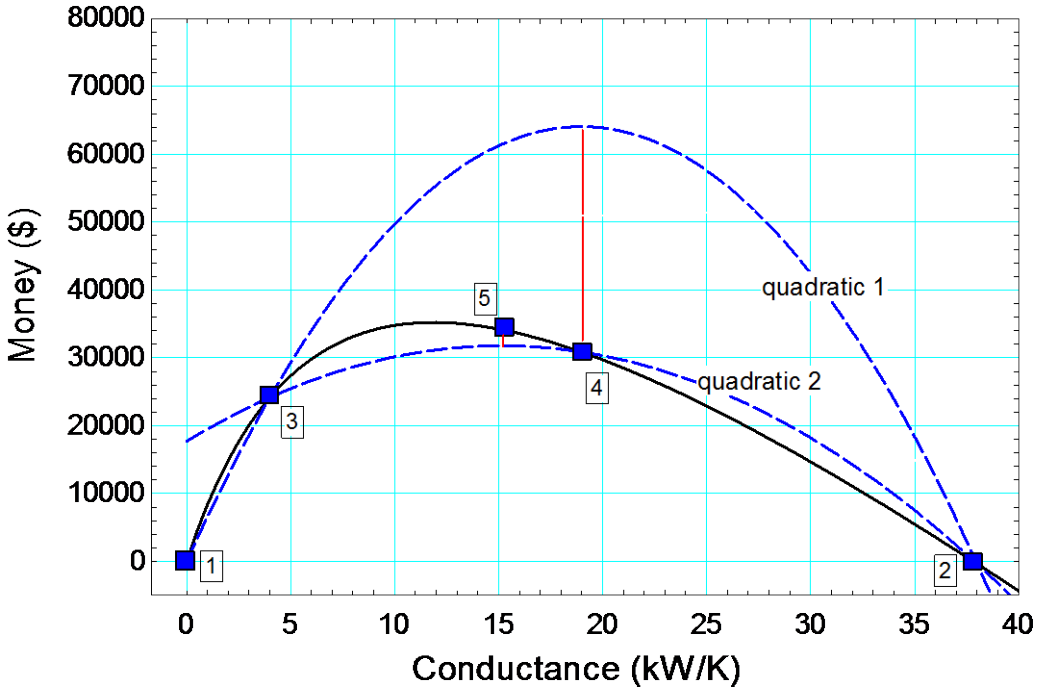


Figure 6-9: Progression of the quadratic approximations optimization method.

The objective function (*NetSavings*) is assumed to depend on the optimization variable (*UA*) in a quadratic manner:

$$NetSavings = aUA^2 + bUA + c \tag{6-6}$$

The coefficients *a*, *b*, and *c* in Eq. (6-6) are selected in order to fit the three points. The quadratic function that passes through points 1, 2, and 3 is shown as quadratic 1 in Figure 6-9. The optimal value of the optimization variable is predicted using the quadratic equation by setting the derivative to zero. Taking the derivative of Eq. (6-6) provides:

$$2aUA_{opt} + b = 0 \tag{6-7}$$

which can be solved for UA_{opt}:

$$UA_{opt} = -\frac{b}{2a} \tag{6-8}$$

The value of the objective function at UA_{opt} is computed, leading to point 4. The point with the smallest value of the objective function (for maximization) is discarded and the process is carried out again using the remaining three points (in this case points 1, 3, and 4). The new quadratic equation is labeled quadratic 2 in Figure 6-9 and leads to the identification of point 5. This process continues until the stopping criteria set for the optimization are achieved.

For most functions, the Quadratic Approximations method will converge to the optimal solution more quickly than the Golden Section method. If Quadratic Approximations is selected from the Find Minimum or Maximum dialog then the result is shown in Figure 6-10. Notice that the same result as the Golden Section search (Figure 6-7) was achieved, but the process required only 18 iterations.

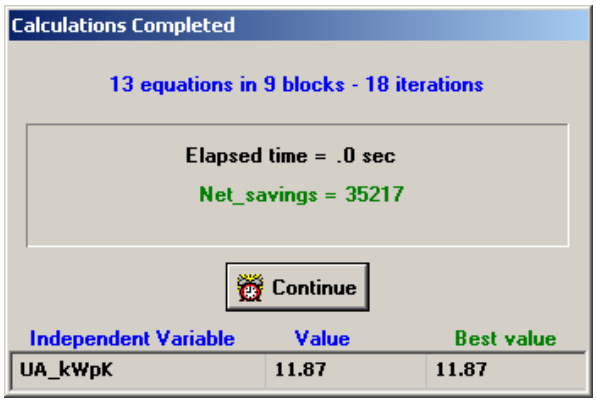


Figure 6-10: Calculations Completed dialog for quadratic approximations method.

Min/Max Table

It is sometimes useful to carry out optimizations within the context of a parametric study. For example, in this example we might want to determine the optimal conductance as a function of some other parameter (e.g., the cost of the heat exchanger or the capacitance rate of the fluids). This type of study can be accomplished using the Min/Max Table command. In order to utilize the Min/Max Table command it is necessary to create a Parametric Table that includes the dependent parameter that is being optimized (*NetSavings*), the independent parameter that is being adjusted (*UA*), and the parameter that is being parametrically varied, which in this case is the capacitance rate ($\dot{C}$). Any other parameter of interest can also be included in the table. Comment out the value of $\dot{C}$ that is specified in the Equations Window and vary the parameter from 5,000 W/K to 50,000 kW/K in the table, as shown in Figure 6-11(a).

1..10	1 $\dot{C}$ [W/K]	2 $Net_{savings}$ [$]	3 UA_{kWpK} [kW/K]
Run 1	5000		
Run 2	10000		
Run 3	15000		
Run 4	20000		
Run 5	25000		
Run 6	30000		
Run 7	35000		
Run 8	40000		
Run 9	45000		
Run 10	50000		

(a)

1..10	1 $\dot{C}$ [W/K]	2 $Net_{savings}$ [$]	3 UA_{kWpK} [kW/K]
Run 1	5000	17609	5.934
Run 2	10000	35217	11.87
Run 3	15000	52826	17.8
Run 4	20000	70435	23.74
Run 5	25000	88043	29.67
Run 6	30000	105652	35.61
Run 7	35000	123260	41.54
Run 8	40000	140869	47.47
Run 9	45000	158478	53.41
Run 10	50000	176086	59.34

(b)

Figure 6-11: Parametric Table (a) before and (b) after the Min/Max Table command is executed.

Select Min/Max Table from the Calculate menu in order to access the Minimize or Maximize Table dialog. Set the dependent and independent variables in the usual way, as shown in Figure 6-12.

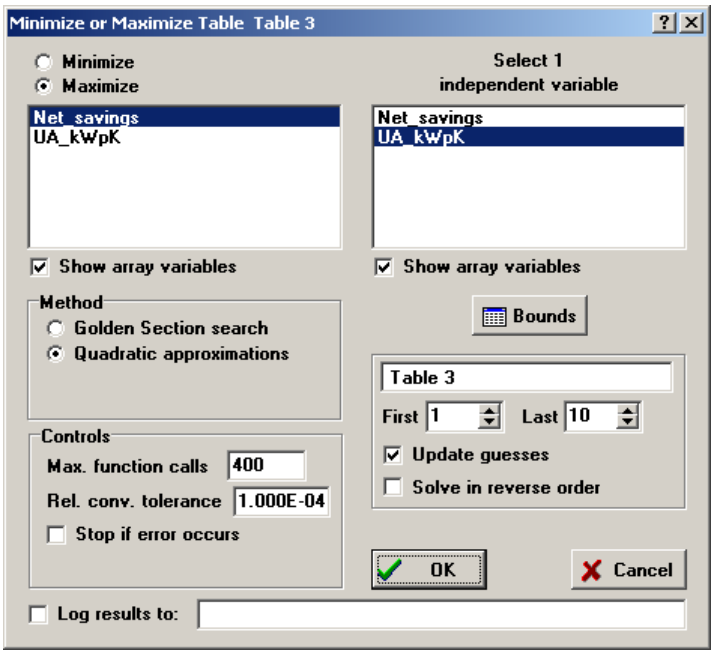


Figure 6-12: Minimize or Maximize Table dialog.

Select OK to initiate the calculations. Each row of the Parametric Table will be solved by carrying out the specified optimization with a different value of the variable $\dot{C}$. The result is shown in Figure 6-11(b). The value of *UA* indicated in each row is the optimal value that results in the maximum value for *NetSavings*, also listed in the row, for the value of $\dot{C}$ that is specified in that row.

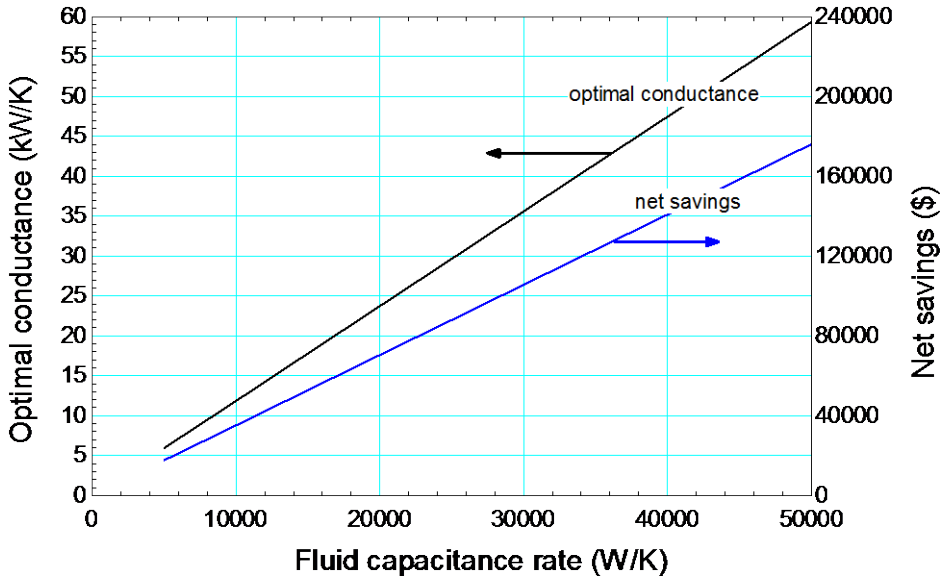


Figure 6-13: Optimal value of conductance and the associated net savings as a function of the capacitance rate of the fluids.

6.2 Multi-Dimensional Optimization

Typical design problems will involve multiple optimization variables and therefore require multi-dimensional optimization. In this section we will examine the multi-dimensional optimization algorithms that are available in EES in the context of the two-dimensional surface defined by the equation:

$$f = 0.7\exp\left(-\frac{r_1^2}{0.2^2}\right)+\left[1-0.7\exp\left(-\frac{r_1^2}{0.2^2}\right)\right]\exp\left(-\frac{r_2^2}{0.05^2}\right) \tag{6-9}$$

where

$$r_1 = \sqrt{(x-0.5)^2+(y-0.5)^2} \tag{6-10}$$

$$r_2 = \sqrt{(x-0.2)^2+(y-0.2)^2} \tag{6-11}$$

```
r1=sqrt((x-0.5)^2+(y-0.5)^2)
r2=sqrt((x-0.2)^2+(y-0.2)^2)
f=0.7*exp(-r1^2/0.2^2)+(1-0.7*exp(-r1^2/0.2^2))*exp(-r2^2/0.05^2)
```

Note that these multi-dimensional optimization algorithms discussed in this section can be easily extended to higher dimensions than two but are most easily visualized in two-dimensions. Contours of constant *f* in the parameter space of *x* and *y* are shown in Figure 6-14.

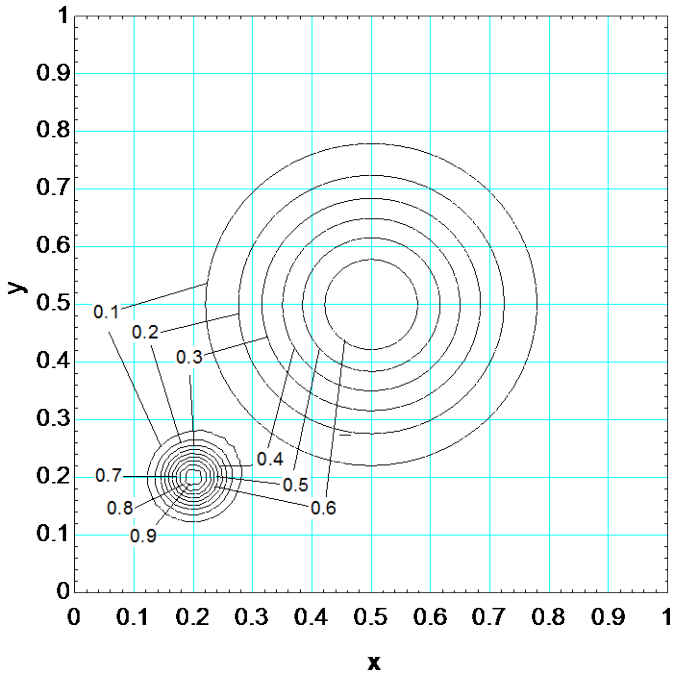


Figure 6-14: Contours of constant *f* in the parameter space of *x* and *y*.

Select Min/Max from the Calculate menu in order to access the Find Minimum or Maximum dialog, shown in Figure 6-15.

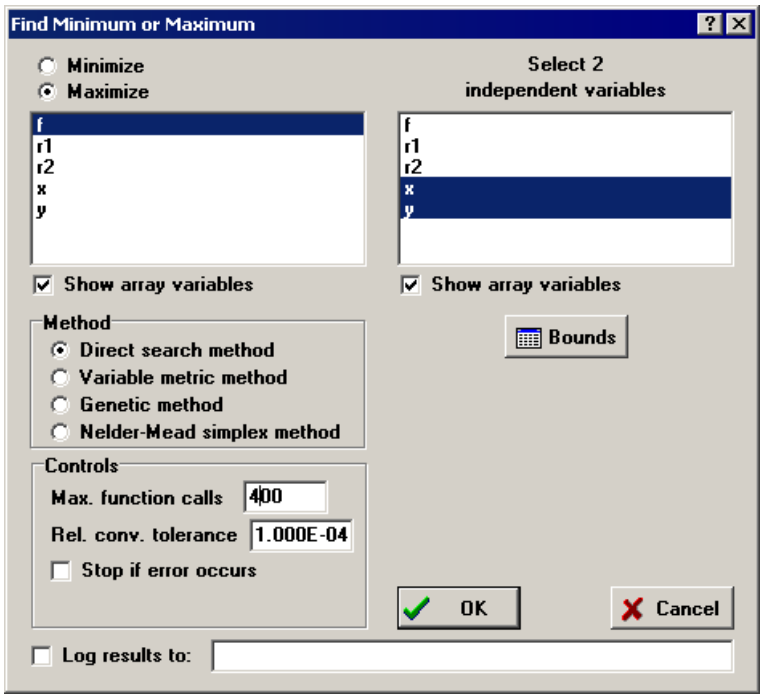


Figure 6-15: Find Minimum or Maximum dialog.

The objective function *f* should be maximized by adjusting the two independent variables (*x* and *y*). Set the bounds for both *x* and *y* to be from 0 to 1 and the guess values to be 0.7.

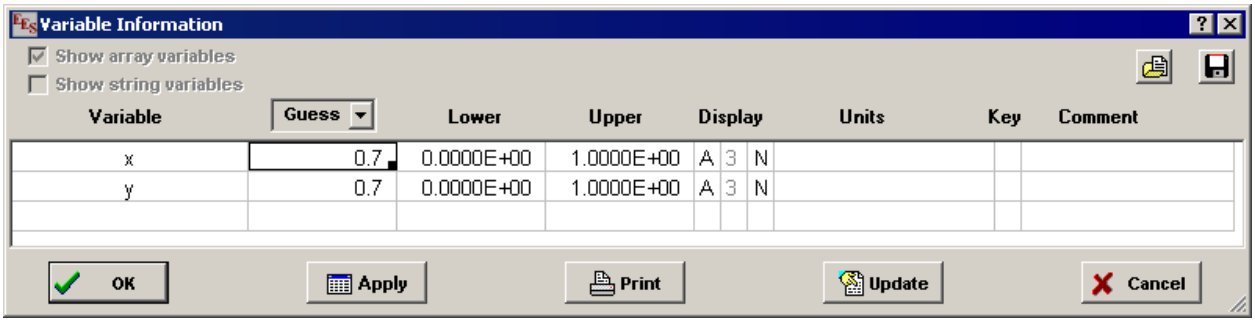


Figure 6-16: Bounds and guess values for *x* and *y*.

There are several methods available for multi-dimensional optimization.

The Direct Search Method

The Direct Search method uses a series of one-dimensional searches to locate the optimum. In its simplest form, EES will hold all but one of the optimization variables constant and then vary the single remaining parameter in order to locate the value at which the objective function is maximized along the one-dimensional path. This process can be accomplished using one of the one-dimensional optimization techniques discussed in Section 6.1. This process is repeated for each independent variable multiple times until the stopping criteria are achieved. One advantage of this technique is that it is not necessary to calculate the derivatives of the objective function and therefore the technique works well for surfaces that are not easily differentiable.

In the context of this two-dimensional problem, the initial one-dimensional optimization occurs along the line $x = 0.7$ as shown by path 1 in Figure 6-17. The objective function is maximized along this line at approximately $y = 0.5$. The second one-dimensional optimization occurs along the line $y = 0.5$ as shown by path 2. The objective function is maximized along this line at approximately $x = 0.5$. In this case, the optimum is located when these two one-dimensional optimizations are completed because the axes of the contours align with the axes of the plot.

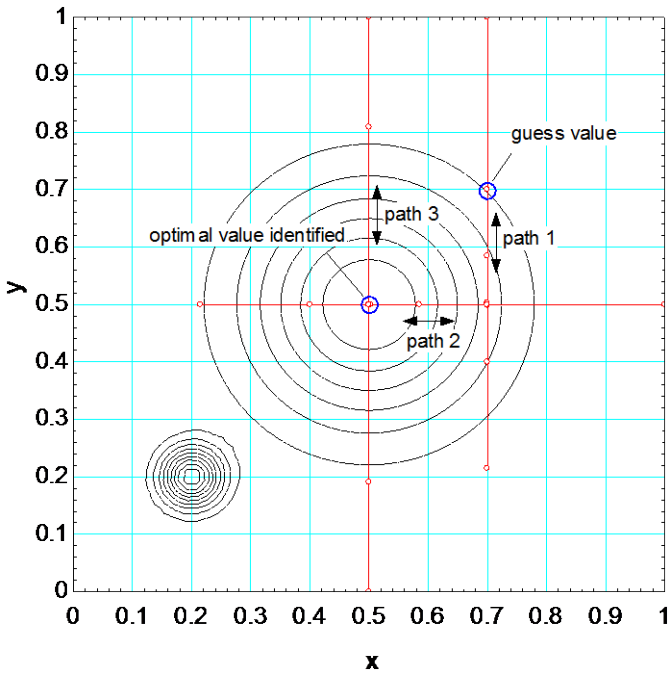


Figure 6-17: Progression of the direct search optimization method.

More commonly the axes of the contours and the plot are not aligned and many one-dimensional searches are required to locate the optimum point. EES employs a method referred to as conjugate directions in order to improve the efficiency of the optimization. The conjugate directions method makes the one-dimensional searches along directions that are oriented more favorably than those defined by the independent variable in the problem. More information about this technique can be found in Press et al. (1989).

Select Direct search method and then OK in order to initiate this method of optimization. The result is shown in Figure 6-18; it required 44 iterations (i.e., function evaluations) to arrive at the

optimal value $x = 0.5$ and $y = 0.5$ where the objective function is $f = 0.7$. Notice from Figure 6-14 that this is not the global maximum but rather a local maximum; the true optimal solution corresponds to the much smaller but taller peak at $x = 0.2$ and $y = 0.2$. The Direct Search method is sensitive to the guess values for the optimization variables (which correspond to the starting point of the optimization process). If a guess value that is closer to the peak is specified (e.g., $x = 0.15$ and $y = 0.15$) then the Direct Search method may converge to the global optimal value of $x = 0.2$ and $y = 0.2$ where the objective function is $f = 1$.

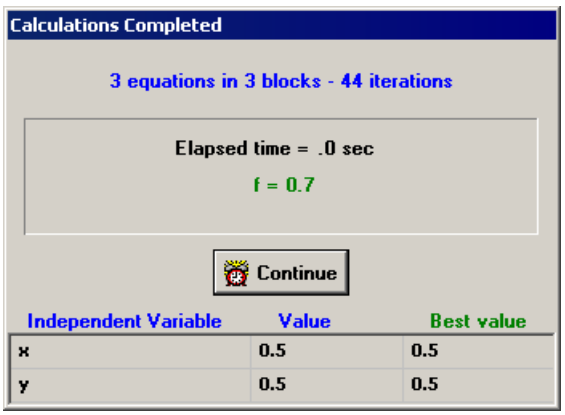


Figure 6-18: Result of the direct search optimization method.

The Variable Metric Method

The Variable Metric method is a multi-dimensional version of the quadratic approximations method that was discussed for one-dimensional optimization in Section 6.1. The objective function is fit to a quadratic function of all of the independent variables. The function is used to locate an estimate of the optimal value, which leads to a new trial point and the process is continued. The Variable Metric method implemented in EES is derived from Press et al. (1989). The Variable Metric method requires numerical derivatives of the objective function, which may be a disadvantage for some applications. The Variable Metric method, like the Direct Search method, is sensitive to the guess values that are used to start the optimization and may not find a global optimum.

The Genetic Method

The Professional version of EES includes the genetic optimization algorithm. The genetic optimization algorithm is designed to reliably locate global optimal values even if the surface has many local peaks. The genetic method implemented in EES is derived from the public domain Pikaia optimization program (version 1.2, April 2002) written by Paul Charbonneau and Barry Knapp at National Center for Atmospheric Research (NCAR). A detailed explanation of genetic optimization algorithms in general and specific details of the Pikaia program are provided on the Pikaia website (http://www.hao.ucar.edu/modeling/pikaia/pikaia.php).

The genetic method intends to mimic the processes occurring in biological evolution. A population of individuals (i.e., sample points) is initially chosen at random from the range specified by the bounds of the independent variables. The individuals in this population are surveyed to determine their fitness (i.e., the values of the objective function). Then a new

generation of individuals is generated in a stochastic manner by 'breeding' selected members of the current population according to their fitness. The characteristics of an individual that are passed on to the next generation are represented by encoded values of its independent variables. The probability that an individual in the current population will be selected for breeding the next generation is an increasing function of its fitness. The 'breeding' combines the characteristics of two parents in a stochastic manner. Additional random variations are introduced by the possibility of 'mutations' that cause the offspring to have characteristics that differ markedly from those of the parents. In the current implementation, the number of individuals in the population remains constant for each generation. The number of individuals, number of generations, and mutation rate can be adjusted once the Genetic method is selected, as shown in Figure 6-19.

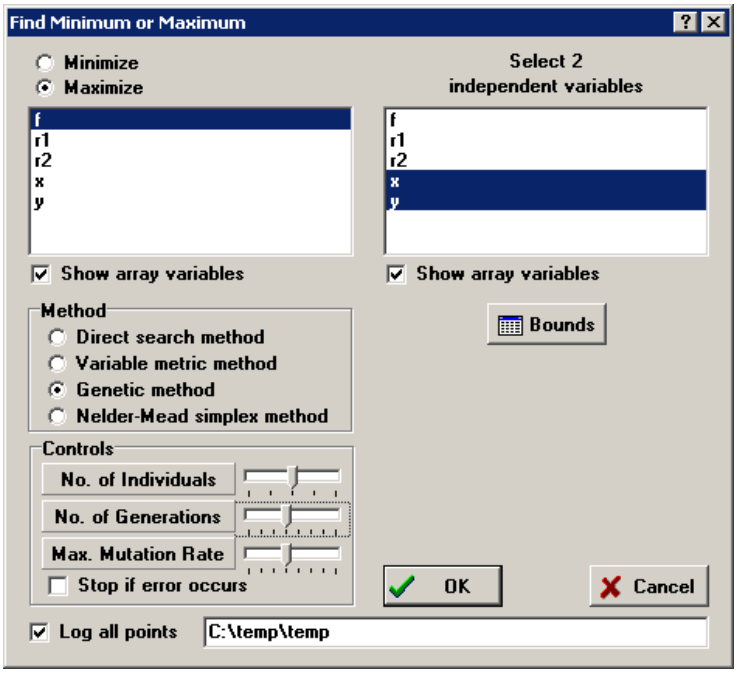


Figure 6-19: Find Minimum or Maximum dialog with genetic method selected.

Figure 6-20 illustrates the progression of the genetic optimization method; the populations for a few selected generations are shown. In the first generation, Figure 6-20(a), the members are uniformly distributed throughout the parameter space (although one member is located exactly at the coordinates associated with the guess value, $x = 0.7$ and $y = 0.7$). The population slowly converges towards the broad peak, as shown in Figure 6-20(b) for generation 5. Mutations cause individual members in later generations to move away from this broad peak, as shown in Figure 6-20(c) for generation 23. If even one member happens to come near the second, smaller but steeper peak then the entire population will eventually be attracted there. This is shown in Figure 6-20(d) for generation 77.

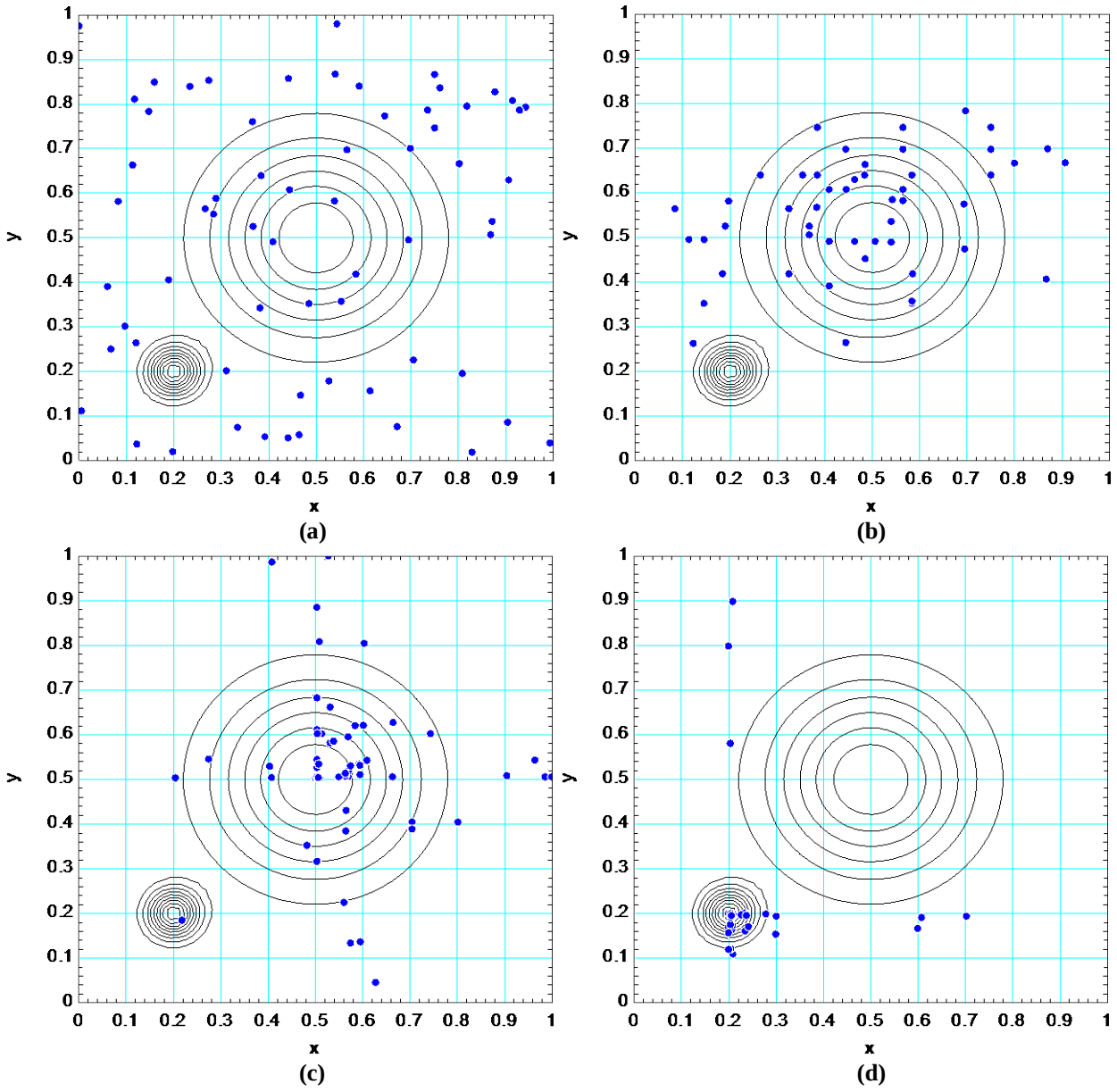


Figure 6-20: Progression of the genetic optimization method: (a) generation 1, (b) generation 5, (c) generation 23, and (d) generation 77.

The primary advantage of genetic optimization is its ability to reliably find a global optimum even if there are many local optimal values in the problem. The other techniques discussed in the context of this example tended to locate the small broad peak and bypass the large sharp peak unless the guess values happened to be set very near that location. Genetic optimization will inevitably find the sharp peak given a sufficiently large population and number of generations. The disadvantage of genetic optimization is that it takes a long time since many of the members of each population are not useful. Figure 6-21 shows the dialog that results when the genetic optimization is complete. Notice that more than 8,000 iterations were required, compared to 44 for the direct search method in Figure 6-18.

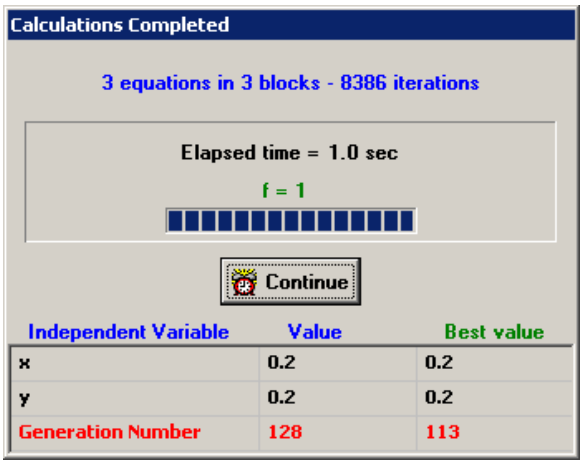


Figure 6-21: Result of the genetic optimization.

The Nelder-Mead Simplex Method

The Professional version of EES also includes the Nelder-Mead simplex method of multi-dimensional optimization (Nelder and Mead, 1965). The algorithm uses N+1 test points at a time, where N is the dimension of the problem (for our two-dimensional optimization problem, the method would use three points). The test points define a simplex (for this 2-D problem, the simplex is a triangle). The method proceeds by throwing out the worst point on the simplex and choosing a new point by reflecting away from the worst point about the axis formed by the other points. The process continues until the stopping criterion is achieved. The method is described more completely by Press et al. (1989).

An advantage of the Nelder-Mead method is that derivatives of the objective function are not required. However, Press et al. indicate the method is not as successful as either the Direct Search or Variable Metric techniques described earlier in this section. There is some evidence that the Nelder-Mead optimization technique is more robust for constrained optimization problems, described in Section 6.3.

6.3 Constrained Optimization

The independent variables in an optimization must be constrained by specified upper and lower bounds. However, it is possible that other quantities within the simulation must also be constrained during the optimization process. Consider the objective function:

$$f = (1-x)^2 + 2\left(y - x^2\right)^2 \tag{6-12}$$

```
f=(1-x)^2+2*(y-x^2)^2                    "objective function"
```

which is shown in Figure 6-22.

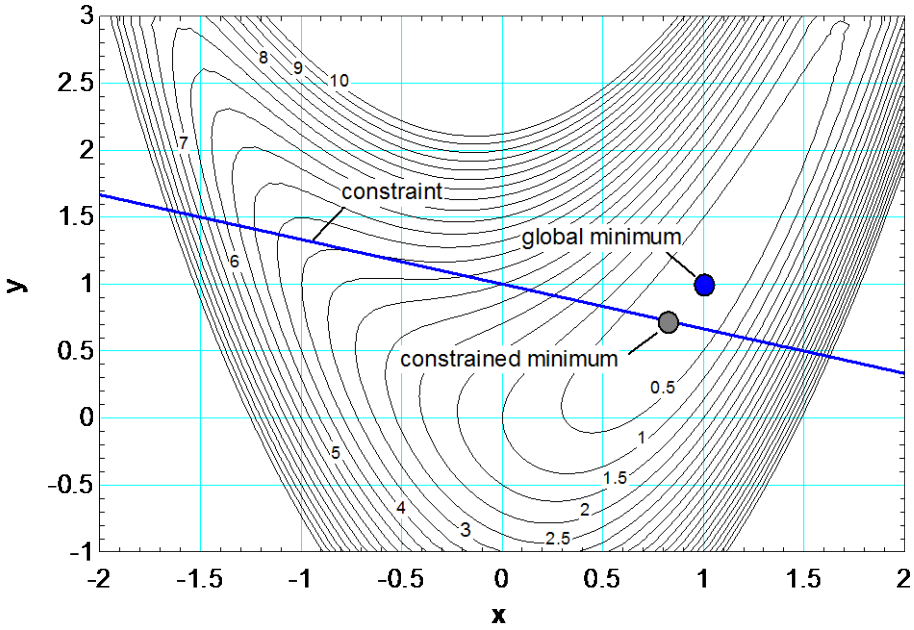


Figure 6-22: Contours of constant *f* in the parameter space of *x* and *y*.

Minimization of the objective function can be accomplished with any of the four methods discussed in Section 6.2 and should result in $f = 0$ at $x = 1$ and $y = 1$, as shown in Figure 6-22. We can constrain the optimization in various ways. For example:

$$x + 3y < 3 \tag{6-13}$$

which requires that all viable points be below the line labeled constraint in Figure 6-22.

Implementing Constraints using Bounds

The most straightforward method for accomplishing the minimization of f subject to the constraint given by Eq. (6-13) is to define a new variable, z:

$$z = x + 3y \tag{6-14}$$

```
z=x+3*y                                    "constraint variable"
```

and set an upper bound on z that is equal to 3 in the Variable Information window, as shown in Figure 6-23. To obtain the same results as shown in these examples, set the Max. function calls to 800 and the Rel. conv. tolerance to 1E-10 in the Controls section of the dialog.

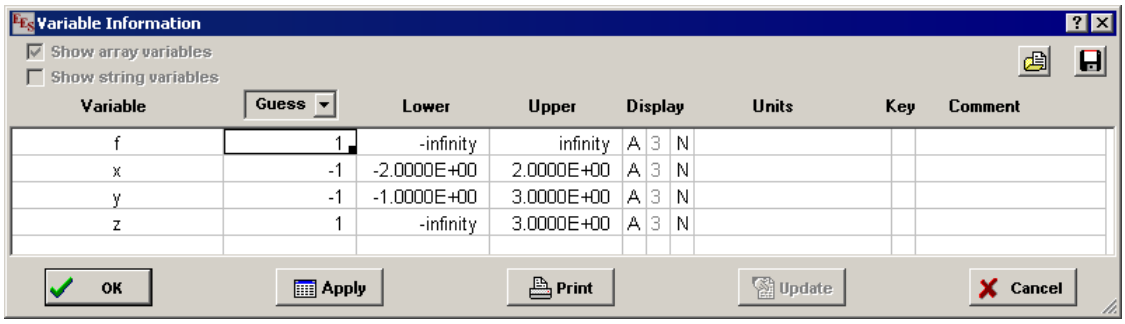


Figure 6-23: Variable Information window with upper bound set for *z*.

Choose one of the optimization methods, for example the Variable metric method, and the result should be as shown in Figure 6-24. The point identified as the constrained minimum is shown in Figure 6-22.

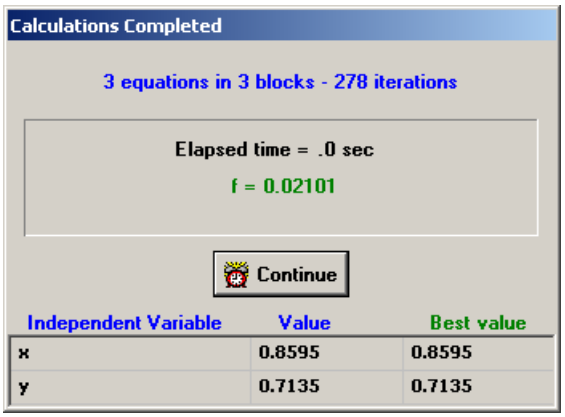


Figure 6-24: Result of using the direct search method to carry out the constrained optimization by setting an upper bound on the constraint variable z.

Parameterizing Variables

It is often the case that the constraints narrow the allowable range of the independent variables. This type of situation is encountered when you are optimizing concentrations (the allowable range for each concentration is limited by those already selected), extraction pressures, intermediate refrigeration temperatures, etc. In this situation it is often best to define a new variable that parameterizes the allowable range.

In this example, once the value of the independent variable y is selected, the allowable range for x is defined by:

$$-2 < x < 3 - 3y \tag{6-15}$$

We can define a new variable, ε, that can only take on values between 0 and 1 as it encompasses the entire allowable range of x. Therefore, x should be assigned based on:

$$x = -2 + \varepsilon\left[3 - 3y - (-2)\right] \tag{6-16}$$

```
x= -2+epsilon*(3-3*y-(-2))          "assign x using the variable epsilon"
```

Now select y and ε to be the independent variables in the optimization, as shown in Figure 6-25, and set the bounds on ε to be from 0 to 1, as shown in Figure 6-26.

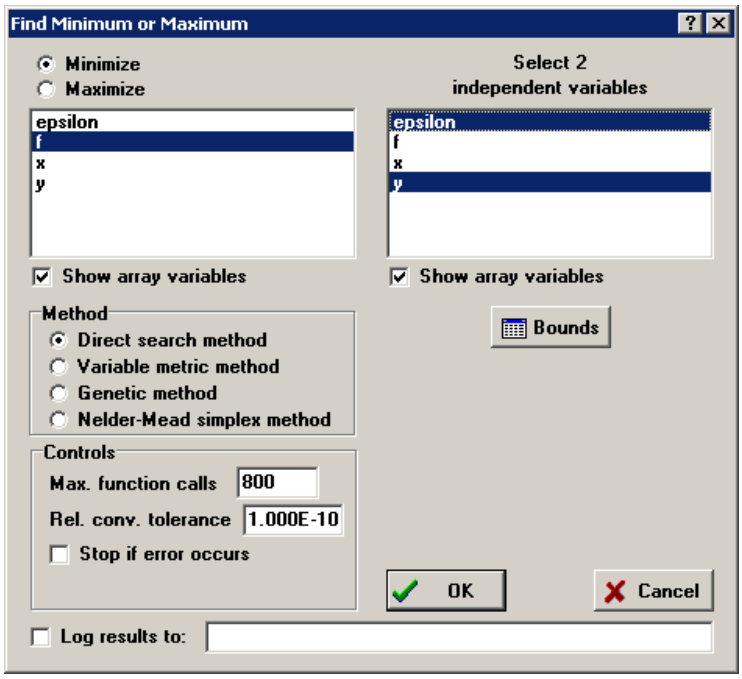


Figure 6-25: Find Minimum or Maximum dialog using the ε.

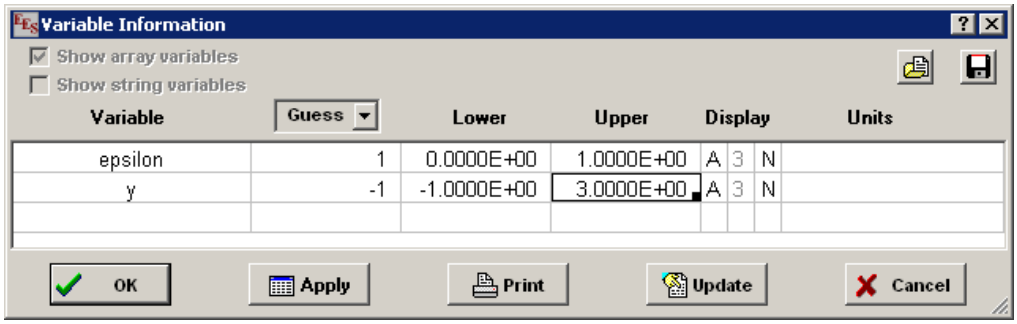


Figure 6-26: Bounds on y and ε.

Carrying out the optimization leads to the result shown in Figure 6-27. Note that only 79 iterations were required using the parameterization method (Figure 6-27) compared with 278 iterations using the technique of bounding a constraint variable (Figure 6-24). This reduction in computational time occurs because every value of ε and y that is selected satisfies the constraint and therefore no time is wasted on independent variables that are subsequently discarded.

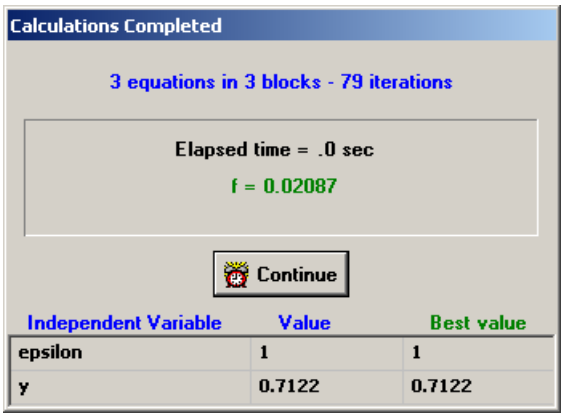


Figure 6-27: Result of using the direct search method to carry out the constrained optimization using a parameter ε.

Penalty Function

The use of a parameterizing variable is clearly the best option for this problem. However, there are problems where it is not possible to use this technique. In this case, it may be useful to enforce the constraint by applying a penalty function that is large and proportional to the degree to which the constraint is violated. The penalty function allows the optimization algorithm to navigate into and back out of the region that is forbidden by the constraint. In this problem, we might enforce a penalty function only if the constraint, Eq. (6-13), is violated and therefore:

$$x + 3y > 3 \tag{6-17}$$

The size of the penalty function should be (1) large relative to the size of the function being optimized; (2) proportional to the degree to which the constraint is not satisfied (i.e., x + 3*y* – 3), and (3) zero if the constraint is satisfied. One possibility for the penalty function is:

$$pf = \begin{cases} 0 & \text{if } x+3y<3 \\ 20+10\left(x+3y-3\right) & \text{if } x+3y>3 \end{cases} \tag{6-18}$$

The penalty function can be implemented directly in the Equations Window using the If function in EES. The If function has the following format:

```
If(A,B,C,D,E)
```

and returns C if A < B, D if A = B, and E if A > B. Therefore, the penalty function can be implemented according to:

```
pf=If(x+3*y,3,0,0,1+(x+3*y-3))                "penalty function"
```

The penalty function is added to the original objective function to achieve:

$$f = \underbrace{(1-x)^2 + 2\left(y - x^2\right)^2}_{\text{original objective function}} + pf \tag{6-19}$$

```
f=(1-x)^2+2*(y-x^2)^2+pf          "objective function modified by penalty function"
```

The addition of the penalty function has essentially transformed the objective function from the smooth surface shown in Figure 6-22 to a surface with a "cliff" along the constraint line and a sloping hill above it, as shown in Figure 6-28. (The surface in Figure 6-28 was drawn using the 3-D surface plotting option described in Chapter 9.)

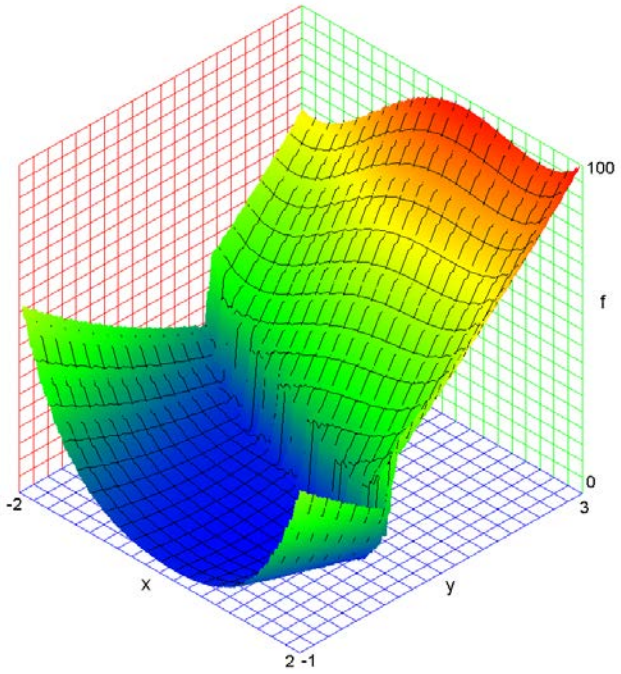


Figure 6-28: Surface modified by the penalty function.

Running the optimization with the penalty function using the Variable Metric technique leads to the result shown in Figure 6-29.

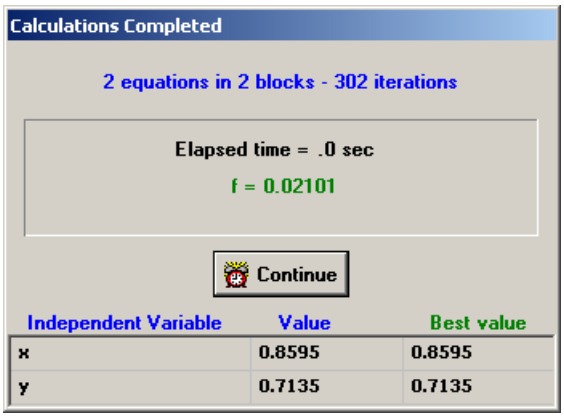


Figure 6-29: Result of using the Variable Metric technique to carry out the constrained optimization using a penalty function.

Notice that 302 iterations are required, which is not as good as parameterization, but about the same as using a bound on a constraint variable in this case.

6.4 Other uses for Optimization

The utility of the optimization algorithms for design problems, in which some performance metric is to be maximized by adjusting one or more input parameters, is clear. However, optimization is also useful for other problems, as discussed in this section.

Solving Difficult Sets of Equations

EES is quite good at solving systems of nonlinear equations. However, its ability to find a solution sometimes depends on the quality of the guess values entered for each of the variables, as discussed in Chapter 5. In some problems, a good set of guess values will not be obvious and therefore it may be best to formulate the problem as an optimization problem.

For example, consider the system of two equations shown below:

$$x_1 \sin\left(x_1\right) - x_2 = 0 \tag{6-20}$$

$$\frac{1}{x_1} + 2\,x_1 - x_2 - 5 = 0 \tag{6-21}$$

These two equations can be programmed in EES:

```
x1*sin(x1)-x2=0
1/x1+2*x1-x2-5=0
```

but will not solve if the default guess values and bounds are used, leading to the message shown in Figure 6-30.

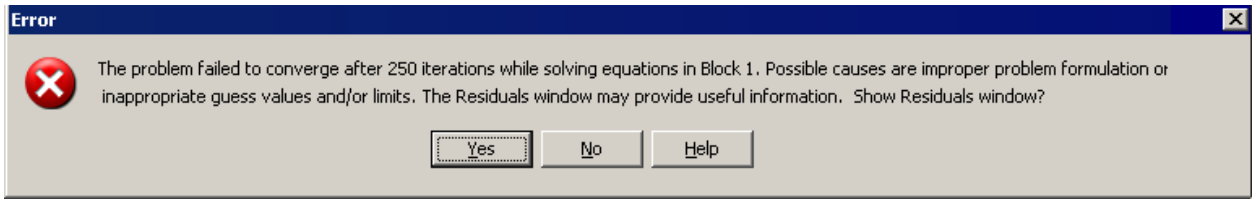


Figure 6-30: Error message.

The problem can be cast as an optimization problem by setting a value for x_1:

```
x1=1                                    "set value for x1"
```

and using Eq. (6-20) to determine x_2:

```
x1*sin(x1)-x2=0                    "calculate x2"
```

The degree to which Eq. (6-21) is not solved is computed according to:

```
err=abs(1/x1+2*x1-x2-5)            "error in solution"
```

Solving leads to *err* = 2.841. Note that the value of the variable err will be zero if both equations are solved. The use of the absolute value (abs) function requires that the value of *err* will be always positive. Therefore, the problem has been recast as a minimization problem; minimize the value of the variable err by adjusting the variable x1 in order to find the solution to the equations.

Comment out the value of x_1:

```
{x1=1}                             "set value for x1"
```

and select Min/Max from the Calculate menu to access the Find Minimum or Maximum dialog shown in Figure 6-31. Place reasonable bounds on the variable x1 and select OK. The result is x_1 = 2.795, x_2 = 0.9483, and *err* = 0.000397. The solution can be obtained more precisely by changing the stopping criteria on the optimization.

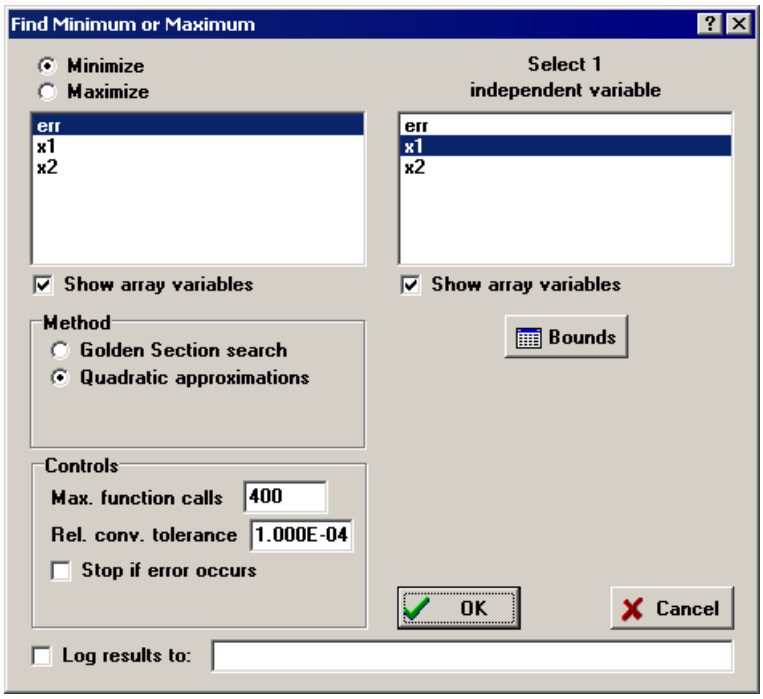


Figure 6-31: Find Minimum or Maximum dialog.

Curve Fitting

Chapter 2 discussed the options that are available within EES for curve fitting. However, curve fitting can also be cast as an optimization problem in which some measure of the disagreement between the predicted values and the data is minimized by adjusting the values of the

coefficients of the equation used for the curve fit. For example, Table 6-1 lists pressure, temperature, and specific volume data for carbon dioxide. We could use these data to find the best-fit coefficients (*a* and *b*) for the van der Waals equation of state:

$$P = \frac{RT}{(v-b)} - \frac{a}{v^2} \tag{6-22}$$

where *R* is the ideal gas constant.

Table 6-1: Pressure, temperature, and specific volume for carbon dioxide.

Pressure (MPa)	Temperature (°C)	Specific volume (m^3/kg)
8	30	0.001423
9	30	0.001343
10	30	0.001296
11	30	0.001262
8	40	0.003601
9	40	0.00206
10	40	0.001587
11	40	0.001461
8	50	0.004562
9	50	0.003509
10	50	0.002602
11	50	0.001989

The data are entered into a Lookup Table in EES, as shown in Figure 6-32.

Lookup Table

Lookup 1

	1 Pressure [MPa]	2 Temperature [C]	3 Specific volume [m3/kg]
Row 1	8	30	0.001423
Row 2	9	30	0.001343
Row 3	10	30	0.001296
Row 4	11	30	0.001262
Row 5	8	40	0.003601
Row 6	9	40	0.00206
Row 7	10	40	0.001587
Row 8	11	40	0.001461
Row 9	8	50	0.004562
Row 10	9	50	0.003509
Row 11	10	50	0.002602
Row 12	11	50	0.001989
Row 13	8	60	0.005218
Row 14	9	60	0.004248
Row 15	10	60	0.003449
Row 16	11	60	0.002795

Figure 6-32: Carbon dioxide data entered in a Lookup Table.

Initial values for a and b are guessed in order to set up the problem and the value of R is computed.

```
$UnitSystem SI Mass J K Pa Rad
$TabStops 0.25 0.5 0.75 3 in

a=100 [N-m^4/kg^2]                          "coefficients for Equation of State"
b=0.01 [m^3/kg]
R=R#/MolarMass('CO2')                       "gas constant"
```

Setup a Duplicate loop in order to cycle through each of the 12 rows of Lookup Table. Use the LookUp command, discussed in Section 1.8, to obtain the pressure, temperature, and specific volume at each data point.

```
duplicate i=1,12
    P[i]=LookUp('CO2Data',i,1)*convert(MPa,Pa)          "pressure from table, converted to Pa"
    T[i]=ConvertTemp(C,K,LookUp('CO2Data',i,2))        "temperature from table, converted to K"
    v[i]=LookUp('CO2Data',i,3)                         "specific volume from table"
```

Use Eq. (6-22) to compute the pressure based on the curve fit, P_{fit}, at the specific volume and temperature associated with the data point.

```
    P_fit[i]=R*T[i]/(v[i]-b)-a/v[i]^2                  "pressure according to Equation of State"
```

Calculate the square of the error between the actual pressure and the pressure calculated from the curve fit.

```
    err[i]=(P[i]-P_fit[i])^2                           "error"
end
```

The total error is defined as the square root of the sum of the error associated each data point, divided by the number of data points.

```
err=sqrt(sum(err[i],i=1,12))/12                        "total error"
```

Solving provides err = 1.566x10^7 Pa for a = 100 N-m^4/kg^2 and b = 0.01 m^3/kg. The best-fit coefficients can be found by commenting out the specified values of a and b.

```
{a=100 [N-m^4/kg^2]                                    "coefficients for Equation of State"
b=0.01 [m^3/kg]}
```

Min/Max is selected from the Calculate menu in order to bring up the Find Minimum or Maximum dialog, shown in Figure 6-33. Minimize the value of err by varying a and b. Set some reasonable bounds and guesses for these independent variables and select OK to initiate the optimization,

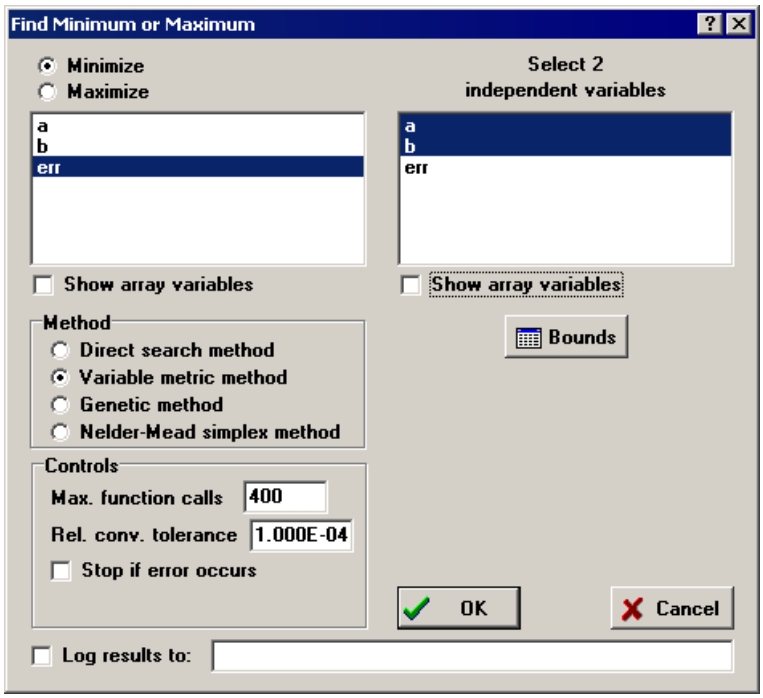


Figure 6-33: Find Minimum or Maximum dialog.

The results of the optimizations are shown in Figure 6-34. Note that the values of *a* and *b* identified by the optimization are close to those calculated theoretically based on the critical parameters of carbon dioxide, $a = 138.2$ N-m^4/kg^2 and $b = 0.0007129$ m^3/kg.

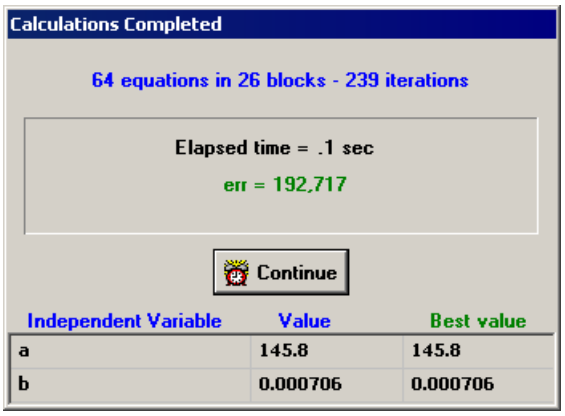


Figure 6-34: Result of optimization.

References

Klein, S.A. and G.F. Nellis, *Thermodynamics*, Cambridge University Press, (2011).

Nelder, J.A. and Mead, R., *The Computer Journal*, Vol. 7, p. 308, (1965).

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in Pascal*, Cambridge University Press, (1989).

7 INTEGRATION

The result of an engineering analysis is often one or more ordinary differential equations that are sufficiently complex as to require numerical, rather than analytical solution. Partial differential equations can also be solved by approximating them as a set of ordinary differential equations. There are a variety of numerical techniques that can be used to solve ordinary differential equations; these techniques range from explicit to implicit and from low order to high order. In the first section of this chapter, we will briefly review these techniques and discuss their characteristics. EES has a built in, powerful, numerical integration algorithm that can be used to solve one or more ordinary differential equations. The remainder of this chapter discusses the proper use of this feature of EES.

7.1 Numerical Integration of Ordinary Differential Equation

The mathematical description of many interesting problems in thermodynamics and other areas of engineering involves ordinary differential equations (ODEs). In some cases, the ODEs are sufficiently simple that an analytical solution can be derived. However, in most cases this is not possible and therefore numerical solutions are required.

The numerical solution to any transient problem begins with the derivation of the governing differential equation, which allows the calculation of the rate of change of the dependent parameter to be computed as a function of the value of the dependent parameter and time. The solution to the problem is therefore a matter of integrating the governing differential equation forward in time. There are a number of techniques available for numerical integration, each with its own characteristic level of accuracy, stability, and complexity. These numerical integration techniques are introduced and discussed in this section in order to understand the Integral command in EES, which provides an automated method for numerical integration. The EES Integral command automatically implements a sophisticated, adaptive step-size integration technique that is described in Section 7.2.

The governing differential equation that provides the rate of change of a variable (or several variables) given its own value (or values) is sometimes referred to as the state equation(s) in the context of a dynamic system. State equations characterize the transient behavior of many engineering problems in dynamics, kinematics, fluids, electrical circuits, heat transfer and thermodynamics.

Example ODE

In this section, the ODE:

$$\frac{dy}{dt} + 4t\,y = -2t \tag{7-1}$$

with initial condition $y = 0$ at $t = 0$ is solved numerically. This ODE is relatively simple and therefore an analytical solution can be determined and used as the basis for evaluating the accuracy of the numerical solutions that are discussed in this section.

Analytical Solution

Equation (7-1) can be separated:

$$\frac{dy}{(2+4y)} = -t\,dt \tag{7-2}$$

and integrated:

$$\int_0^y \frac{dy}{(2+4y)} = -\int_0^t t\,dt \tag{7-3}$$

The variable w is defined as:

$$w = 2 + 4y \tag{7-4}$$

The differential of w is:

$$dw = 4\,dy \tag{7-5}$$

Substituting Eqs. (7-4) and (7-5) into Eq. (7-3) provides:

$$\int_2^{2+4y} \frac{dw}{4w} = -\int_0^t t\,dt \tag{7-6}$$

Carrying out the integration in Eq. (7-6) provides:

$$\frac{1}{4}\ln\left(\frac{2+4y}{2}\right) = -\frac{t^2}{2} \tag{7-7}$$

Equation (7-7) is solved for y:

$$y = \frac{1}{2}\exp\left(-2t^2\right) - \frac{1}{2} \tag{7-8}$$

The analytical solution is entered in EES:

```
y = -1/2+exp(-2*t^2)/2                    "analytical solution"
```

and used to generate Figure 7-1.

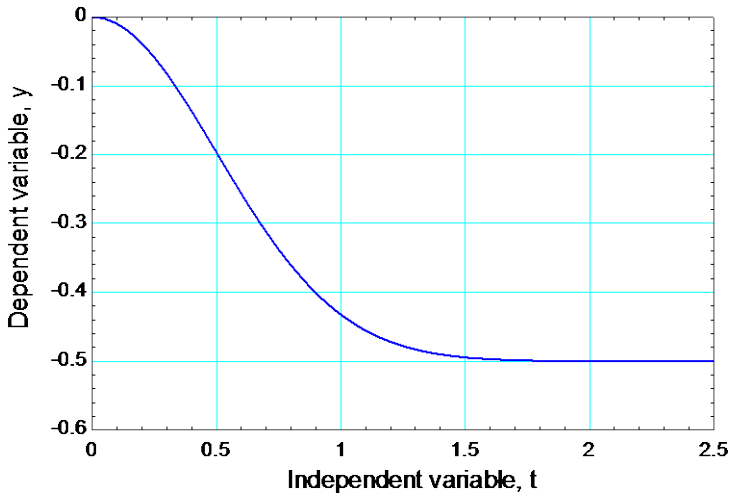


Figure 7-1: Analytical solution to Eq. (7-1).

Discretizing Time

The solution to this problem can also be obtained numerically using one of several available numerical integration techniques. Each numerical technique requires that the total simulation time (t_{sim}) be broken into small time steps. The simplest possibility uses equal-sized steps, each with duration Δt:

$$\Delta t = \frac{t_{sim}}{(M-1)} \tag{7-9}$$

where M is the number of times at which the solution will be evaluated. The solution at each time step (y_j) is computed by the numerical model, where j indicates the time step (y_1 is the initial condition at $t_1 = 0$). The time corresponding to each time step is therefore:

$$t_j = \frac{(j-1)}{(M-1)} t_{sim} \quad \text{for } j = 1..M \tag{7-10}$$

Euler's Technique (First Order Explicit)

The solution at the end of each time step is computed based on the value at the beginning of the time step and the governing differential equation using one of many possible numerical techniques. The simplest (and generally the least efficient) method for numerical integration is Euler's method. Euler's method approximates the rate of change within the time step as being constant and equal to its value at the beginning of the time step. Therefore, for any time step j:

$$y_{j+1} = y_j + \left.\frac{dy}{dt}\right|_{y=y_j, t=t_j} \Delta t \tag{7-11}$$

Because the value of the solution at the end of the time step (y_{j+1}) can be calculated explicitly using information at the beginning of the time step (y_j), Euler's method is referred to as an explicit numerical technique. The solution at the end of the first time step (y_2) is given by:

$$y_2 = y_1 + \left.\frac{dy}{dt}\right|_{y=y_1,t=0} \Delta t \tag{7-12}$$

where $\left.\frac{dy}{dt}\right|_{y=y_1,t=0}$ is computed using the governing differential equation. Equation (7-12) is written for every time step, resulting in a set of explicit equations that can be solved sequentially for y at each discrete time.

As an example, we can solve Eq. (7-1) with $y_1 = 0$ at $t_1 = 0$. The time step duration and the array of times for which the solution will be computed is specified using Eqs. (7-9) and (7-10):

```
t_sim=2.5                          "time to be simulated"
M=21                               "number of time steps"
Dt=t_sim/(M-1)                     "time step duration"
duplicate j=1,M
    t[j]=(j-1)*Dt                  "time at each time step"
end
```

The initial value of the variable y is specified as the first element in the array y.

```
y[1]=0                             "initial value"
```

It is often useful to carry out a single step in a numerical solution before simulating all of the steps. The value of the time rate of change of y at $y = y_1$ and $t = t_1$ is determined using Eq. (7-1) and substituted into Eq. (7-12):

```
"take the first step"
dydt[1]+4*t[1]*y[1]=-2*t[1]
y[2]=y[1]+dydt[1]*Dt
```

Once we have established that it is possible to take one step, it is easy to take all of the steps. Comment out the code used to take a single step:

```
{"take the first step"
dydt[1]+4*t[1]*y[1]=-2*t[1]
y[2]=y[1]+dydt[1]*Dt}
```

and setup a duplicate loop that goes from $j = 1$ to $j = (M - 1)$. Copy the code to take one step and place it inside the duplicate loop. Change the element index 1 to j and the element index 2 to $j+1$.

```
"take all of the steps"
duplicate j=1,(M-1)
```

```
    dydt[j]+4*t[j]*y[j]=-2*t[j]
    y[j+1]=y[j]+dydt[j]*Dt
end
```

Solving the EES code results in a solution for y at every value of t in the Arrays Table. The numerical solution is overlaid onto the analytical solution in Figure 7-2. Clearly the numerical solution is approximate. However, the approximation improves as the number of time steps used in the simulation increases (i.e., the duration of the time step is reduced). Figure 7-2 shows that numerical simulation more closely approaches the analytical solution if the time step duration is reduced from $\Delta t = 0.125$ s to $\Delta t = 0.05$ s.

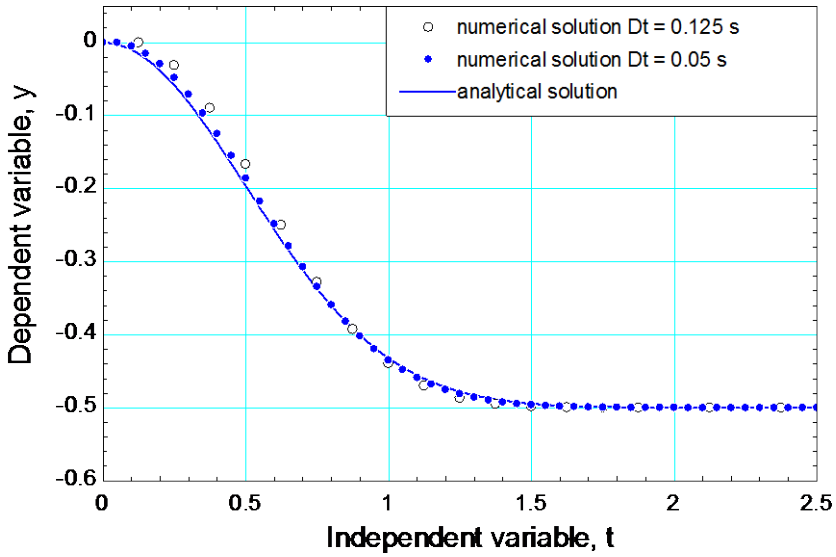


Figure 7-2: Euler solution with $\Delta t = 0.125$ s and $\Delta t = 0.05$ s overlaid onto the analytical solution.

The analytical solution allows us to compute the error associated with the numerical solution according to the discrepancy between the numerical and analytical solution at each value of time. The maximum error at any time step, sometimes referred to as the global error, is computed using the Max function.

```
duplicate j=1,M
    y_an[j] = -1/2+exp(-2*t[j]^2)/2          "analytical solution"
    err[j]=abs(y_an[j]-y[j])                 "error"
end
error=Max(err[1..M])                         "maximum or global error"
```

Figure 7-3 illustrates the global error associated with Euler's technique as a function of the time step duration.

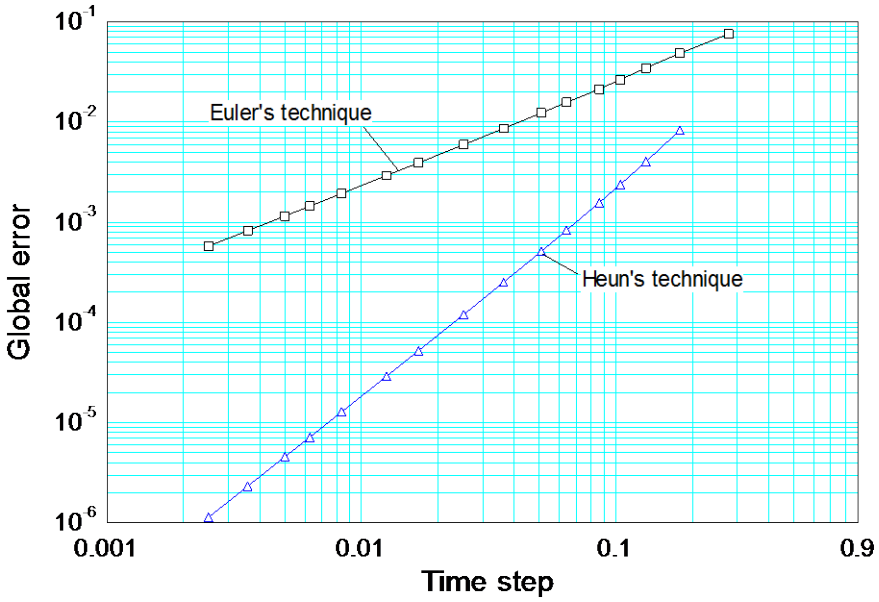


Figure 7-3: Global error as a function of the time step duration for Euler's technique and Heun's technique.

Figure 7-3 shows that the global error is proportional to the duration of the time step to the first power and therefore Euler's technique is referred to as a first order technique. This characteristic of the solution can be inferred by examination of Eq. (7-12), which is essentially the first two terms of a Taylor series expansion of the function y about time $t = 0$:

$$\underbrace{y_2 = y_1 + \left.\frac{dy}{dt}\right|_{y=y_1,t=0} \Delta t}_{\text{Euler's approximation}} + \underbrace{\left.\frac{d^2y}{dt^2}\right|_{y=y_1,t=0} \frac{\Delta t^2}{2!} + \left.\frac{d^3y}{dt^3}\right|_{y=y_1,t=0} \frac{\Delta t^3}{3!} + \ldots}_{\text{local err} \approx \text{ neglected terms}} \tag{7-13}$$

Examination of Eq. (7-13) shows that the local error associated with each time step corresponds to the neglected terms in the Taylor's series and is therefore approximately proportional to Δt^2 (neglecting the smaller, higher order terms). The local error accumulates during a simulation and becomes the global error, resulting in an error that is proportional to the product of the number of time steps and the local error:

$$\text{global error} \propto M\, \Delta t^2 \tag{7-14}$$

The number of time steps is given by:

$$M = \frac{t_{sim}}{\Delta t} \tag{7-15}$$

Substituting Eq. (7-15) into Eq. (7-14) provides:

$$\text{global error} \propto \Delta t \tag{7-16}$$

which agrees with the error characteristic observed in Figure 7-3. Most numerical techniques that are commonly used have higher order and therefore achieve higher accuracy.

Another drawback of Euler's technique (and of any explicit numerical technique) is that it may become unstable if the duration of the time step is too large. For our example, if the time step duration is increased above approximately 0.6 s, then the numerical solution becomes unstable, as shown in Figure 7-4. The time step at which the solution becomes unstable is called the critical time step.

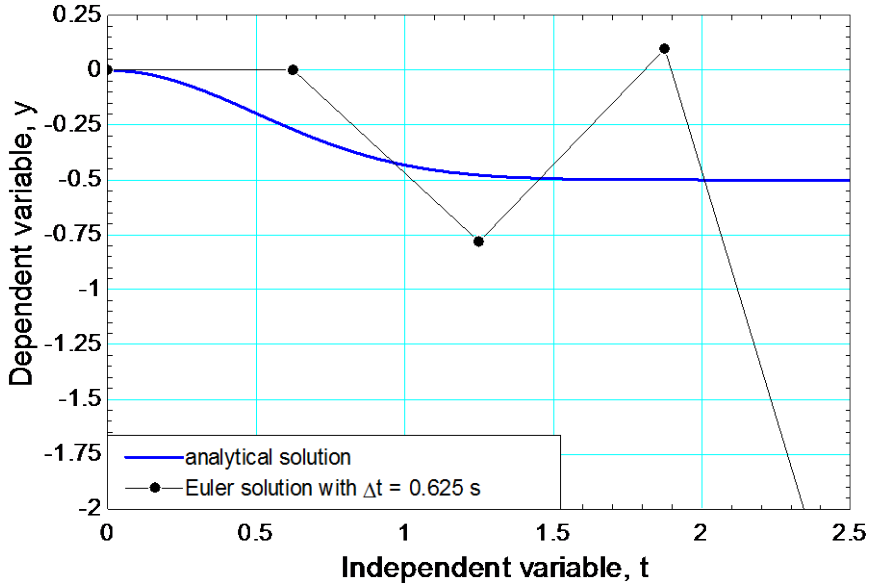


Figure 7-4: Analytical and numerical solution with Δt = 0.625 s.

Heun's Method (Second Order Explicit)

Euler's method is the simplest example of a numerical integration technique; it is an explicit technique with first order accuracy. In this section, Heun's method is presented. Heun's method is an explicit technique with second order accuracy (but the same stability characteristics as Euler's method).

Heun's method is an example of a predictor-corrector technique. In order to simulate any time step *j*, Heun's method begins with an Euler step to obtain an initial prediction for the solution at the conclusion of the time step ($\hat{y}_{j+1}$). This first step in the solution is referred to as the predictor step and the details are essentially identical to Euler's method:

$$\hat{y}_{j+1} = y_j + \left.\frac{dy}{dt}\right|_{y=y_j, t=t_j} \Delta t \tag{7-17}$$

However, Heun's method uses the results of the predictor step to carry out a corrector step. The solution predicted at the end of the time step ($\hat{y}_{j+1}$) is used to predict the rate of change at the

end of the time step ($\left.\frac{dy}{dt}\right|_{y=\hat{y}_{j+1},t=t_{j+1}}$). The corrector step predicts the solution at the end of the time step (y_{j+1}) based on the average of the time rates of change at the beginning and end of the time step:

$$y_{j+1} = y_j + \left[\left.\frac{dy}{dt}\right|_{y=y_j,t=t_j} + \left.\frac{dy}{dt}\right|_{y=\hat{y}_{j+1},t=t_{j+1}}\right]\frac{\Delta t}{2} \tag{7-18}$$

Heun's method is illustrated in the context of the simple problem discussed previously. The process of moving through the first time step begins with the predictor step:

$$\hat{y}_2 = y_1 + \left.\frac{dy}{dt}\right|_{y=y_1,t=t_1}\Delta t \tag{7-19}$$

where Eq. (7-1) is used to evaluate $\left.\frac{dy}{dt}\right|_{y=y_1,t=t_1}$.

```
"take the first step"
dydt[1]+4*t[1]*y[1]=-2*t[1]                "time rate of change at t[1]"
y_hat[2]=y[1]+dydt[1]*Dt                   "predictor step"
```

The corrector step follows:

$$y_2 = y_1 + \left[\left.\frac{dy}{dt}\right|_{y=y_1,t=t_1} + \left.\frac{dy}{dt}\right|_{y=\hat{y}_2,t=t_2}\right]\frac{\Delta t}{2} \tag{7-20}$$

where Eq. (7-1) is also used to evaluate $\left.\frac{dy}{dt}\right|_{y=\hat{y}_2,t=t_2}$.

```
dydt_hat[2]+4*t[2]*y_hat[2]=-2*t[2]        "time rate of change at t[2] based on y_hat[2]"
y[2]=y[1]+(dydt[1]+dydt_hat[2])*Dt/2       "corrector step"
```

Once we have established that it is possible to take one step it is easy to take all of the steps. Comment out the code used to take a single step:

```
{"take the first step"
dydt[1]+4*t[1]*y[1]=-2*t[1]                "time rate of change at t[1]"
y_hat[2]=y[1]+dydt[1]*Dt                   "predictor step"
dydt_hat[2]+4*t[2]*y_hat[2]=-2*t[2]        "time rate of change at t[2] based on y_hat[2]"
y[2]=y[1]+(dydt[1]+dydt_hat[2])*Dt/2       "corrector step"}
```

and setup a duplicate loop that goes from $j = 1$ to $j = (M - 1)$. Copy the code required to take one step and place it inside the duplicate loop. Change the element index 1 to j and the element index 2 to $j + 1$.

```
"take all of the steps"
duplicate j=1,(M-1)
    dydt[j]+4*t[j]*y[j]=-2*t[j]                    "time rate of change at t[j]"
    y_hat[j+1]=y[j]+dydt[j]*Dt                     "predictor step"
    dydt_hat[j+1]+4*t[j+1]*y_hat[j+1]=-2*t[j+1]    "time rate of change at t[j+1] based on y_hat[j+1]"
    y[j+1]=y[j]+(dydt[j]+dydt_hat[j+1])*Dt/2       "corrector step"
end
```

Figure 7-5 illustrates the analytical solution as well as the Euler's solution and Heun's solution with $\Delta t = 0.125$ s. There is an improvement in accuracy associated with the use of Heun's technique for the same time step duration. Figure 7-3 illustrates the global error for Heun's technique as a function of the time step and shows this improvement in accuracy. Notice that the global error is proportional to the time step to the second power; Heun's technique is second order with respect to global error.

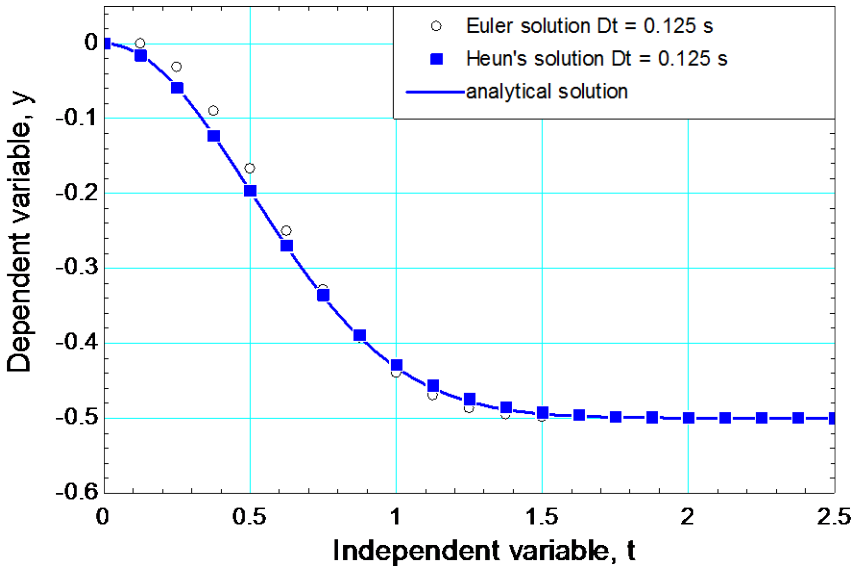


Figure 7-5: Analytical and numerical solutions with Euler's and Heun's technique for $\Delta t = 0.125$ s.

Heun's method is a two-step predictor/corrector technique that improves the order of accuracy by one (i.e., the order of Heun's method is two whereas the order of Euler's method is one). It is possible to carry out additional predictor/corrector steps that further improve the accuracy of the numerical solution. One of the most popular higher order techniques is the fourth order Runge-Kutta method (which involves four predictor/corrector steps and is therefore fifth order accurate).

The Fully Implicit Method (First Order Implicit)

The methods discussed thus far are explicit; they all therefore share the characteristic of becoming unstable when the time step exceeds a critical value. An implicit technique avoids this problem. The fully implicit first-order method is similar to Euler's method in that the time rate of change is assumed to be constant throughout the time step. However, the time rate of change is computed at the end of the time step rather than the beginning. Therefore, for any time step j:

$$y_{j+1} = y_j + \left.\frac{dy}{dt}\right|_{y=y_{j+1},t=t_{j+1}} \Delta t \tag{7-21}$$

The time rate of change at the end of the time step depends on the solution at the end of the time step (y_{j+1}). Therefore, y_{j+1} cannot be calculated explicitly using information that is available at the beginning of the time step (y_j) and instead an implicit equation is obtained for y_{j+1}. Because EES solves implicit equations automatically, it is easy to implement the fully implicit technique.

```
"Fully implicit technique"
duplicate j=1,(M-1)
    dydt[j+1]+4*t[j+1]*y[j+1]=-2*t[j+1]        "time rate of change at t[j+1] and y[j+1]"
    y[j+1]=y[j]+dydt[j+1]*Dt                   "fully implicit solution"
end
```

The fully implicit solution does not become unstable even when the duration of the time step is greater than the critical time step. Figure 7-6 illustrates the analytical solution as well as the Euler and fully implicit solution with Δt = 0.625 s. Notice that the fully implicit solution remains stable while the Euler's solution does not.

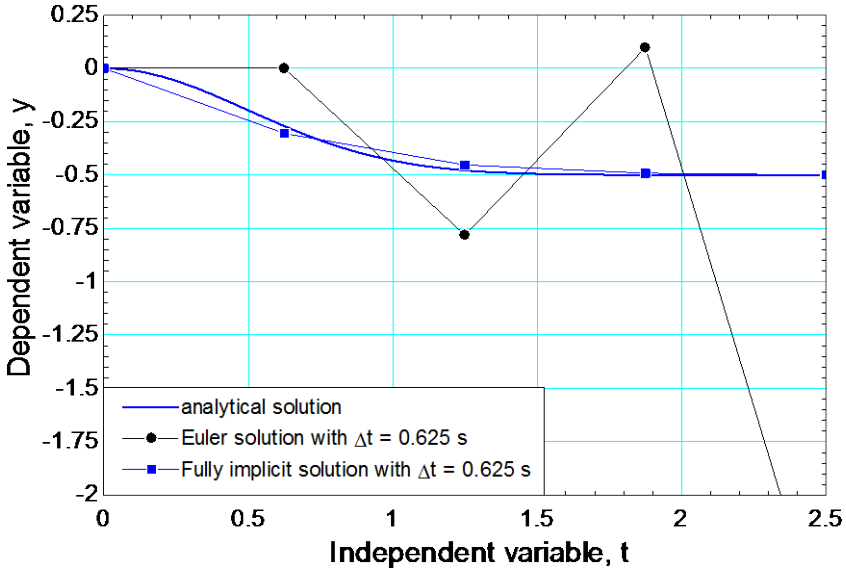


Figure 7-6: Euler's technique and fully implicit technique with Δt = 0.625 s.

The Crank-Nicolson Method (Second Order Implicit)

The Crank-Nicolson method combines Euler's method with the fully implicit method. The time rate of change for the time step is estimated based on the average of its values at the beginning and end of the time step. Therefore, for any time step *j*:

$$y_{j+1} = y_j + \left[\left.\frac{dy}{dt}\right|_{y=y_j,t=t_j} + \left.\frac{dy}{dt}\right|_{y=y_{j+1},t=t_{j+1}} \right] \frac{\Delta t}{2} \tag{7-22}$$

Notice that the Crank-Nicolson is an implicit method because the solution for y_{j+1} involves a time rate of change that must be evaluated based on y_{j+1}. Therefore, the technique will have the stability characteristics of the fully implicit method. The solution also involves two estimates for the time rate of change and will therefore be second order accurate with respect to the global error. Because EES solves implicit equations it is relatively easy to implement a Crank-Nicolson solution using EES:

```
"Crank-Nicolson technique"
duplicate j=1,M
    dydt[j]+4*t[j]*y[j]=-2*t[j]                "time rate of change at t[j] and y[j]"
end
duplicate j=1,(M-1)
    y[j+1]=y[j]+(dydt[j]+dydt[j+1])*Dt/2       "Crank-Nicolson solution"
end
```

The Crank-Nicolson method is a popular choice because it is simple yet it combines high accuracy with stability.

The introduction to numerical integration provided in this section should make the difference between implicit and explicit techniques clear and emphasize the importance of the order of the integration technique and the size of the step that is taken. In this section, the various integration techniques were implemented manually, taking steps using a duplicate loop and storing all of the intermediate variables in arrays. This process is inconvenient and EES offers a better alternative. The EES Integral command provides a high order, semi-implicit adaptive step-size integration technique that will automatically operate on one or more state equations that are entered in the Equations Window.

7.2 Equation-Based Integral Function

The implementation of the techniques discussed in Section 7.1 used a fixed duration time step for the entire simulation. This implementation is often not efficient because there are regions of time during the simulation where the solution is not changing substantially and therefore large time steps could be taken with little loss of accuracy. Adaptive step-size solutions adjust the size of the time step that is used based on the local characteristics of the state equation. Typically, the absolute value of the local time rate of change or the second derivative of the time rate of change is used to establish a step-size that is as large as possible while guaranteeing a specified level of accuracy. For the current example, shown in Figure 7-1, smaller time steps would be used near $t = 0$ s because the solution is changing quickly at this time. Later in the simulation, for $t > 2$ s, the solution is not changing substantially and therefore large time steps could be used.

Calling Protocol for the Integral Command

An iterative, second-order numerical integration routine that optionally uses an adaptive step-size is provided with EES and can be accessed using the equation-based Integral command. There is another form of Integral command referred to as the table-based Integral command, which is described in Section 7.3. The equation-based Integral command requires four arguments and allows an optional fifth argument:

```
F = Integral(Integrand, VarName, LowerLimit, UpperLimit, StepSize)
```

where Integrand is the EES variable or expression that must be integrated, VarName is the integration variable, LowerLimit and UpperLimit define the limits of integration, and StepSize (optionally) provides the duration of the time step.

Entering the State Equations

It is important to remember that the Integral command expects that there is a system of equations in the Equations Window that will, when provided with the value(s) of the dependent variable(s) and the integration variable, provide the time rate of change of the dependent variable(s). EES will automatically manipulate this system of equations in order to accomplish the integration and move through each time step. With this in mind, it is useful to begin the process of using the Integral command by first specifying some set of values for the dependent variable(s) and time and verifying that the equations in the Equations Window are capable of computing the necessary derivatives.

For this problem, the first step would be to set some arbitrary value of the dependent variable (y) and time (t):

```
"Integral Solution"
y=0                                   "arbitrary value of y"
t=0                                   "arbitrary value of t"
```

Next, enter the equations necessary to provide the time rate of change of y. In this case we need only to program Eq. (7-1):

```
dydt+4*t*y=-2*t                    "state equation"
```

Solving leads to dydt = 0.

Carrying out the Integration

Once we have entered the state equations, the next step is to comment out the arbitrary values of *y* and *t* that are used to test the computation of the state equation and instead let EES' Integral function control these variables for the numerical integration. The solution is given by:

$$y = y_{ini} + \int_0^{t_{sim}} \frac{dy}{dt} dt \tag{7-23}$$

where $y_{ini} = 0$ is the initial condition. Therefore, the solution to our example problem is obtained by calling the Integral function with Integrand replaced with the variable dydt, VarName replaced with the variable t, LowerLimit replaced with 0, UpperLimit replaced with the variable t_sim, and StepSize with Dt, the specified duration of the time step:

```
"Integral Solution"
{y=0                               "arbitrary value of y"
t=0                                "arbitrary value of t"}
dydt+4*t*y=-2*t                    "state equation"

t_sim=2.5                          "simulation time"
Dt=0.1                             "time step duration"
y=Integral(dydt,t,0,t_sim,Dt)      "solution obtained using the integral command"
```

Note that array variables and duplicate loops are not needed for the numerical integration when the EES Integral function is used. The elimination of array variables allows much larger problems to be solved and improves the computational speed.

In order to accomplish the numerical integration, EES will adjust the value of the variable t from 0 to t_sim in increments of Dt. At each value of time, EES will iteratively solve all of the equations in the Equations Window that directly or indirectly depend on *t*. For the example above, this process will result in the variable y being evaluated at each value t. When the solution converges, the value of the variable t is incremented and the process is repeated until the variable t becomes equal to t_sim. The result is shown in the Solution Window, Figure 7-7, which shows the value of all of the variables at the end of the process (i.e., the result of the integration, which is *y* at $t = t_{sim}$).

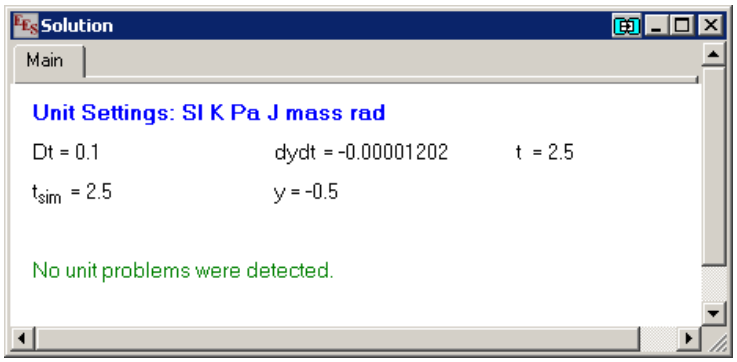


Figure 7-7: Solution Window.

The Integral Table

Often it is interesting to know the trajectory of the solution with time during the process that is being simulated. This information can be provided by including the $IntegralTable directive in the file. The format of the $IntegralTable directive is:

```
$IntegralTable VarName: Step, x, y, z
```

where VarName is the integration variable and x, y, z are dependent variables defined in the Main section of the Equation Window that are of interest. The variables must be separated by a space or list delimiter (comma or semicolon). Algebraic equations involving variables are not accepted within the $Integral directive. The first column in the Integral Table will contain values of the integration variable and the remaining columns will contain the values of the other variables. The colon followed by the parameter Step (which can either be a number or a variable name) is optional. If a value is provided for Step, then the integration variables will be reported in the Integral Table at increments of the integration variable specified by Step. The output step size may be a variable name, rather than a number, provided that the variable has been set to a constant value preceding the $IntegralTable directive. The step size that is used to report integration results is totally independent of the duration of the time step that is used in the numerical integration. If the numerical integration step size and output step size are not the same, then linear interpolation is used to determine the integrated quantities at the specified values of the integration variable. If the colon and an output step size is not specified, then EES will output all specified variables at every time step.

Solving the EES code will result in the generation of an Integral Table that is filled with intermediate values resulting from the numerical integration. The values in the Integral Table can be plotted, printed, saved, and copied in exactly the same manner as for other tables. The Integral Table is saved when the EES file is saved and the table is restored when the EES file is loaded. If an Integral Table exists when calculations are initiated, it will be deleted if a new Integral Table is created by the calculations.

Use the EES code below to generate an Integral Table containing the results of the numerical simulation as well as the analytical solution:

```
$IntegralTable t, y
```

After solving the code with a time step duration, Dt, of 0.1 s, the Integral Table shown in Figure 7-8 will be generated. Note that the result of every time step is included in the Integral Table because the optional colon and output step size were not provided.

Integral Table

	1 t	2 y
Row 1	0	0
Row 2	0.1	-0.009804
Row 3	0.2	-0.03808
Row 4	0.3	-0.08166
Row 5	0.4	-0.1359
Row 6	0.5	-0.1955
Row 7	0.6	-0.2553
Row 8	0.7	-0.3111

Figure 7-8: **Integral Table generated by EES.**

The results of the integration can be plotted by selecting the Integral Table as the source of the data to plot. Figure 7-9 shows the results obtained from the Integral command with a fixed step size overlayed onto the analytical solution.

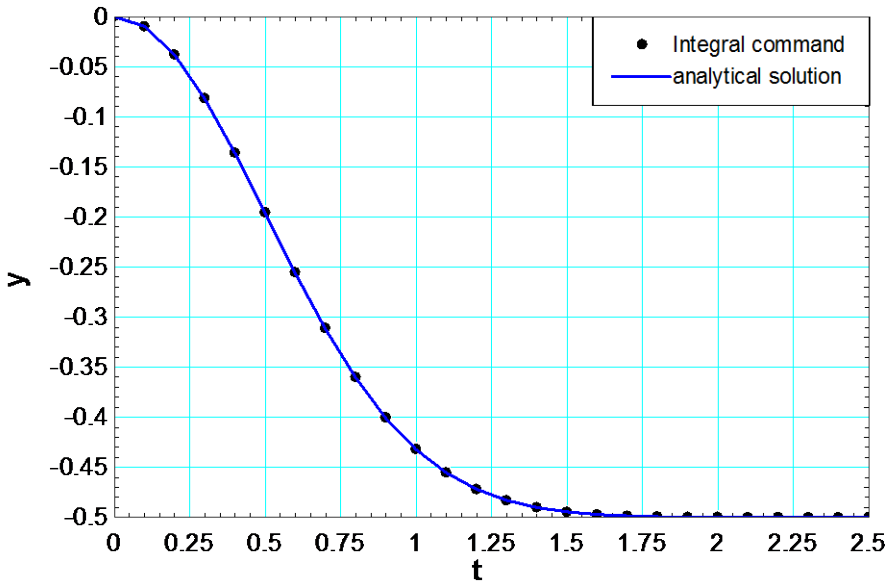


Figure 7-9: Analytical solution and the results obtained from the Integral command with a fixed step size.

Adaptive Step Size

The StepSize input to the Integral command is optional. If the parameter StepSize is not included or if it is set to a value of 0, then EES will use an adaptive step-size algorithm that maintains accuracy while maximizing computational speed. The parameters used to control the adaptive step-size algorithm can be accessed and adjusted by selecting Preferences from the Options menu and selecting the Integration tab, shown in Figure 7-10. The settings can also be provided with a $IntegralAutoStep directive.

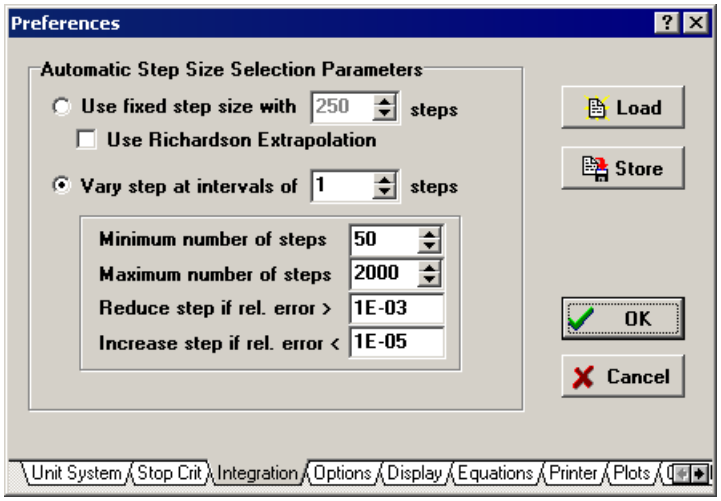


Figure 7-10: Integration preferences dialog.

The user can specify how often the step interval will be examined and adjusted, as well as the absolute minimum and maximum number of integration steps that will be allowed. The relative error criteria used to either reduce or increase the step size can also be specified. It is advisable to use the $IntegralAutoStep directive to set the unit system so that your code produces consistent results across various computers and is as transparent as possible. For example, the code:

```
$IntegralAutoStep  Vary = 1  Min = 5  Max = 2000  Reduce = 1e-1  Increase = 1e-3
```

specifies each of these integration parameters. The integration step size is varied after each step (Vary = 1) and the number of steps must be greater than 5 (Min = 5) and less than 2000 (Max = 2000)[2]. The integration step will be reduced if the relative error is larger than $1x10^{-1}$ (Reduce = 1e-1) and increased if the relative error is less than $1x10^{-3}$ (Increase = 1e-3).

Removing the specified time step from the Integral command and including the $IntegralAutoStep directive:

```
$IntegralAutoStep  Vary=1  Min=5  Max=2000  Reduce=1e-1  Increase=1e-3
"Integral Solution"
{y=0                                    "arbitrary value of y"
t=0                                     "arbitrary value of t"}
dydt+4*t*y=-2*t                         "state equation"
```

[2] The minimum number of steps was set to 5 (rather than 50) so that the non-uniform step duration can be seen in Figure 7-11.

```
t_sim=2.5                                "simulation time"
y=Integral(dydt,t,0,t_sim)               "solution obtained using the integral command"
$IntegralTable t, y                      "save results in an Integral table"
```

will generate the numerical solution shown in Figure 7-11. Notice the non-uniform time step duration that is selected to maintain accuracy while maximizing the computational speed.

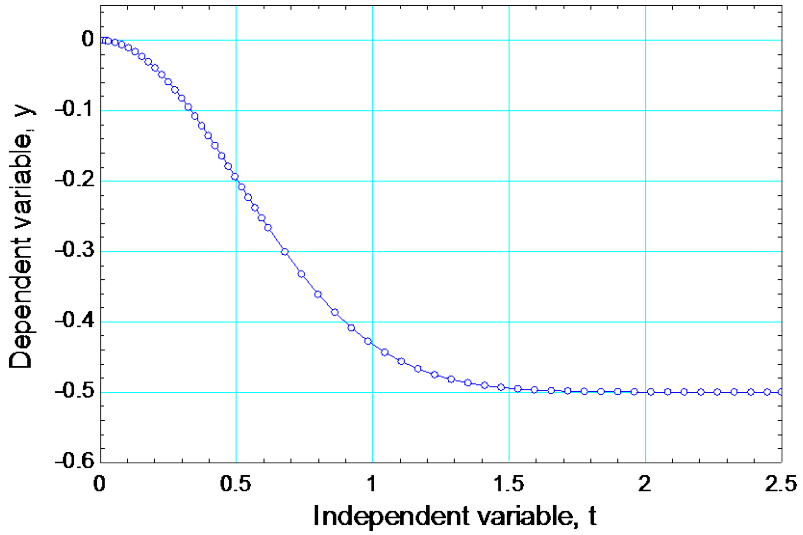


Figure 7-11: Numerical solution obtained with EES' Integral command.

Integrating Simultaneous ODEs

Systems of ODEs occur in the solution of coupled or higher order dynamic systems as well as in finite difference solutions to transient problems. Multiple Integral commands can be used together to numerically integrate a system of coupled ODEs. Note that EES will evaluate the integrands of all of the equations together as it takes each integration step. Therefore, it must use the same integration step size and limits for each of the integrals. Either the same value of the step size must be specified or adaptive step size selection must be used for all of the integrals.

Consider the two mass system shown in Figure 7-12.

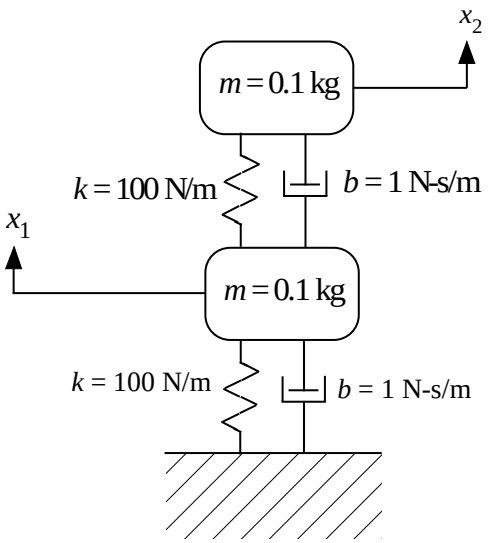


Figure 7-12: Two mass dynamic system.

The top mass is initially displaced an amount $x_{2,ini}$ = 0.05 m from its equilibrium position and then released. These inputs are entered in EES.

```
$UnitSystem SI Mass J kg K Pa
$IntegralAutoStep  Vary=1  Min=5  Max=2000  Reduce=1e-1  Increase=1e-3

"Inputs"
m=0.1 [kg]                                  "mass"
k=100 [N/m]                                 "spring constant"
b=1 [N-s/m]                                 "damper"

x1_ini=0 [m]                                "initial position of mass 1"
x2_ini=0.05 [m]                             "initial position of mass 2"
v1_ini=0 [m/s]                              "initial velocity of mass 1"
v2_ini=0 [m/s]                              "initial velocity of mass 2"
```

We are interested in predicting the position of the two masses (x_1 and x_2) as a function of time as they return to equilibrium. The dependent variables for this situation are x_1 and x_2 as well as v_1 and v_2, the velocities of the masses. In order to implement any numerical integration technique, it is necessary to develop the state equations for the problem. That is, we must be able to predict the time derivatives of x_1, x_2, v_1, and v_2 given their values and the value of time. Initially, arbitrary values for the state variables are entered:

```
"dependent variables and time"
x1=x1_ini                                   "position of mass 1"
v1=v1_ini                                   "velocity of mass 1"
x2=x2_ini                                   "position of mass 2"
v2=v2_ini                                   "velocity of mass 2"
t=0 [s]                                     "time"
```

The time derivatives for x_1 and x_2 are simply their velocities:

$$\frac{dx_1}{dt} = v_1 \tag{7-24}$$

$$\frac{dx_2}{dt} = v_2 \tag{7-25}$$

```
dx1dt=v1                                    "derivative of position is velocity"
dx2dt=v2
```

The derivatives of the velocities (accelerations) are obtained from force balances on each mass. Neglecting gravity, the force balance on the top mass provides the time rate of change of v_2:

$$-k\,x_2 + k\,x_1 - b\,v_2 + b\,v_1 = m\frac{dv_2}{dt} \tag{7-26}$$

The force balance on the lower mass provides the time rate of change of v_1:

$$k\,x_2 - 2\,k\,x_1 + b\,v_2 - 2\,b\,v_1 = m\frac{dv_1}{dt} \tag{7-27}$$

```
-k*x2+k*x1-b*v2+b*v1=m*dv2dt                "force balances"
k*x2-2*k*x1+b*v2-2*b*v1=m*dv1dt
```

Solving provides all of the state variable derivatives; therefore, we have successfully implemented the state equations and can move on to the numerical integration. Comment out the specified values of the state variables.

```
{"dependent variables and time"
x1=x1_ini                                   "position of mass 1"
v1=v1_ini                                   "velocity of mass 1"
x2=x2_ini                                   "position of mass 2"
v2=v2_ini                                   "velocity of mass 2"
t=0 [s]                                     "time"}
```

Implement the integration of each of the state variables using four Integral commands.

$$x_1 = x_{1,ini} + \int_0^{t_{sim}} \frac{dx_1}{dt}\,dt \tag{7-28}$$

$$x_2 = x_{2,ini} + \int_0^{t_{sim}} \frac{dx_2}{dt}\,dt \tag{7-29}$$

$$v_1 = v_{1,ini} + \int_0^{t_{sim}} \frac{dv_1}{dt}\,dt \tag{7-30}$$

$$v_2 = v_{2,ini} + \int_0^{t_{sim}} \frac{dv_2}{dt} dt \tag{7-31}$$

```
"Integrate state equations"
x1=x1_ini+Integral(dx1dt,t,0,t_sim)
v1=v1_ini+Integral(dv1dt,t,0,t_sim)
x2=x2_ini+Integral(dx2dt,t,0,t_sim)
v2=v2_ini+Integral(dv2dt,t,0,t_sim)
```

The results are stored at each time step in an Integral Table using the $IntegralTable directive.

```
$IntegralTable t, x1,v1,x2,v2
```

Solving provides the result shown in Figure 7-13.

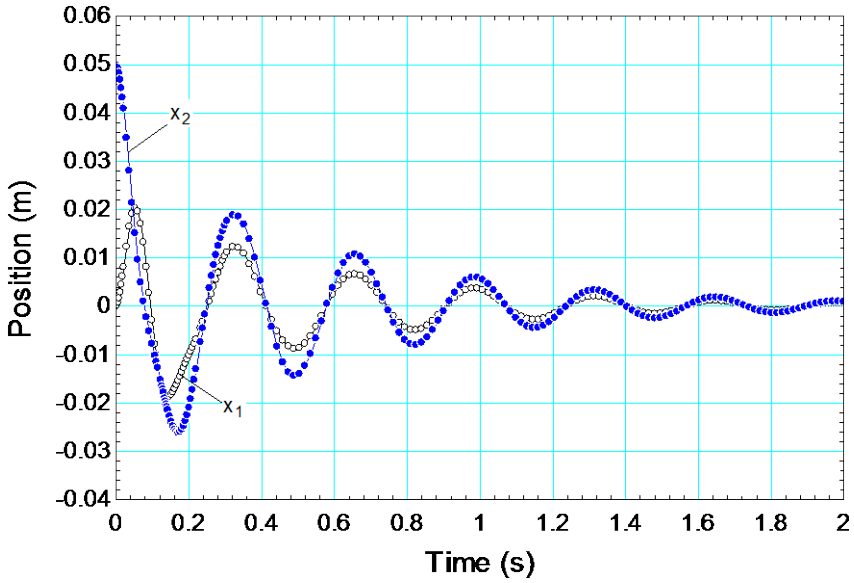


Figure 7-13: Positions of the two masses (relative to their equilibrium position) as a function of time.

The IntegralValue Command

The IntegralValue command is used to retrieve data from the Integral Table so that it can be used within the Equations Window. The calling protocol for the IntegralValue command is:

```
IntegralValue(t,'X')
```

where t is the value of the integration variable at which the value of the variable X will be determined. The variable X must be one of the variables contained in the Integral Table.

There are many reasons that you might want to use data stored within the Integral Table. One common reason for using the IntegralValue command is to retrieve a temperature or pressure at a previous value of time so that the properties required by the state equation can be evaluated

explicitly (i.e., using previous temperatures that are already known) as opposed to implicitly (i.e.,. using the unknown temperature at the end of the current time step).

Consider the flow of carbon dioxide through a round tube, shown in Figure 7-14.

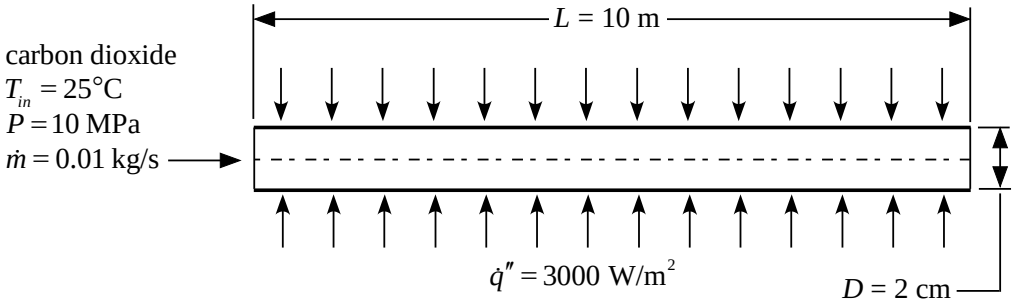


Figure 7-14: Flow of carbon dioxide through a tube subjected to a constant heat flux.

The inlet temperature of the carbon dioxide is T_{in} = 25ºC and the mass flow rate is $\dot{m}$ = 0.01 kg/s. The tube is L = 10 m long and it has an outer diameter of D = 2 cm. The outer surface of the tube is exposed to a constant heat flux, $\dot{q}''$ = 3000 W/m^2. The pressure of the carbon dioxide may be assumed to be constant and can vary from 7.4 MPa to 10.0 MPa; initially assume that the pressure is 10 MPa. The inputs are entered in EES:

```
$TabStops 0.2 0.4 3 in
$UnitSystem SI Mass J Pa K Radian
$IntegralAutoStep  Vary=1  Min=5  Max=2000  Reduce=1e-1  Increase=1e-3

"Inputs"
m_dot=0.01 [kg/s]                    "mass flow rate"
T_in=ConvertTemp(C,K,25 [C])         "inlet temperature"
D=2 [cm]*convert(cm,m)               "diameter"
L=10 [m]                             "length"
q``=3000 [W/m^2]                     "heat flux"
P=10 [MPa]*convert(MPa,Pa)           "pressure"
```

We are interested in predicting the bulk temperature of the carbon dioxide as a function of position in the tube. The state variable is temperature (T) and the integration variable is position (x). Initially, arbitrary values of these variables are defined:

```
"Dependent variables"
x=0.1 [m]                            "position"
T=300 [K]                            "temperature"
```

A differential energy balance on the fluid leads to:

$$\dot{m}\,h + \dot{q}''\,\pi\,D\,dx = \dot{m}\,h + \dot{m}\frac{dh}{dx}dx \tag{7-32}$$

where h is specific enthalpy. Expanding the specific enthalpy term and recognizing that pressure is constant provides:

$$\dot{q}''\,\pi\,D = \dot{m}\left(\frac{\partial h}{\partial T}\right)_P \frac{dT}{dx} \tag{7-33}$$

The partial derivative of specific enthalpy with respect to temperature at constant pressure is the specific heat capacity at constant pressure (c_P); therefore, the state equation for this problem is:

$$\frac{dT}{dx} = \frac{\dot{q}''\,\pi\,D}{\dot{m}\,c_P(T)} \tag{7-34}$$

The specific heat capacity is evaluated at the temperature (T) and pressure (P):

```
c=cP(CarbonDioxide,T=T,P=P)                    "specific heat capacity"
```

and the state equation, Eq. (7-34), is implemented:

```
dTdx=q``*pi*D/(m_dot*c)                        "rate of change of temperature"
```

Solving leads to dTdx = 6.312 K/m. Comment out the specified values of x and T

```
"Dependent variables"
{x=0.1 [m]                                     "position"
T=300 [K]                                      "temperature"}
```

and utilize the Integral command to integrate the state equation.

$$T = T_{in} + \int_0^L \frac{dT}{dx}\,dx \tag{7-35}$$

```
T=T_in+Integral(dTdx,x,0,L)                    "integrate state equations"
T_C=ConvertTemp(K,C,T)                         "temperature, in C"
$IntegralTable x,T,T_C,c
```

Solving leads to the result shown in Figure 7-15.

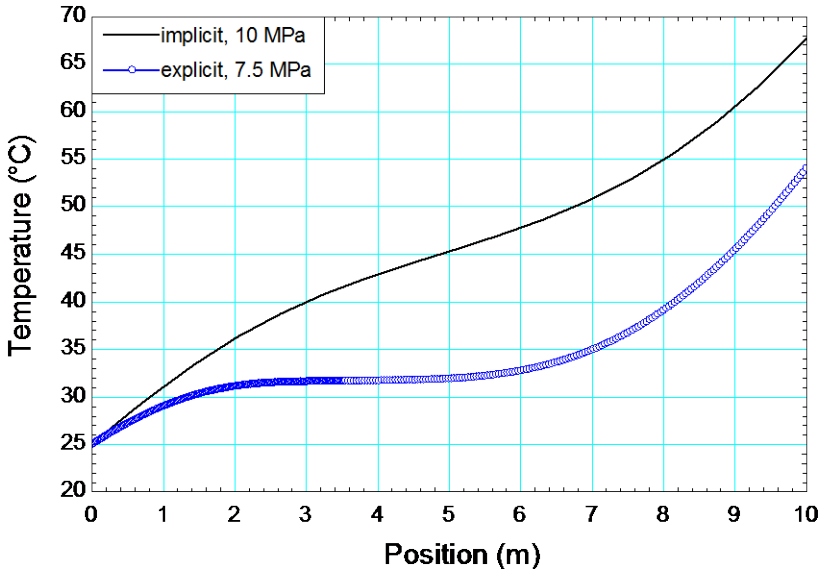


Figure 7-15: Temperature as a function of position.

If the pressure is reduced from 10 MPa to 7.5 MPa

```
P=7.5 [MPa]*convert(MPa,Pa)                    "pressure"
```

then you are likely to encounter the error message shown in Figure 7-16.

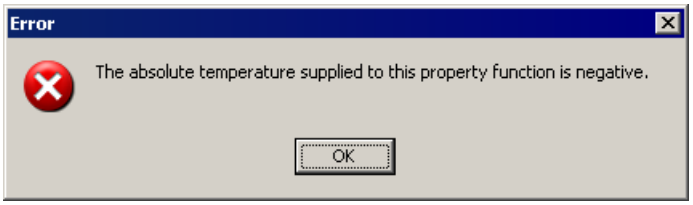


Figure 7-16: Error message.

This error occurs because EES uses a semi-implicit numerical solution algorithm and therefore very large variations in the properties with temperature may cause it to be unable to converge during a particular step. Figure 7-17(a) illustrates a temperature-specific entropy diagram for carbon dioxide (made using a property plot generated in EES, as discussed in Section 4.6). Note that the pressures and temperatures considered in this problem are very near the critical point (T_{crit} = 30.98ºCand P_{crit} = 7.377 MPa). Figure 7-17(b) illustrates the specific heat capacity as a function of temperature for various values of pressure and shows that c_P varies dramatically with temperature as the pressure is reduced towards the critical pressure.

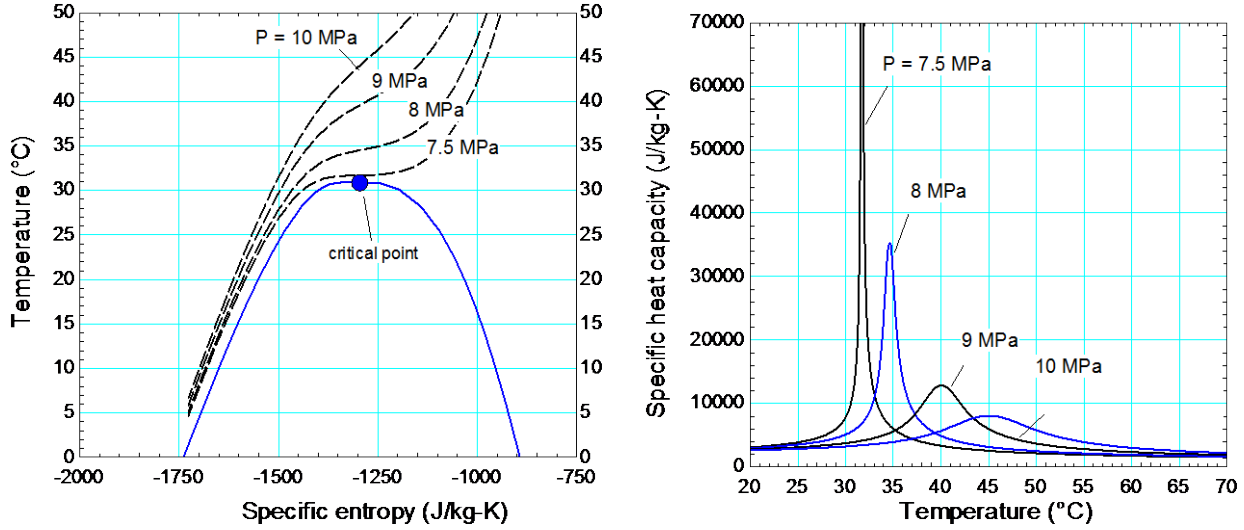


Figure 7-17: (a) Temperature-specific entropy diagram for carbon dioxide and (b) specific heat capacity of carbon dioxide as a function of temperature for several values of pressure near the critical pressure.

One method of overcoming this problem with the numerical integration is to make the algorithm explicit in terms of the properties; that is, the value of c_P will always be evaluated using the temperature from the previous step in the integration. The resulting value of c_P will be assumed to be constant during the current integration step. This approach is made possible using the IntegralValue function.

We need to write a function T_last that returns the value of the temperature at the previous length step unless $x = 0$ in which case it returns the inlet temperature. The function is placed at the top of the EES code.

```
Function T_last(x,T_in)
    If (x=0) Then
        T_last=T_in
    Else
        T_last=IntegralValue(x-0.001 [m],T)
    EndIf
End
```

Rather than compute c_P using the value of T that is being adjusted actively in order to take the integration step,

```
{c=cP(CarbonDioxide,T=T,P=P)                    "specific heat capacity"}
```

the explicit technique obtains the value of T from the previous time step using the function T_last and uses that temperature to evaluate c_P:

```
"Explicit"
Tl=T_last(x,T_in)                               "last value of temperature"
c=cP(CarbonDioxide,T=Tl,P=P)                    "specific heat capacity"
```

Solving provides the result shown in Figure 7-15.

Double Integration

Numerical integration with two variables can be nested in order to carry out double integration. As an example, consider a rectangular block with an initial temperature distribution given by:

$$T_{ini}(x,y) = 300[\mathrm{K}] + 10\left[\frac{\mathrm{K}}{\mathrm{m}^2}\right]x^2 + 20\left[\frac{\mathrm{K}}{\mathrm{m}^2}\right]y^2 \tag{7-36}$$

The width of the block in the *x*-direction is $W = 2$ m and the height in the *y*-direction is $H = 1$ m.

```
$UnitSystem SI Mass J K Pa Rad

W=2 [m]                    "width"
H=1 [m]                    "height"
```

The initial temperature is shown in Figure 7-18 on a contour plot; note that the generation of contour plots is discussed in Section 9.3.

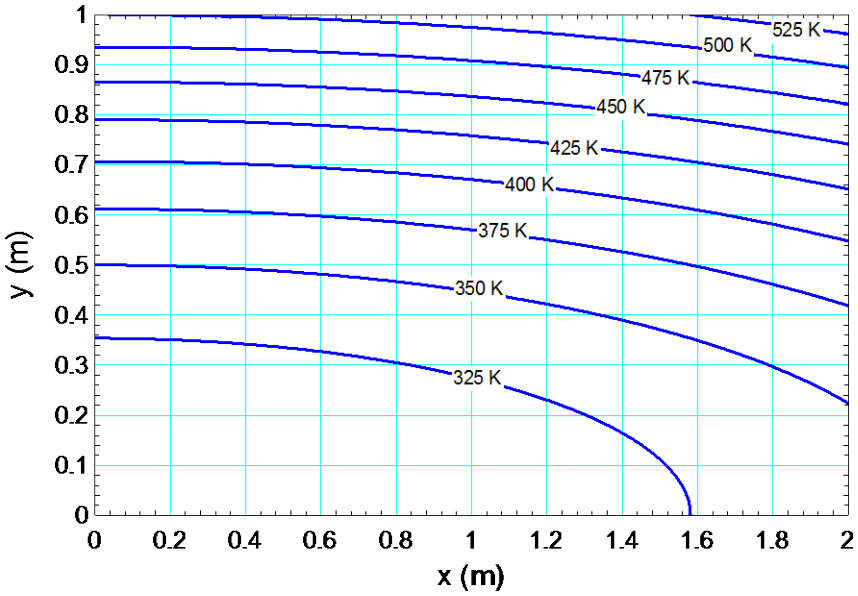


Figure 7-18: Initial temperature distribution.

If the edges of the block are insulated then the final, uniform temperature of the block can be determined using an energy balance. We will assume that the material has a constant specific heat capacity. The initial internal energy per unit length is:

$$U_{ini} = \int_{y=0}^{H}\int_{x=0}^{W} cT(x,y)\,dx\,dy \tag{7-37}$$

The final internal energy per unit length is:

$$U_{final} = cT_{final}\,W\,H \tag{7-38}$$

If the block is adiabatic, then an energy balance provides:

$$U_{ini} = U_{final} \tag{7-39}$$

Substituting Eqs. (7-37) and (7-38) into Eq. (7-39) provides:

$$\int_{y=0}^{H} \int_{x=0}^{W} c\,T(x,y)\,dx\,dy = c\,T_{final}\,W\,H \tag{7-40}$$

Equation (7-40) can be solved for the final temperature:

$$T_{final} = \frac{1}{W\,H} \overbrace{\int_{y=0}^{H} \underbrace{\int_{x=0}^{W} T(x,y)\,dx}_{\text{inner integral}}\,dy}^{\text{outer integral}} \tag{7-41}$$

As with the integrations discussed thus far, it is worthwhile making sure that the integrand of the double integral (T) can be evaluated at arbitrary values of the integration variables (x and y).

```
"arbitrary values of integration variables"
x=0 [m]
y=0 [m]

"integrand"
T=300 [K]+10 [K/m^2]*x^2+200 [K/m^2]*y^2        "temperature"
```

Once the integrand has been programmed, the specified values of x and y should be commented out:

```
{x=0 [m]
y=0 [m]}
```

The double integral in Eq. (7-41) can be evaluated using two nested Integral commands. The inner integral is:

```
Integral(T,x,0,W)
```

So the double integral is:

```
T_final=Integral(Integral(T,x,0,W),y,0,H)/(W*H)    "final temperature"
```

Solving provides T_{final} = 380 K. Note that the double integral could have equivalently been evaluated according to:

```
InnerIntegral=Integral(T,x,0,W)                    "inner integral"
T_final=Integral(InnerIntegral,y,0,H)/(W*H)        "final temperature"
```

7.3 Table-Based Integral Function

The Integral command can also be used in conjunction with a Parametric Table. The table-based Integral function uses the entries in the Parametric Table to provide the limits and the step size. Therefore, the calling protocol for the table-based Integral function is:

```
F = Integral(Integrand, VarName)
```

where F is the integral and VarName is the name of the integration variable. The variables F and VarName must be included in the Parametric Table and values of the variable VarName must be specified in order to provide the limits and step size. It is not necessary that the step size be constant. The variable Integrand must be calculated in the Equations Window.

As an example, we will numerically integrate the expression:

$$\frac{dy}{dt} + 4t\,y = -2t \tag{7-42}$$

from $t = 0$ to $t = 2.5$. First, set up the Parametric Table containing both the integration variable (t) and the dependent variable (y). Set the values of t in order to specify the limits and step size. The result is shown in Figure 7-19(a).

Parametric Table

Table 1

1..11	t	y
Run 1	0	
Run 2	0.25	
Run 3	0.5	
Run 4	0.75	
Run 5	1	
Run 6	1.25	
Run 7	1.5	
Run 8	1.75	
Run 9	2	
Run 10	2.25	
Run 11	2.5	

Parametric Table

Table 1

1..11	t	y
Run 1	0	0
Run 2	0.25	-0.05556
Run 3	0.5	-0.1889
Run 4	0.75	-0.3303
Run 5	1	-0.4293
Run 6	1.25	-0.4782
Run 7	1.5	-0.4953
Run 8	1.75	-0.4994
Run 9	2	-0.5
Run 10	2.25	-0.5
Run 11	2.5	-0.5

Figure 7-19: Parametric Table (a) before solving and (b) after solving.

Enter Eq. (7-42) in the Equations Window in order to evaluate the derivate of y with respect to t. The value of y at any time (t) specified in the Parametric Table is found using the Integral command. The analytical solution can also be entered for comparison to the numerical solution.

```
dydt + 4*t*y = -2*t                    "differential equation"
```

```
y=Integral(dydt,t)                    "numerical integration"
y_an = -1/2+exp(-2*t^2)/2             "analytical solution"
```

Select Solve Table from the Calculate menu in order to fill in the values of *y*, as shown in Figure 7-19(b). The solution is shown in Figure 7-20 together with the analytical solution given by Eq. (7-8).

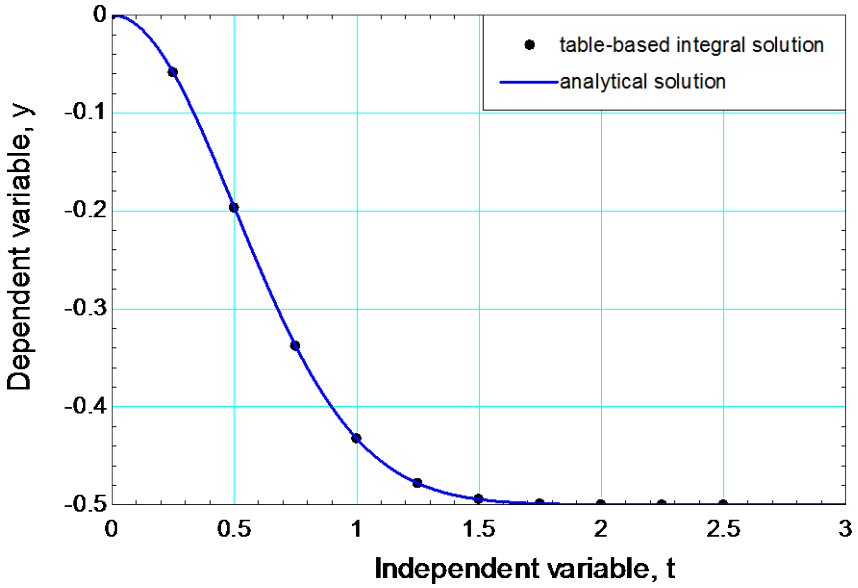


Figure 7-20: Solution obtained using the table-based integral function and the analytical solution.

7.4 Solving Partial Differential Equations

The Integral command can be used to solve a partial differential equations (PDE). Discretization is applied in order to approximate the PDE as a set of coupled ODEs. Coupled ODEs can be solved by simultaneous application of multiple Integral commands, as discussed in Section 7.2. As an example, consider the case of a plane wall that is subjected to a convective boundary condition on one surface, as shown in Figure 7-21.

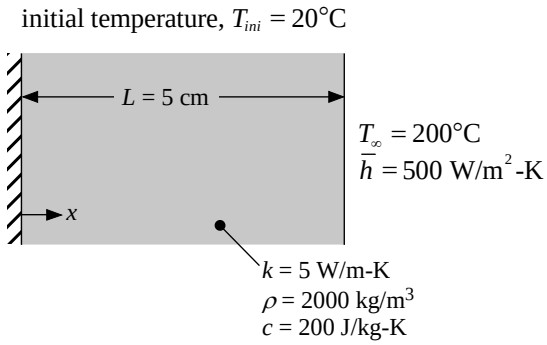


Figure 7-21: A plane wall exposed to a convective boundary condition at time t = 0.

The plane wall has thickness L = 5.0 cm and properties k = 5.0 W/m-K, ρ = 2000 kg/m^3, and c = 200 J/kg-K. The wall is initially at T_{ini}= 20ºC when at time t = 0, the surface (at $x = L$) is exposed to fluid at T_∞ = 200ºC with average heat transfer coefficient $\bar{h}$ = 500 W/m^2-K. The wall at x = 0 is adiabatic. The known information is entered into EES.

```
$UnitSystem SI MASS RAD PA  K J
$TabStops   0.2 0.4 0.6 0.8 3.5 in

"Inputs"
L=5 [cm]*convert(cm,m)                          "wall thickness"
k=5.0 [W/m-K]                                   "conductivity"
rho=2000 [kg/m^3]                               "density"
c=200 [J/kg-K]                                  "specific heat capacity"
T_ini=converttemp(C,K,20 [C])                   "initial temperature"
T_infinity=converttemp(C,K,200 [C])             "fluid temperature"
h_bar=500 [W/m^2-K]                             "heat transfer coefficient"
```

The partial differential equation that describes this problem is:

$$\alpha \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t} \tag{7-43}$$

where α is the thermal diffusivity of the material.

```
alpha=k/(rho*c)                                 "thermal diffusivity"
```

The boundary conditions are:

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = 0 \tag{7-44}$$

$$-k \left.\frac{\partial T}{\partial x}\right|_{x=L} = \bar{h}\left[T_{x=L} - T_\infty\right] \tag{7-45}$$

$$T_{t=0} = T_{ini} \tag{7-46}$$

In order to solve this problem, it is necessary to position nodes throughout the material as shown in Figure 7-22. The axial position of each node is given by:

$$x_i = \frac{(i-1)}{(N-1)} L \quad \text{for } i = 1..N \tag{7-47}$$

where N is the number of nodes used for the simulation. The distance between adjacent nodes (Δx) is:

$$\Delta x = \frac{L}{(N-1)} \tag{7-48}$$

This node spacing is specified in EES:

```
"Setup grid"
N=6 [-]                                          "number of nodes"
Duplicate i=1,N
    x[i]=(i-1)*L/(N-1)                           "position of each node"
End
DELTAx=L/(N-1)                                   "distance between adjacent nodes"
```

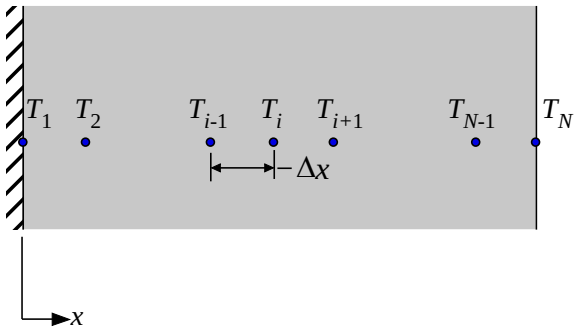


Figure 7-22: Nodes positioned within the computational domain.

Equation (7-43) is discretized and written for each internal node:

$$\frac{dT_i}{dt} = \alpha \frac{\left(\left.\frac{dT}{dx}\right|_{x=x_i+\Delta x/2} - \left.\frac{dT}{dx}\right|_{x=x_i-\Delta x/2} \right)}{\Delta x} \quad \text{for } i = 2..(N-1) \tag{7-49}$$

Substituting:

$$\left.\frac{dT}{dx}\right|_{x=x_i+\Delta x/2} = \frac{(T_{i+1} - T_i)}{\Delta x} \tag{7-50}$$

$$\left.\frac{dT}{dx}\right|_{x=x_i-\Delta x/2} = \frac{(T_i - T_{i-1})}{\Delta x} \tag{7-51}$$

into Eq. (7-49) leads to:

$$\frac{dT_i}{dt} = \alpha \frac{(T_{i+1} + T_{i-1} - 2T_i)}{\Delta x^2} \quad \text{for } i = 2..(N-1) \tag{7-52}$$

which is the state equation for each of the internal node temperatures.

Equation (7-43) is discretized and written for node 1:

$$\frac{dT_1}{dt} = \alpha \frac{\left(\left.\frac{dT}{dx}\right|_{x=\Delta x/2} - \left.\frac{dT}{dx}\right|_{x=0} \right)}{\frac{\Delta x}{2}} \tag{7-53}$$

Substituting Eq. (7-44) and:

$$\left.\frac{dT}{dx}\right|_{x=\Delta x/2} = \frac{\left(T_2 - T_1\right)}{\Delta x} \tag{7-54}$$

into Eq. (7-53) provides:

$$\frac{dT_1}{dt} = 2\alpha \frac{\left(T_2 - T_1\right)}{\Delta x^2} \tag{7-55}$$

which is the state equation for the temperature of node 1.

Equation (7-43) is discretized and written for node *N*:

$$\frac{dT_N}{dt} = \alpha \frac{\left(\left.\frac{dT}{dx}\right|_{x=L} - \left.\frac{dT}{dx}\right|_{x=L-\Delta x/2} \right)}{\frac{\Delta x}{2}} \tag{7-56}$$

Substituting Eq. (7-45) and:

$$\left.\frac{dT}{dx}\right|_{x=L-\Delta x/2} = \frac{\left(T_N - T_{N-1}\right)}{\Delta x} \tag{7-57}$$

into Eq. (7-56) provides:

$$\frac{dT_N}{dt} = 2\alpha \frac{\left(-\frac{\bar{h}}{k}\left(T_N - T_\infty\right) - \frac{\left(T_N - T_{N-1}\right)}{\Delta x} \right)}{\Delta x} \tag{7-58}$$

which is the state equation for the temperature of node *N*.

The first step in the solution process is to assign arbitrary values for the integration variable (t) and the state variables (T_1 through T_N) and calculate the time rate of change of each of the state variables using Eqs. (7-52), (7-55), and (7-58).

```
"Abitrary values of time and temperatures"
time=0 [s]
Duplicate i=1,N
    T[i]=T_ini
End

"Implement the state equations"
Duplicate i=2,(N-1)
    dTdt[i]=alpha*(T[i+1]+T[i-1]-2*T[i])/DELTAx^2
End
dTdt[1]=2*alpha*(T[2]-T[1])/DELTAx^2
dTdt[N]=2*alpha*(-h_bar*(T[N]-T_infinity)/k-(T[N]-T[N-1])/DELTAx)/DELTAx
```

Once the state equations have been implemented and checked it is possible to comment out the arbitrary values of t and T_1 through T_N

```
{"Abitrary values of time and temperatures"
time=0 [s]                                                    "time"
Duplicate i=1,N
    T[i]=T_ini
End}
```

and use the Integral command to integrate the set of simultaneous ODEs represented by Eqs. (7-52), (7-55), and (7-58) through time.

$$T_i = T_{ini} + \int_0^{t_{sim}} \frac{dT_i}{dt} \quad \text{for } i = 1..N \tag{7-59}$$

```
t_sim = 100 [s]                                               "duration of the simulation"
Duplicate i=1,N
    T[i]=T_ini+Integral(dTdt[i],time,0,t_sim)
End
```

Note that the problem considered here is identical to the bounded transient conduction problem that is included in the Transient Conduction menu of the Heat Transfer Library, discussed in Section 12.9 and therefore this solution can be directly compared to the analytical solution using the function planewall_T.

```
Duplicate i=1,N
    T_an[i]=planewall_T(x[i], time, T_ini, T_infinity, alpha, k, h_bar, L)
End
```

A $IntegralTable directive is used to store the trajectory of the integration process for each node.

```
$IntegralTable time:1,T[1..N]
```

The numerical and analytical solution for the temperature at each axial location as a function of time is shown in Figure 7-23.

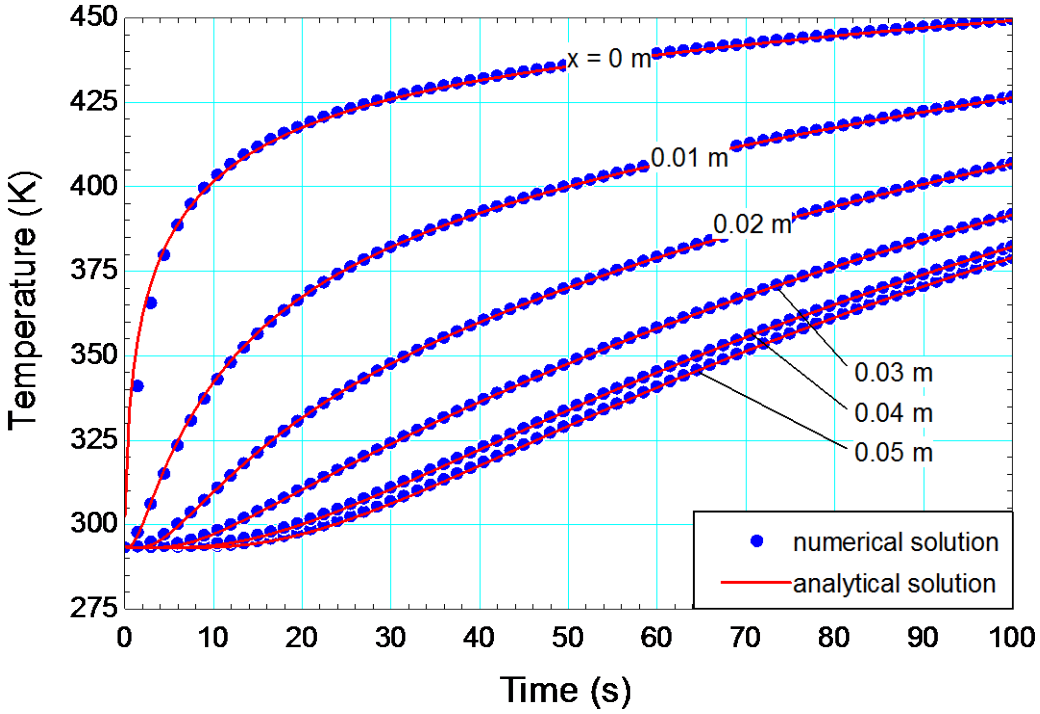


Figure 7-23: Numerical and analytical solution for the temperature at various axial locations as a function of time.

8 UNCERTAINTY PROPAGATION

Variables in engineering problems typically represent physical quantities. Therefore, EES allows variables to be assigned units, as discussed in Section 1.5. Variables in engineering problems may also be characterized by some uncertainty, particularly if they are the result of measurements. The results of calculations that involve these variables will, themselves, have some uncertainty. The process of keeping track of how the uncertainty of measured variables causes uncertainty in calculated variables is referred to as uncertainty propagation. The most common technique for propagation of uncertainty is referred to as the Root Sum Squares (RSS) method and it is reviewed in Section 8.1. EES allows the user to specify not only a value and a unit to each variable, but also an uncertainty. The propagation of this uncertainty through the calculations using the RSS method is automatically accomplished using EES' internal uncertainty propagation capability, described in this chapter.

8.1 Uncertainty Propagation using the RSS Method

The inputs to many engineering calculations are measured variables with some uncertainty. The uncertainty of experimental measurements can be estimated directly by taking a sample and obtaining the confidence interval. Alternatively, during the design stages of an experiment the uncertainty must be estimated from the characteristics of the instrument and the data acquisition system resolution. The value of any measurement (x) should be reported together with its uncertainty (u_x).

The Root Sum Square Method

It is rare that we directly measure the quantity that is actually of interest. More often, measurements are used in a series of calculations that lead finally to the desired quantity. For example, voltage and current may be measured in order to calculate power. The uncertainty of the basic measurements must be propagated through the calculations in order to obtain the resulting uncertainty in the calculated value. The Root Sum Square (RSS) technique for uncertainty propagation assumes that the measurements are normally distributed quantities that are uncorrelated and also that there are no systematic errors in the measurements. If the calculated quantity (Q) is a function of N measurements (x_1 through x_N) each with its own uncertainty (u_{x_1} through u_{x_N}):

$$Q = Q\left(x_1, x_2, ..., x_N\right) \tag{8-1}$$

then the uncertainty in Q (u_Q) can be divided into its elemental uncertainties due to each of the measured variables (u_{Q,x_1} through u_{Q,x_N}). The elemental uncertainty in Q due to measurement x_i is given by:

$$u_{Q,x_i} = \frac{\partial Q}{\partial x_i} u_{x_i} \tag{8-2}$$

The elemental uncertainties can be combined to provide the total uncertainty in Q by taking the square root of the sum of squares of the elemental uncertainties:

$$u_Q = \sqrt{u_{Q,x_1}^2 + u_{Q,x_2}^2 + ... + u_{Q,x_N}^2} \tag{8-3}$$

Example of the RSS Method

The RSS method will be illustrated using a simple example. The mass of argon in a tank is to be calculated by measuring the pressure, temperature, and internal volume in the tank. The temperature is T = 22.3°C and is measured with a thermocouple that has an estimated uncertainty of u_T = 1.2 K. The pressure is P = 934 kPa and is measured by a pressure transducer with uncertainty u_P = 22 kPa. The internal volume of the tank is V = 10 liter and the measurement of volume has an uncertainty of u_V = 0.4 liter. These inputs are entered in EES.

```
$UnitSystem SI Mass J K Pa

T=ConvertTemp(C,K,22.3 [C])                "temperature"
u_T=1.2 [K]                                "uncertainty in temperature"
P=934 [kPa]*convert(kPa,Pa)                "pressure"
u_P=22 [kPa]*convert(kPa,Pa)               "uncertainty in pressure"
Vol=10 [liter]*convert(liter,m^3)          "volume"
u_Vol=0.4 [liter]*convert(liter,m^3)       "uncertainty in volume"
```

The argon is assumed to behave according to the ideal gas law. The ideal gas constant is obtained according to:

$$R = \frac{R_{univ}}{MW} \tag{8-4}$$

where R_{univ} is the universal gas constant and MW is the molar mass of argon. We will assume that there is no uncertainty in the value of the ideal gas constant. The specific volume of argon is computed according to the ideal gas law:

$$v = \frac{RT}{P} \tag{8-5}$$

```
R=R#/MolarMass(Argon)                      "ideal gas constant"
v=R*T/P                                    "specific volume - ideal gas law"
```

The elemental uncertainty in the specific volume due to the uncertainty in temperature is given by:

$$u_{v,T} = \frac{\partial v}{\partial T} u_T = \frac{R}{P} u_T \tag{8-6}$$

and the elemental uncertainty in the specific volume due to the uncertainty in pressure is given by:

$$u_{v,P} = \frac{\partial v}{\partial P} u_T = -\frac{RT}{P^2} u_P \tag{8-7}$$

The total uncertainty in specific volume is obtained from:

$$u_v = \sqrt{u_{v,T}^2 + u_{v,P}^2} \tag{8-8}$$

```
u_v_T=R*u_T/P                    "elemental uncertainty in specific volume due to u_T"
u_v_P=-R*T*u_P/P^2               "elemental uncertainty in specific volume due to u_P"
u_v=sqrt(u_v_T^2+u_v_P^2)        "uncertainty in specific volume"
```

Solving provides v = 0.06584 m^3/kg with elemental uncertainties $u_{v,T}$ = 0.00027 m^3/kg and $u_{v,P}$ = -0.0016 m^3/kg. The total uncertainty in specific volume is u_v = 0.001574 m^3/kg.

The total mass of argon in the tank is:

$$m = \frac{V}{v} \tag{8-9}$$

The elemental uncertainty in the mass due to the uncertainty in the volume is:

$$u_{m,V} = \frac{\partial m}{\partial V} u_V = \frac{u_V}{v} \tag{8-10}$$

and the elemental uncertainty in the mass due to the uncertainty in the specific volume is:

$$u_{m,v} = \frac{\partial m}{\partial v} u_v = -\frac{V}{v^2} u_v \tag{8-11}$$

The total uncertainty in the mass is:

$$u_m = \sqrt{u_{m,V}^2 + u_{m,v}^2} \tag{8-12}$$

```
m=Vol/v                          "mass of argon"
u_m_Vol=u_Vol/v                  "elemental uncertainty in mass due to u_Vol"
u_m_v=-Vol*u_v/v^2               "elemental uncertainty in mass due to u_v"
u_m=sqrt(u_m_Vol^2+u_m_v^2)      "uncertainty in mass"
```

Solving provides m = 0.1519 kg with elemental uncertainties $u_{m,V}$ = 0.0061 kg and $u_{m,v}$ = -0.0036 kg. The total uncertainty in the mass is u_m = 0.007078 kg.

8.2 Uncertainty Propagation in EES

EES automates the process of uncertainty propagation using the RSS technique. Uncertainty values can be assigned to measured variables and these values of uncertainty are automatically used to determine the associated uncertainty in calculated variables. In order to illustrate this process, let's revisit the example carried out in Section 8.1. This time, only the measurements and their uncertainties are entered:

```
$UnitSystem SI Mass J K Pa

"Inputs"
T=ConvertTemp(C,K,22.3 [C])                "temperature"
u_T=1.2 [K]                                "uncertainty in temperature"
P=934 [kPa]*convert(kPa,Pa)                "pressure"
u_P=22 [kPa]*convert(kPa,Pa)               "uncertainty in pressure"
Vol=10 [liter]*convert(liter,m^3)          "volume"
u_Vol=0.4 [liter]*convert(liter,m^3)       "uncertainty in volume"
```

The calculations required to obtain the mass, Eqs. (8-4), (8-5), and (8-9) are entered:

```
R=R#/MolarMass(Argon)                      "ideal gas constant"
v=R*T/P                                    "specific volume - ideal gas law"
m=Vol/v                                    "mass of argon"
```

Solving provides m = 0.1519 kg but no indication of the uncertainty in this calculated mass.

Assigning Uncertainties to Measured Variables

Select Uncertainty Propagation from the Calculate menu in order to access the Determine Propagation of Uncertainty dialog, shown in Figure 8-1.

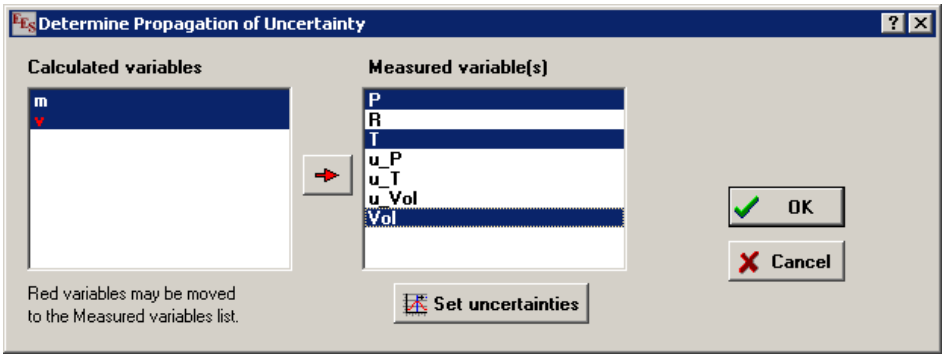


Figure 8-1: Determine Propagation of Uncertainty dialog.

The list of variables in the window on the right has been identified by EES as being measured variables that could have some uncertainty assigned to them. The variables that initially

populate this list are simply constants; that is, variables that are independent of all other variables. The variables in the window on the left are judged by EES to be calculated quantities. Note that the variables listed in red can be moved from the list of calculated variables to the list of measured variables. Select from the list of measured variables those that have some uncertainty (i.e., the variables P, T, and Vol in Figure 8-1). The maximum number of "measured" variables is 30 in the Commercial version of EES and 200 in the Professional version. Select Set uncertainties in order to access the Uncertainties of Measured Variables dialog, shown in Figure 8-2.

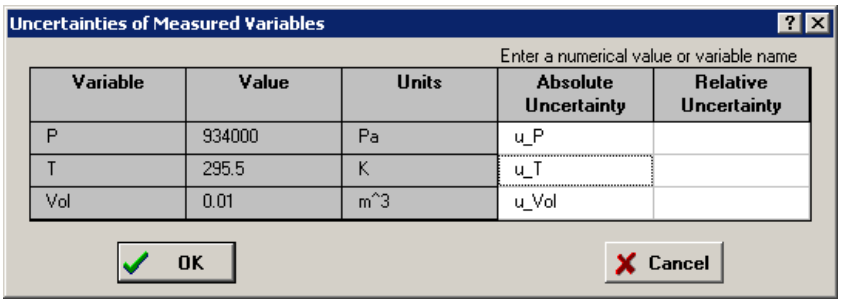


Figure 8-2: Uncertainties of Measured Variables dialog.

The uncertainties can be set on an absolute or relative basis using either numerical constants or variables. The units column provides the units for the measured variable and these are the units that must be used to set uncertainty on an absolute basis. In Figure 8-2, the uncertainties are set using the variables u_P, u_T, and u_Vol which have been assigned in the Equations Window. Alternatively, numerical values could have been entered.

Determining the Uncertainties of Calculated Variables

Once the uncertainties have been set, press the OK button to return to the Determine Propagation of Uncertainty dialog, shown in Figure 8-1. Select from the list of calculated variables in the window on the left those variables that will be considered in the uncertainty analysis. Up to 12 calculated variables can be selected in the Commercial version of EES and 50 in the Professional version. In Figure 8-1, the variables v and m have been selected. Select OK to initiate the calculations. EES will display a Solution Window that includes an additional tab labeled Uncertainty Results, as shown in Figure 8-3

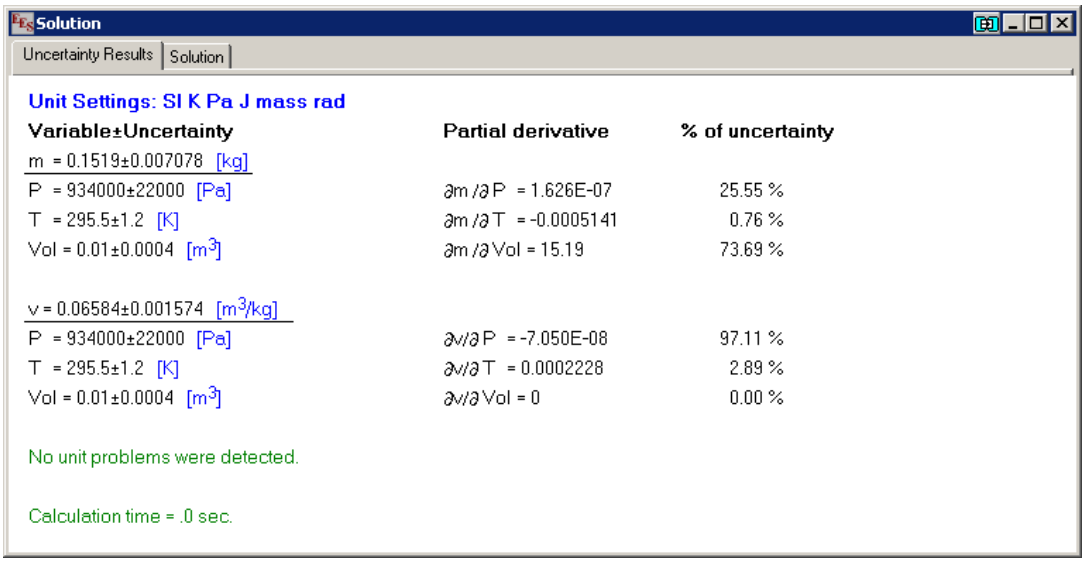


Figure 8-3: Uncertainty Results tab of the Solution Window.

Figure 8-3 shows that the specific volume is v = 0.0658 m^3/kg with a total uncertainty of u_v = 0.001574 m^3/kg and the mass is m = 0.1519 kg with a total uncertainty of u_m = 0.007078 kg. The source of the uncertainty in these values is also delineated. For example, Figure 8-3 shows that 74% of the total uncertainty in mass can be assigned to the uncertainty in the measurement of volume.

Internally, EES is carrying out the same procedure that we did by hand in Section 8.1 and therefore these results are identical. The difference is that EES calculates the required partial derivatives (e.g., the partial derivative of specific volume with respect to pressure) numerically rather than analytically. As a result, it is possible to carry out uncertainty analyses for large and complex systems of equations where analytical derivatives are not available. For example, rather than using the ideal gas law, we can utilize the equation of state for real fluid argon that is accessed with the volume function:

```
{v=R*T/P}                          "specific volume - ideal gas law"
v=volume(Argon, T=T, P=P)          "specific volume - equation of state"
```

Carrying out the uncertainty propagation calculations leads to the solution shown in Figure 8-4. The results are very similar to those for the ideal gas representation of argon in this case. However, this would not have been the case at higher pressures where deviations from ideal gas behavior occur. It would not have been possible to analytically determine the partial derivatives for real fluid argon.

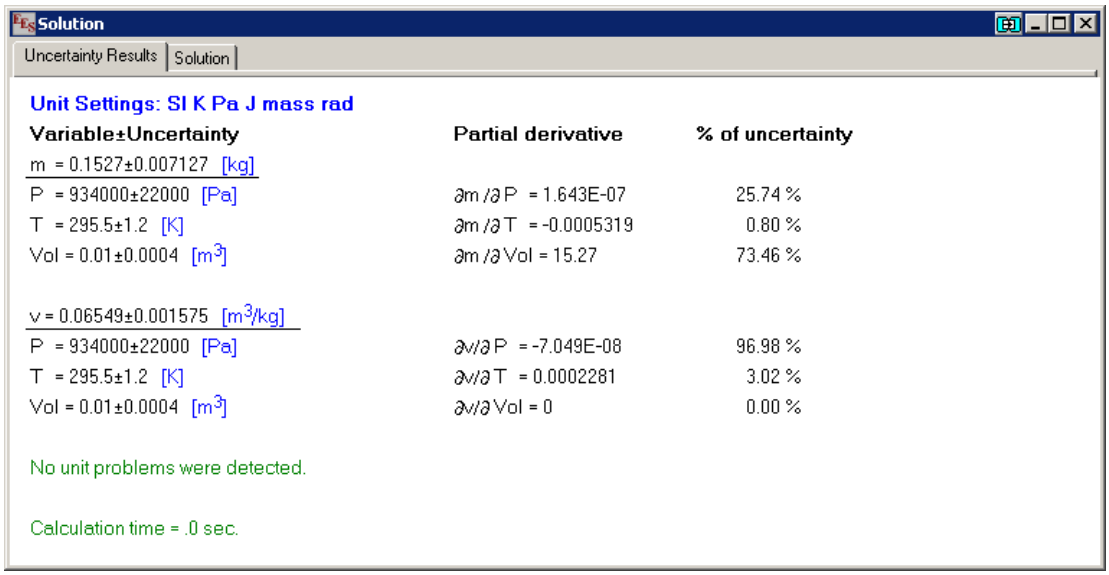


Figure 8-4: Uncertainty Results tab of the Solutions Window using the volume function.

UncertaintyOf Command

The UncertaintyOf function returns the assigned or calculated uncertainty of the variable that is provided as the argument of the function. For example, the command:

```
u_m=UncertaintyOf(m)                    "uncertainty of mass"
```

would provide u_m = 0.007078 kg when the uncertainty propagation calculations are run for the example from the last section. Note that the value of u_m will be zero if the calculations are not initiated using the Uncertainty Propagation command. The EES code:

```
percent_u_m=UncertaintyOf(m)/m*convert(-,%)  "percent uncertainty of mass"
```

will show that mass can be calculated to within 4.66%.

8.3 Uncertainty Propagation in Tables

Data are often contained in tables and therefore it is useful to be able to carry out uncertainty propagation calculations for a table of data. EES allows uncertainty propagation to be applied when the data are provided in either a Parametric Table or a Lookup Table.

Uncertainty Propagation Table

The uncertainty propagation calculations in EES can use data in the Parametric Table. Comment out the specified value of temperature in the example from Section 8.2:

```
{T=ConvertTemp(C,K,22.3 [C])            "temperature"}
```

and set up a Parametric Table that includes the variables T, m, and v. Vary the values of temperature using the Alter Values command, so that the Parametric Table appears as shown in Figure 8-5(a).

Table 1

1..10	1 T [K]	2 v [m³/kg]	3 m [kg]
Run 1	200		
Run 2	222.2		
Run 3	244.4		
Run 4	266.7		
Run 5	288.9		
Run 6	311.1		
Run 7	333.3		
Run 8	355.6		
Run 9	377.8		
Run 10	400		

(a)

Table 1

1..10	1 T [K]	2 v [m³/kg]	3 m [kg]
Run 1	200±1.2	0.04457±0.001083	0.2244±0.0105
Run 2	222.2±1.2	0.04952±0.001197	0.2019±0.009437
Run 3	244.4±1.2	0.05447±0.001311	0.1836±0.008569
Run 4	266.7±1.2	0.05942±0.001425	0.1683±0.007848
Run 5	288.9±1.2	0.06437±0.00154	0.1553±0.00724
Run 6	311.1±1.2	0.06933±0.001655	0.1442±0.006719
Run 7	333.3±1.2	0.07428±0.00177	0.1346±0.006268
Run 8	355.6±1.2	0.07923±0.001886	0.1262±0.005874
Run 9	377.8±1.2	0.08418±0.002001	0.1188±0.005527
Run 10	400±1.2	0.08913±0.002117	0.1122±0.005219

(b)

Figure 8-5: Parametric Table with (a) the values of T set and (b) the uncertainty propagation carried out.

Select Uncertainty Propagation Table from the Calculate menu in order to access the Determine Propagation of Uncertainty dialog, shown in Figure 8-6. Calculated variables can be selected from the window on the left; these include only those variables that are in the Parametric Table. Measured variables can be selected from the window on the right, which includes variables in the Parametric Table as well as those defined in the Equations window. The uncertainties of all measured variables must be specified.

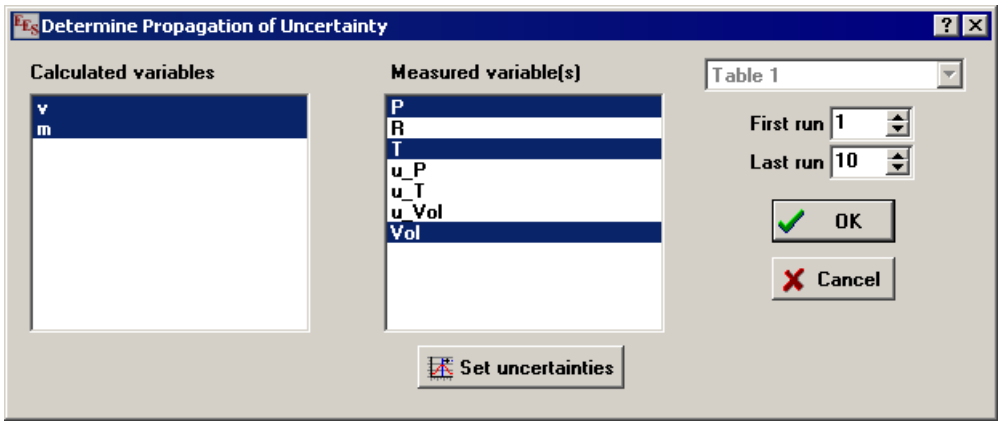


Figure 8-6: Determine Propagation of Uncertainty dialog.

Select OK to initiate the propagation of uncertainty calculations. The values of the calculated variables in the table will be assigned along with their uncertainties, as shown in Figure 8-5(b).

Plotting Data with Uncertainties

Because the results in the Parametric Table shown in Figure 8-5(b) are assigned values of uncertainty, any plot of these data will automatically include error bars. For example, select New Plot Window from the Plots menu and then select X-Y plot. Select the variable `T` as the X-axis and `m` as the Y-axis; the resulting plot automatically includes appropriate error bars, as shown in Figure 8-7.

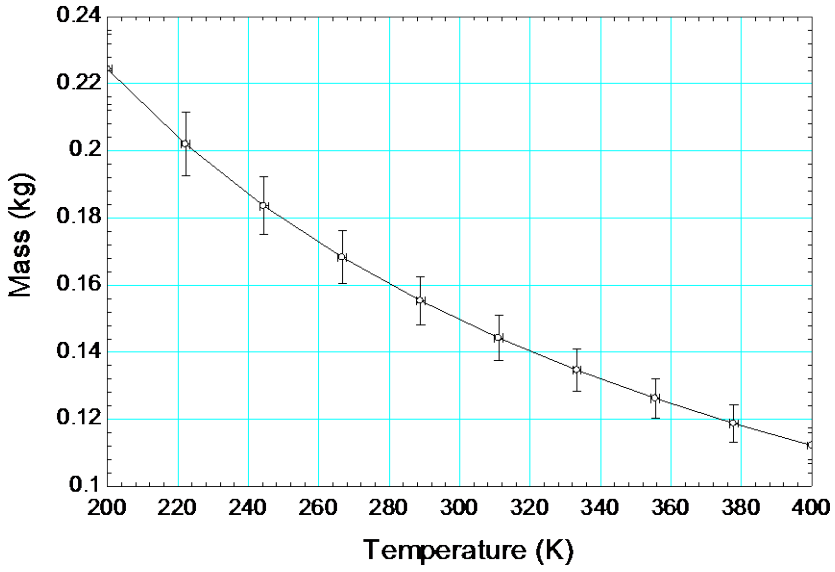


Figure 8-7: Mass as a function of temperature shown with error bars.

The error bars can be removed by double clicking on the graph and de-selecting the error bars.

Uncertainty Propagation with Lookup Tables

Consider the experiment shown in Figure 8-8 that is used to measure the heat transfer coefficient associated with a flowing fluid.

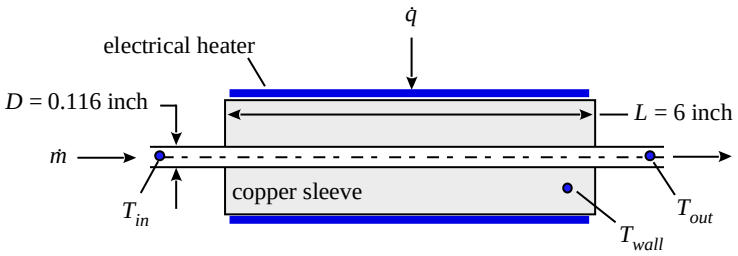


Figure 8-8: Heat transfer coefficient measurement experiment.

The mass flow rate ($\dot{m}$) through the tube is controlled and measured. The experimental uncertainty of the mass flow rate measurement is $u_{\dot{m}}$ = 0.005 kg/s. The diameter of the tube is D = 0.115 inch and its length is L = 6 inch; these characteristics have been measured with an uncertainty of u_D = 50 μm and u_L = 1 mm, respectively. The tube is surrounded by a copper sleeve so that its wall is nearly isothermal. A measured rate of heat input ($\dot{q}$) is applied to the test section electrically using a heater that is installed on the outer surface of the copper. The rate of heat input is measured with a precision of $u_{\dot{q}}$ = 1 W. The temperature of the wall (T_{wall}) is measured as is the bulk temperature of the fluid entering and leaving the tube (T_{in} and T_{out}). The temperatures are each measured with an uncertainty of u_T = 1 K. The measurements are carried out over a range of mass flow rates and the data are shown in Table 8-1.

Table 8-1: Data from heat transfer coefficient measurement experiment.

Mass flow rate (kg/s)	Inlet fluid temperature (ºC)	Wall temperature (ºC)	Outlet fluid temperature (ºC)	Rate of heat addition (W)
0.012	21.4	41.2	25.74	217.8
0.022	21.7	43.5	26.47	438.8
0.031	21.3	43.1	25.95	603.6
0.041	21.5	40.9	25.54	692.3
0.052	21.6	42.6	25.87	928.4
0.061	21.4	44.2	25.95	1161
0.071	21.3	42.9	25.54	1260
0.083	21.7	42.0	25.61	1359

The data in Table 8-1 are entered into a Lookup Table, as shown in Figure 8-9.

Lookup Table

Data

	1 $\dot{m}$ [kg/s]	2 $T_{in,C}$ [C]	3 $T_{wall,C}$ [C]	4 $T_{out,C}$ [C]	5 $\dot{q}$ [W]
Row 1	0.012	21.4	41.2	25.74	217.8
Row 2	0.022	21.7	43.5	26.47	438.8
Row 3	0.031	21.3	43.1	25.95	603.6
Row 4	0.041	21.5	40.9	25.54	692.3
Row 5	0.052	21.6	42.6	25.87	928.4
Row 6	0.061	21.4	44.2	25.95	1,161
Row 7	0.071	21.3	42.9	25.54	1,260
Row 8	0.083	21.7	42	25.61	1,359

Figure 8-9: Lookup Table containing the measurement data.

The diameter and length of the tube are entered in EES:

```
$UnitSystem SI Mass Rad J K Pa
D=0.115 [inch]*convert(inch,m)          "diameter"
L=6 [inch]*convert(inch,m)              "length"
```

Each run of the table can be processed in the Equations Window and the results placed in a Parametric Table using the TableRun# function in EES. The TableRun# function returns the current row of the Parametric Table that is being run when the Equations Window is activated using the Solve Table command. If the Equations Window is activated using the Solve command, then the TableRun# command returns zero. The mass flow rate, temperatures, and rate of heat transfer for the row being processed is obtained using the Lookup command and the TableRun# function:

```
m_dot=Lookup('Data',TableRun#,'m_dot')          "mass flow rate"
T_in=Lookup('Data',TableRun#,'T_in')            "inlet temperature, in C"
T_wall=Lookup('Data',TableRun#,'T_wall')        "wall temperature, in C"
T_out=Lookup('Data',TableRun#,'T_out')          "outlet temperature, in C"
q_dot=Lookup('Data',TableRun#,'q_dot')          "rate of heat transfer"
```

If the Solve command is used then the TableRun# will return zero which leads to the Lookup command returning the value from the first row in the table (because there is no row zero) and a warning is issued. The heat transfer coefficient ($\bar{h}$) is computed from the data using the formula:

$$\dot{q} = \bar{h}\,\pi\,D\,L\,\frac{\left(T_{out}-T_{in}\right)}{\ln\left[\dfrac{\left(T_{wall}-T_{in}\right)}{\left(T_{wall}-T_{out}\right)}\right]} \tag{8-13}$$

```
q_dot=h_bar*pi*D*L*(T_out-T_in)/ln((T_wall-T_in)/(T_wall-T_out))
```

A Parametric Table is created with 8 rows (one for each of the data points in Table 8-1 and corresponding row in the Lookup Table shown in Figure 8-9). The variables m_dot and h_bar are included in the Parametric Table, as shown in Figure 8-10(a).

Table 1

1..8	1 $\bar{h}$ [W/m2-K]	2 $\dot{m}$ [kg/s]
Run 1		
Run 2		
Run 3		
Run 4		
Run 5		
Run 6		
Run 7		
Run 8		

(a)

Table 1

1..8	1 $\bar{h}$ [W/m2-K]	2 $\dot{m}$ [kg/s]
Run 1	8879	0.012
Run 2	16243	0.022
Run 3	22268	0.031
Run 4	28612	0.041
Run 5	35341	0.052
Run 6	40614	0.061
Run 7	46434	0.071
Run 8	53173	0.083

(b)

Table 1

1..8	1 $\bar{h}$ [W/m2-K]	2 $\dot{m}$ [kg/s]
Run 1	8879±162.8	0.012±0.005
Run 2	16243±297.8	0.022±0.005
Run 3	22268±408.2	0.031±0.005
Run 4	28612±524.5	0.041±0.005
Run 5	35341±647.9	0.052±0.005
Run 6	40614±744.6	0.061±0.005
Run 7	46434±851.3	0.071±0.005
Run 8	53173±974.8	0.083±0.005

(c)

Figure 8-10: Parametric Table (a) before running, (b) after selecting Solve Table, and (c) after selecting Uncertainty Propagation Table.

Select Solve Table from the Calculate menu in order to populate the table with the values of heat transfer coefficient that are calculated for each data point, as shown in Figure 8-10(b). The uncertainty propagation calculations are carried out by selecting Uncertainty Propagation Table from the Calculate menu, which provides access to the Determine Propagation of Uncertainty dialog shown in Figure 8-11.

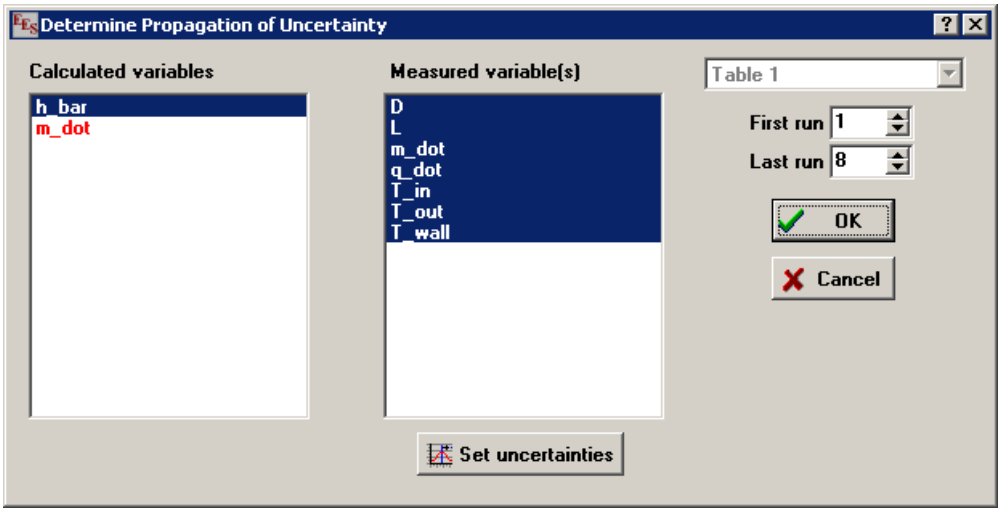


Figure 8-11: Determine Propagation of Uncertainty dialog.

Set the uncertainties in the measured variables, as shown in Figure 8-12.

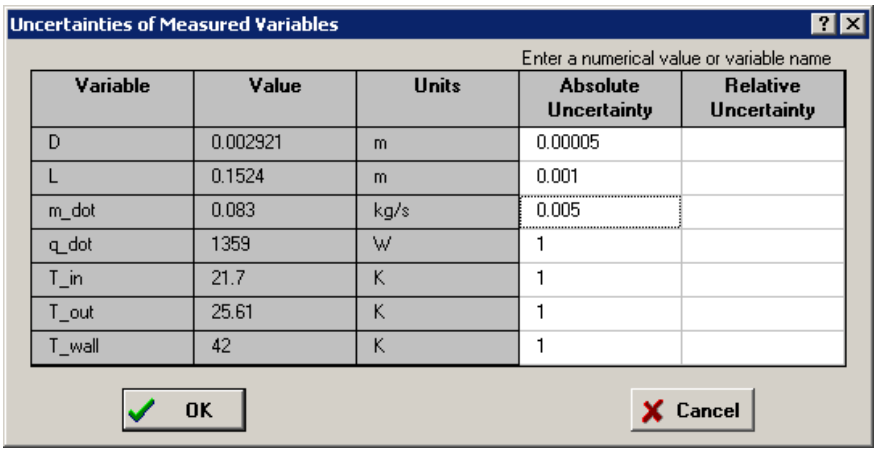


Figure 8-12: Uncertainties of Measured Variables dialog.

Select OK to set the uncertainties and then select OK again in order to run the Parametric Table. The results are shown in Figure 8-10(c); note that each calculated quantity now includes an uncertainty. Plotting the data shown in Figure 8-10(c) produces the plot with error bars that is shown in Figure 8-13.

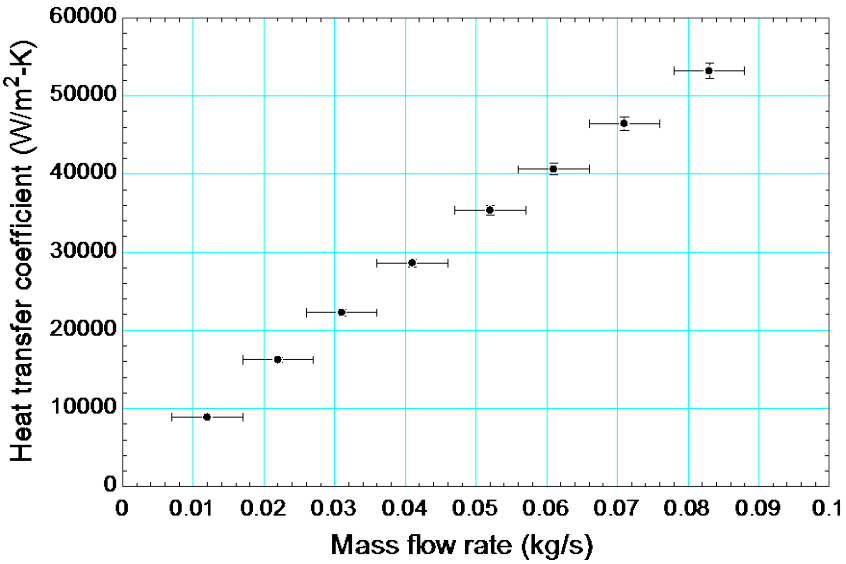


Figure 8-13: Calculated heat transfer coefficient as a function of mass flow rate with error bars indicated.

9 ADVANCED PLOTTING

The basic plotting features that are provided by EES were described in Section 1.4.. This chapter presents some additional types of plots that are available as well as some options for copying and documenting plots in EES.

9.1 Two-Dimensional Plots

The most common plot that engineers use is a two-dimensional plot. This type of plot displays one or more variables on the *y*-axis as a function of a single variable on the *x*-axis, as discussed in Section 1.4. A two-dimensional plot is created by selecting New Plot Window from the Plots menu and then selecting X-Y plot, as shown in Figure 9-1.

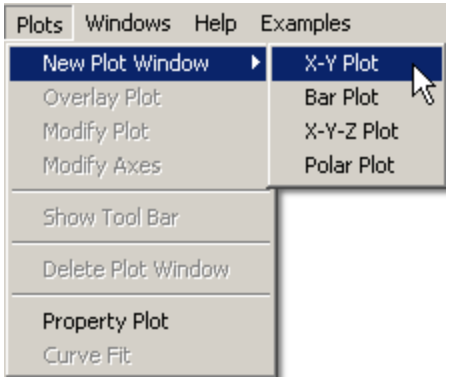


Figure 9-1: New Plot Window selection from the Plots menu.

Notice that there are several other plotting options available in EES, including Bar Plots, X-Y-Z (or 3-D) plots, and Polar Plots.

Bar Plots

A bar chart is a chart that has rectangular bars with lengths that are proportional to the values that they represent. This type of plot is ideal for illustrating binned data (e.g., histograms or other data where a number of counts have been assigned to various bins) or discontinuous or discrete data (e.g., color or college major). Consider Table 9-1, which lists the grades assigned in an undergraduate class; the average value of these scores is $\overline{x}$ = 84.3 and the standard deviation is σ = 7.2.

Table 9-1: Grades in an undergraduate class.

Letter Grade	Range	Number of students
A^+	97-100	7
A	93-97	14
A^-	90-93	26
B^+	87-90	33
B	83-87	42
B^-	80-83	36
C^+	77-80	30
C	73-77	17
C^-	70-73	6
D^+	67-70	2
D	63-67	0
D^-	60-63	1

These data are entered in a Lookup Table, shown in Figure 9-2(a). Note that the letter grades can be entered in the first column of the Lookup Table only after it has been formatted to be of type "string" by right-clicking on the column heading and selecting properties. Select string as the Style, as shown in Figure 9-3.

Lookup 1

	1 Grade	2 Lower end of range	3 Upper end of range	4 Number of students
Row 1	A+	97	100	7
Row 2	A	93	97	14
Row 3	A-	90	93	26
Row 4	B+	87	90	33
Row 5	B	83	87	42
Row 6	B-	80	83	36
Row 7	C+	77	80	30
Row 8	C	73	77	17
Row 9	C-	70	73	6
Row 10	D+	67	70	2
Row 11	D	63	67	0
Row 12	D-	60	63	1

(a)

Lookup 1

	1 Grade	2 Lower end of range	3 Upper end of range	4 Number of students	5 Center of range
Row 1	A+	97	100	7	98.5
Row 2	A	93	97	14	95
Row 3	A-	90	93	26	91.5
Row 4	B+	87	90	33	88.5
Row 5	B	83	87	42	85
Row 6	B-	80	83	36	81.5
Row 7	C+	77	80	30	78.5
Row 8	C	73	77	17	75
Row 9	C-	70	73	6	71.5
Row 10	D+	67	70	2	68.5
Row 11	D	63	67	0	65
Row 12	D-	60	63	1	61.5

(b)

Figure 9-2: Lookup Table (a) containing data from Table 9-1 and (b) with a column that includes the center of each range.

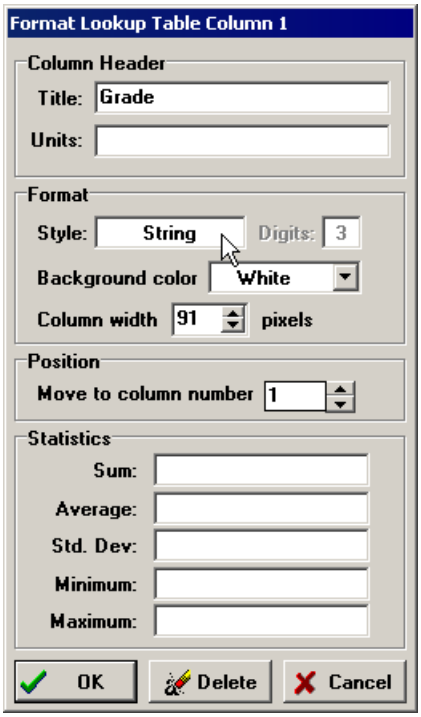


Figure 9-3: Format the Grade column of the Lookup Table as string.

In order to create a fifth column that contains the center of each range, right click on the fourth column and select Insert column to the right. Right-click on the new column and select Alter Values; use the Enter Equation option, as shown in Figure 9-4 to create the Lookup Table shown in Figure 9-2(b).

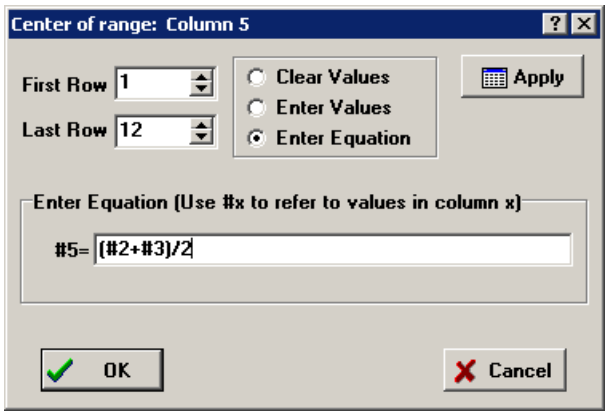


Figure 9-4: Enter Equation option to create column with center of each range.

To create a bar chart showing these results, select New Plot Window from the Plots menu and then select Bar Plot. Select the column Center of range as the X-Axis and Number of students as the Y-Axis to create the histogram shown in Figure 9-5.

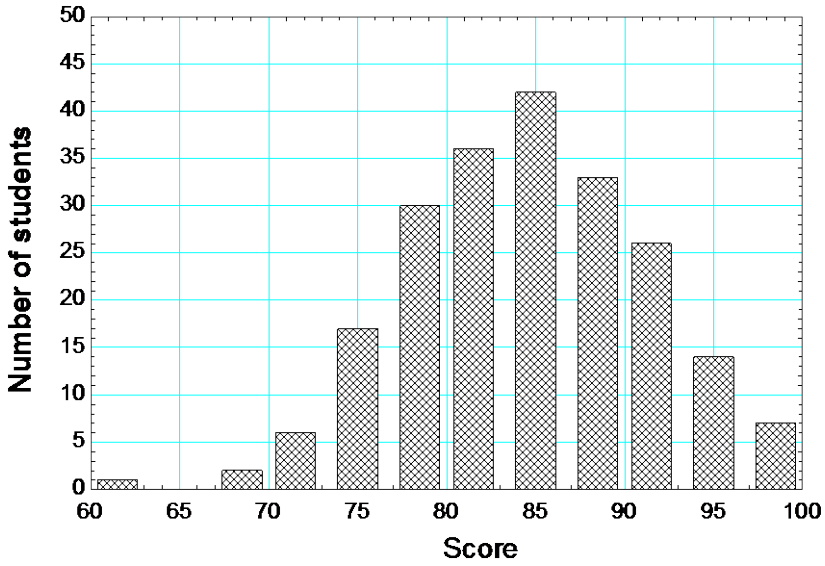


Figure 9-5: Histogram of students' grades shown with a bar plot.

Controlling the Appearance of Bar Plots

Figure 9-5 shows the bar plot with the default cross-hatched fill, color and bar thickness. These appearance options can be changed by clicking the right mouse button within the plot rectangle or by selecting the Modify Plot command from the Plots menu. Either action will display the Modify Plot dialog shown in Figure 9-6. The Line color box changes the color of the bars. The Symbol box provides a choice of six different fill types for the bars. The slider to the right of the Symbol box controls the width of the bars. When the slider is positioned to its rightmost position, as shown in Figure 9-6, there will be no space between the bars. The other controls shown in Figure 9-6 operate in the same manner as for X-Y plots.

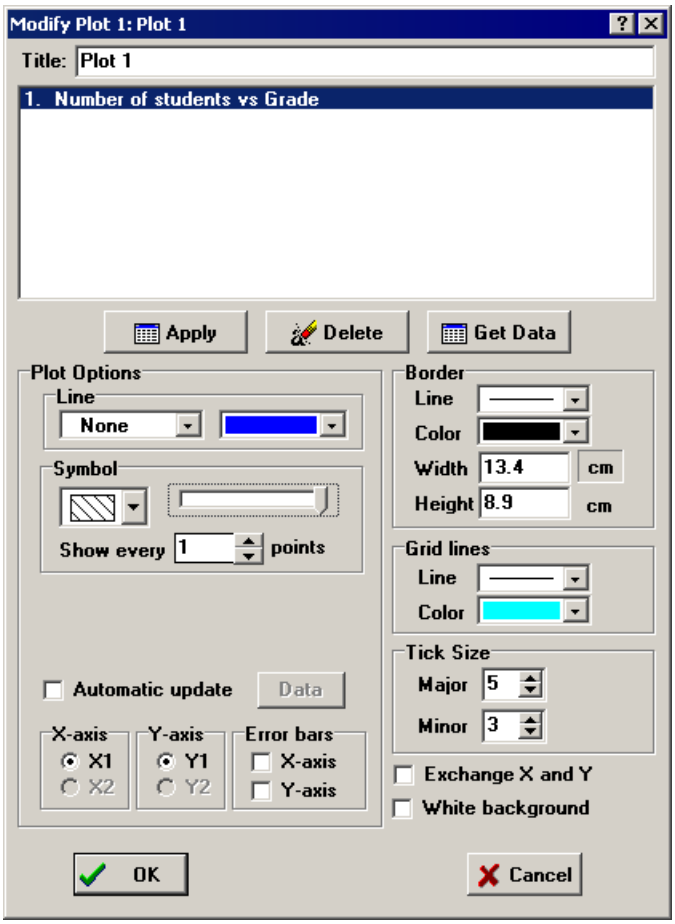


Figure 9-6: Modify Plot dialog for a bar plot.

Overlays on Bar Plots

It is possible to overlay additional bar plots or X-Y plots onto an existing bar plot. For example, we can overlay the normal probability distribution onto the histogram of students' grades in order to see if they are approximately normally distributed. The normal distribution is given by:

$$p = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-\bar{x})}{\sigma^2} \right] \tag{9-1}$$

This equation is entered in EES:

```
x_bar=84.3                          "mean"
sigma=7.2                           "standard deviation"
p=exp(-(x-x_bar)^2/sigma^2)/sqrt(2*pi)   "normal probability distribution"
```

A Parametric Table is created containing the variables x and p. The value of the variable x is varied from 60 to 100. The value of the variable p is calculated for each value of x using the Solve Table command. Select Overlay Plots from the Plots menu and then select the Parametric Table as the data source with the variable x as the X-Axis and p as the Y-Axis. Plot these results

on the right y-axis. If the symbol is chosen to be a cross-hatched bar (the default), the plot will appear as shown in Figure 9-7 with both bar plots visible.

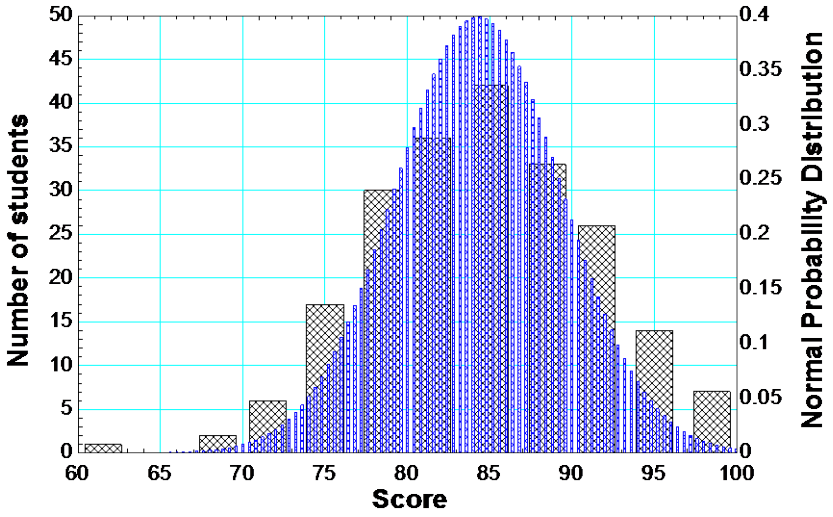
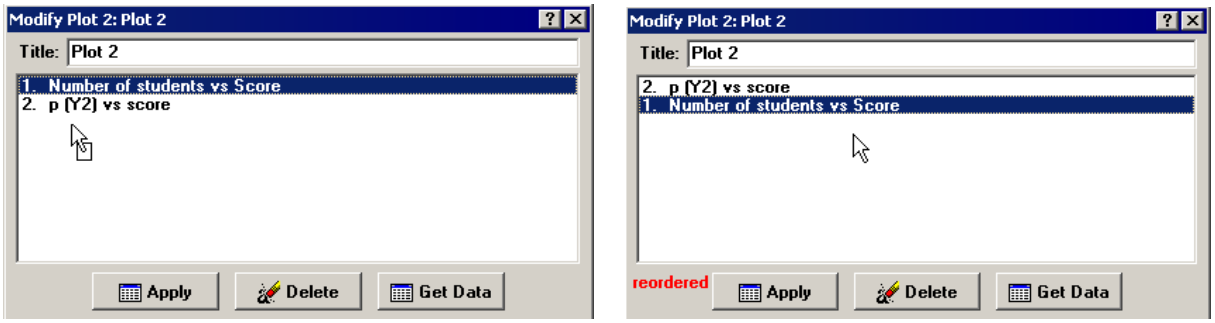


Figure 9-7: Superimposed bar plots.

The most recent plot appears in front of earlier plots. In some cases, such as this one, this order is not desirable. The order of bar or X-Y plots can be rearranged by selecting Modify Plot from the Plots menu or by clicking the right-mouse button when the mouse is positioned within the plot rectangle (but not on a text or graphic item). The Modify Plot dialog will display all of the plots. To change the order of the plots, press and hold the right mouse button down on the name of the plot you wish to move. The cursor will change to a drag cursor, as shown in (a) (b)

Figure 9-8(a). Move the cursor to the position that you wish the plot to appear at and release the mouse button. The position of the plot will change and the word reordered will be shown in red below the list, as seen in (a) (b)

Figure 9-8(b). Press the OK button and the plot will be drawn with the probability distribution bar plot in the background.



(a) (b)

Figure 9-8: Changing the order of plots in the Modify Plot dialog

In this case, however, the plot would look best if the normal probability distribution were shown with a line, rather than with bars. Right-click in the plot rectangle to bring up the Modify Plot dialog again. Click the symbol control to change the symbol to none. Change the line type to a solid line. The plot will appear as shown in Figure 9-9 after the OK button is pressed and the legend information is added with the Add text button on the plot tool bar.

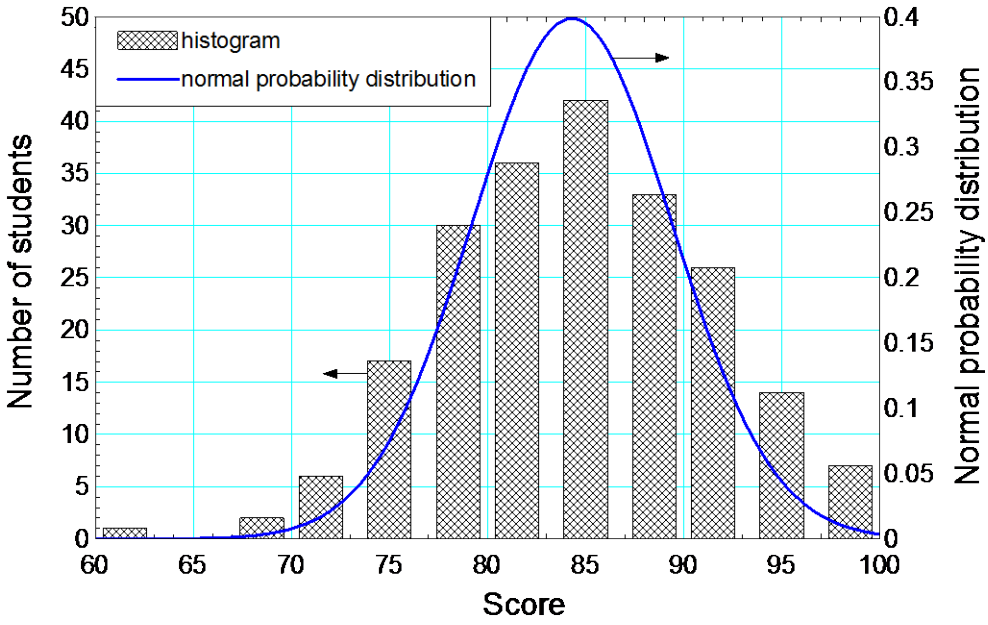


Figure 9-9: Normal probability distribution overlaid onto the histogram.

Bar Plots using String Values

String values can also be used as the X-labels for an X-Y or bar chart. This option is most appropriate for bar plots. For example, select New Plot Window from the Plots menu and then select Bar Plot. Select the Lookup Table shown in Figure 9-2 as the source of the data and then select the string column Grade as the X-Axis and Scores as the Y-Axis to obtain the histogram shown in Figure 9-10. The option exists to display strings vertically or at a 45° angle, which is convenient for longer strings.

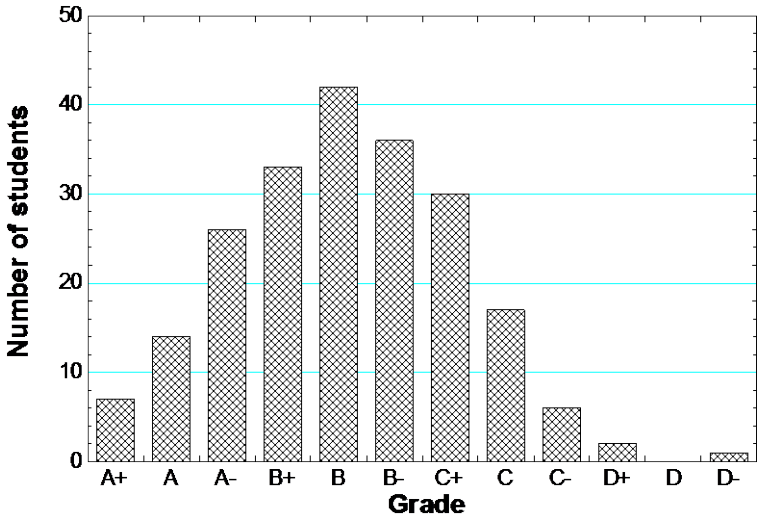


Figure 9-10: Histogram of students' grades with string values for x-axis.

Polar Plots

The Professional version of EES allows the generation of a polar plot. A polar plot has circumferential and radial axes and it is a convenient method for representing data in polar coordinates. For example, consider the equation for a polar rose:

$$r = a\cos(k\theta) \tag{9-2}$$

where a and k are constants, r is the radial location and θ is the angular position. Equation (9-2) is entered in EES.

```
$UnitSystem SI Mass J K Pa Rad

a=1
k=3
r=a*cos(k*theta)
```

A Parametric Table is generated and the variables theta and r are added to the table. The value of theta is varied from 0 to 2π and the table is solved. Select New Plot Window from the Plots menu and then select Polar Plot. Setup the polar plot with the variable theta as the circumferential axis and the variable r as the radial axis. Be sure to select radians as the angular units, as shown in Figure 9-11. The resulting polar plot is shown in Figure 9-12.

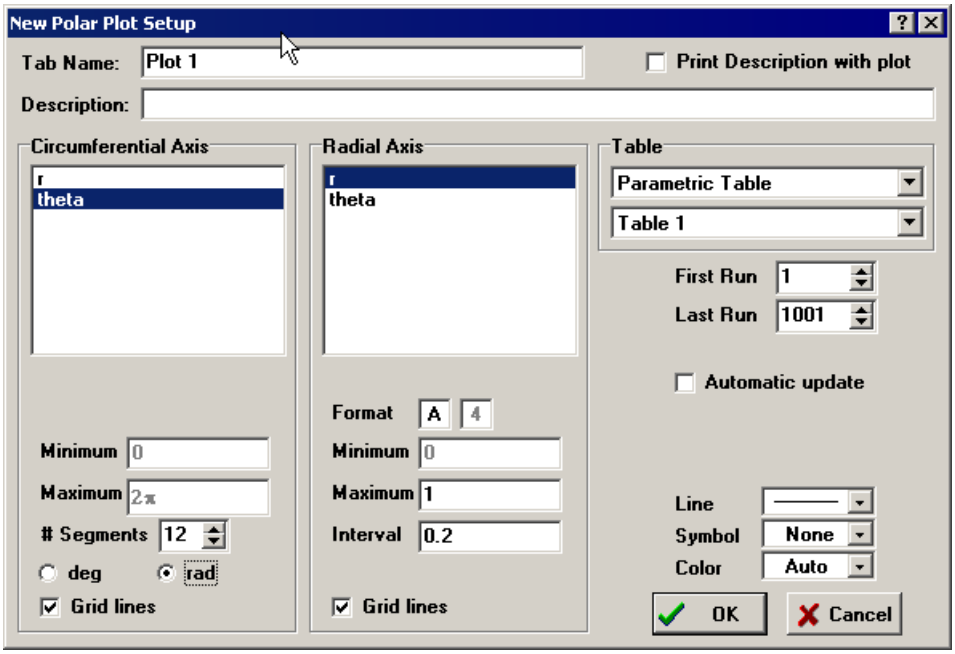


Figure 9-11: New Polar Plot Setup dialog.

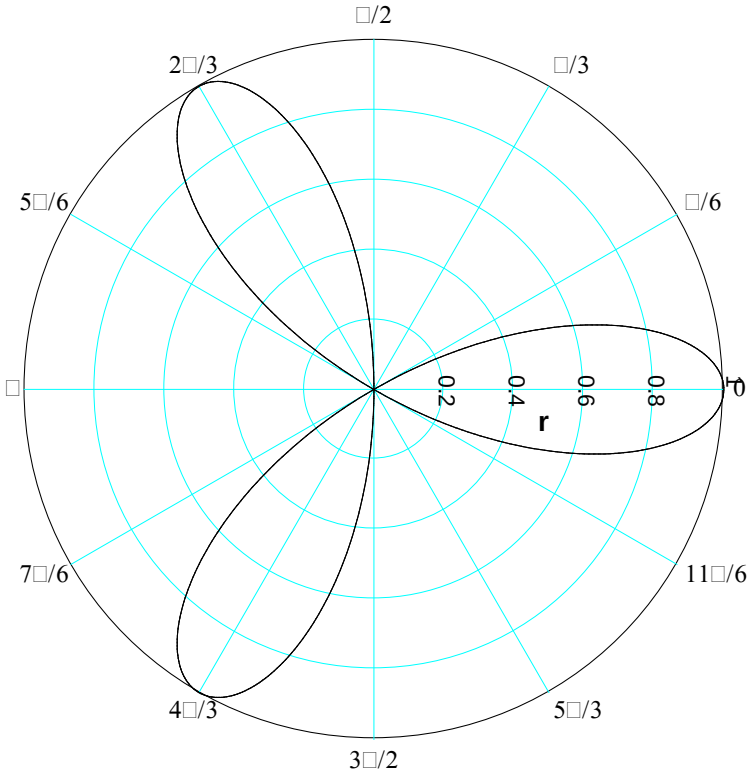


Figure 9-12: Polar plot.

Modifying the Appearance of Polar Plots

Figure 9-12 shows the default appearance of a polar plot. The circumferential axis is divided into 16 segments with the zero of the circumferential axis located at the right of the plot. There are five radial contours that are labeled along the line in which the circumferential axis is 0. The size of the text is based on the size selected in the Plot tab of the Preferences dialog (Options menu). All of the attributes of the polar plot can be modified. The plot color, line type, symbol type and other information can be changed in the usual manner, by clicking the right mouse button within the plot circle (but not on a text item) to bring up the Modify Plot dialog. The attributes of the circumferential and radial axes of the polar plot can be changed by clicking the right mouse button at a location just outside of the plot circle. This action will bring up the Modify Axes dialog. The radio buttons at the upper left control access to the Radial and Circumferential parameters. Both of these dialogs are shown in Figure 9-13.

The number of radial contours can be adjusted by specifying a value for the interval in Figure 9-13(a). The position of the contour labels is controlled with the Angle parameter. Angle is always entered in degrees and valid entries are from -90° to 90°. The radial contours will be labeled along a radius with a constant circumferential value equal to the value entered for the Angle.

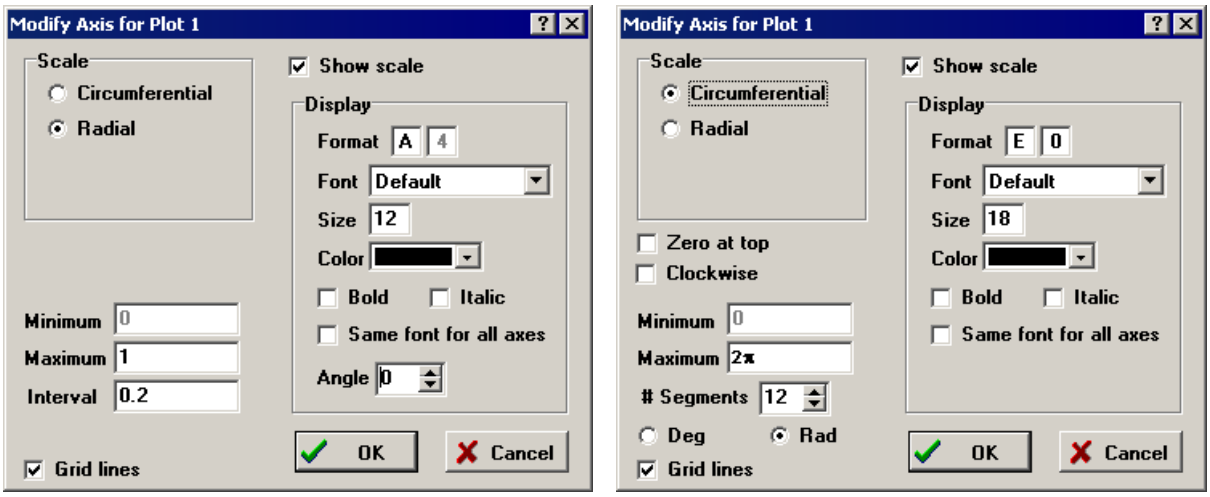


Figure 9-13: Modify Axis Dialog for a) Radial and b) Circumferential axes of Polar Plots.

Click the Circumferential button to access the attributes of the circumferential axis. By default, the circumferential direction increases in the counter-clockwise direction. This choice is not suitable for wind maps and other applications. It can be changed to clockwise by checking the Clockwise control. The location of zero for the circumferential axis can be changed from the right side of the plot to the top by checking the Zero at top control.

Using Variables as Axis Limits

In the Professional version of EES, the limits and interval that are used to create the plot axes can be set using variables rather than with numerical constants. This is particularly useful if you would like your plot to automatically update in response to changes in your program. For example, let's write a program that will allow us to graphically examine the 0^{th} order Bessel function of the 1^{st} kind over various ranges that are defined by user inputs x_{min} and x_{max}.

```
x_min=0                                   "minimum value of x"
x_max=6*pi                                "maximum value of x"
```

The interval used for the x-axis of the plot will be the range divided by seven and rounded to the nearest integer (using the Floor function in EES).

```
x_int=floor((x_max-x_min)/7)              "interval for plot"
```

The values to plot will be generated within an array that includes N = 100 values. The value of the argument (x) is varied from x_{min} to x_{max} in equal increments:

$$x_i = x_{min} + \frac{(i-1)}{(N-1)}(x_{max} - x_{min}) \quad \text{for } i = 1..N \tag{9-3}$$

and the value of the Bessel function is evaluated at each value of x using the EES function BesselJ.

```
N=100                                          "number of values"
duplicate i=1,N
    x[i]=x_min+(i-1)*(x_max-x_min)/(N-1)
    J0[i]=BesselJ(0,x[i])                      "0th order Bessel function of the 1st kind"
end
```

The values in the Arrays Table are used to prepare a plot of J0[i] versus x[i] . Double click on the plot and select Automatic Update so that the plot automatically changes if the data in the array are changed. Double click on the x-axis of the plot in order to access the Modify Axis dialog. Specify the minimum, maximum, and interval for the axis using the variables x_min, x_max, and x_int, as shown in Figure 9-14.

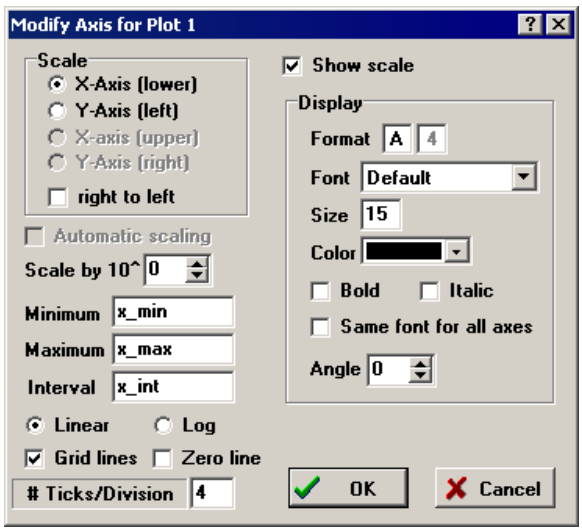


Figure 9-14: Modify Axis dialog showing the limits and interval set using variables.

The resulting plot is shown in Figure 9-15(a).

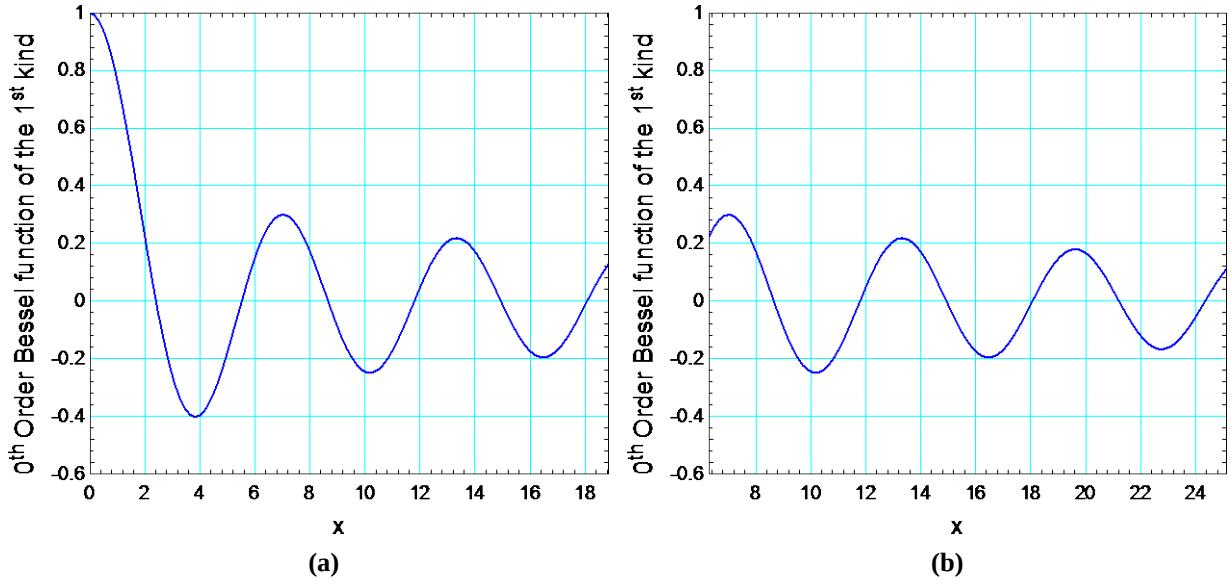


Figure 9-15: Plot for (a) x_{min} = 0 and x_{max} = 6π and (b) x_{min} = 2π and x_{max} = 8π.

Change the values of the limits in the Equations Window to $x_{min} = 2\pi$ and $x_{max} = 8\pi$:

```
x_min=2*pi                    "minimum value of x"
x_max=8*pi                    "maximum value of x"
```

and the plot should automatically change, as shown in Figure 9-15(b).

Copying and Saving Plots

It is possible to copy a plot from EES into another application in a manner that retains its high quality display. Open the plot window containing the plot of interest. Provided that no objects are selected on the plot, the Copy command in the Edit menu will appear as Copy Plot. Select Copy Plot (or Ctrl-C) in order to place a copy of the plot onto the clipboard so that it is available to be pasted into other applications.

The plot will be copied using the attributes that are specified in the Plot tab of the Preferences dialog. To view or changes these preferences, select Preferences from the Options menu and then select the Plot tab, shown in Figure 9-16, in order to adjust the format used to copy the plot to the clipboard. The attributes that affect how the plot is copied appear in the Clipboard Copy box at the bottom of the dialog. The plot can be copied in color or in black and white, depending on whether the Copy in color check box is selected. The plot can be copied as a picture (i.e., an enhanced metafile) and/or a high resolution bitmap. The resolution of the bitmap can be adjusted from 100 dpi (dots per inch) to 1200 dpi. The picture option, which provides a set of instructions to reproduce the plot, uses relatively little memory. A high resolution bitmap will produce a higher quality image that is a more exact replica of what is seen on screen, but it uses more memory. The differences are only visible when the plot is printed, as most screens do not provide sufficient resolution to see the differences.

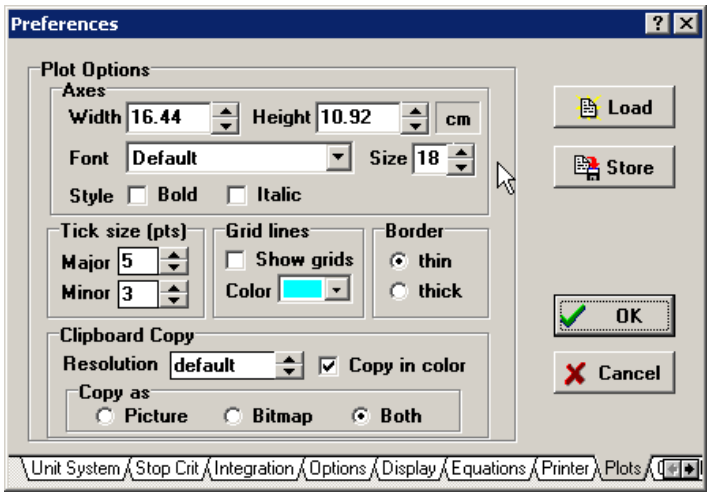


Figure 9-16: Plot tab of the Preferences dialog.

The plot can be saved in a file in a variety of formats and later imported into other applications. Click the right mouse button on the appropriate tab in order to access the Plot Information dialog shown in Figure 9-17.

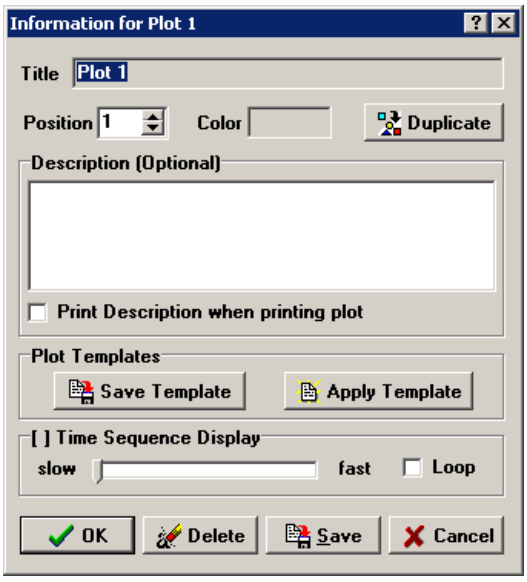


Figure 9-17: Information dialog.

Click the Save button in order to access the Save Plot dialog shown in Figure 9-18. This dialog is primarily dedicated to determining where on your computer the plot file will be stored. The plot can be saved as an enhanced metafile (.emf), bitmap (.bmp), jpeg (.jpg), tiff (.tif), or picture (.wmf) file, depending on the selection of the file type made just below the file name edit box. If the plot is saved as a bitmap, the resolution can be specified with the resolution drop-down control at the lower right of the dialog. Higher resolution requires more memory. A 300 dpi color bitmap is normally all that would be needed for excellent printing reproduction. The plot file will be written in black and white unless the Save Plot in Color control is checked.

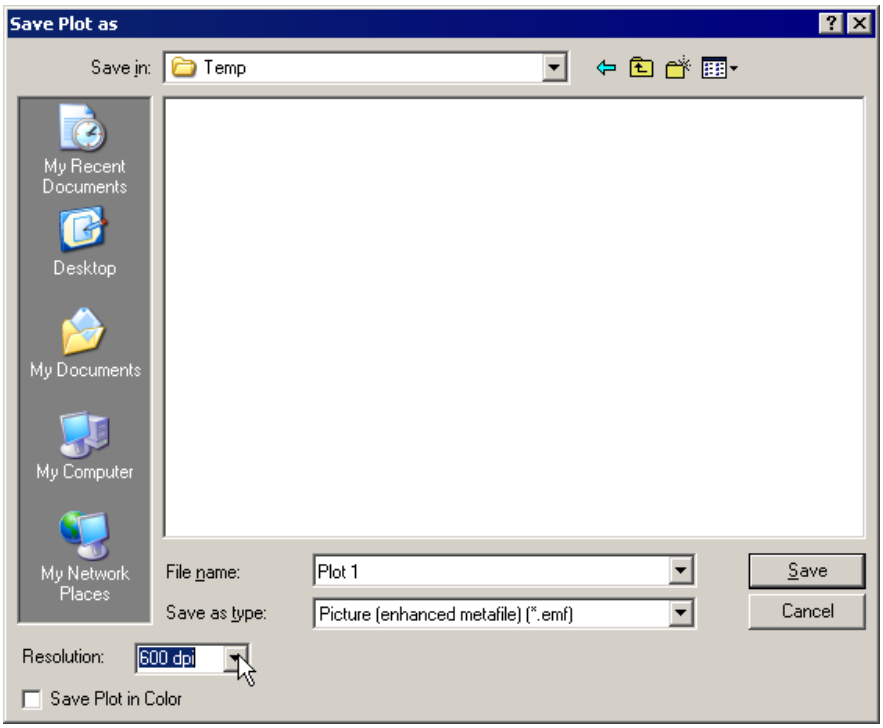


Figure 9-18: Save Plot dialog.

Naming and Documenting Plots

The Information dialog for a plot is accessed by clicking the right mouse button on the tab in the plot window. This action will display Plot Information dialog shown in Figure 9-17. The Information dialog allows the title of the plot that appears in the plot tab to be changed. The position of the plot tab with respect to other plots can be adjusted with the position control. The plot tabs are normally a gray color, matching the color of the window frame. However, you can change the color of the plot tab to make it stand out using the Color control below the plot title. In addition, you can add a description of the plot using up to 255 characters. If you position the mouse cursor over the tab for a plot, the descriptive text will be shown (or, if no descriptive text is entered, the variables that are plotted will be shown). An option is provided to print the description above the plot when the plot is printed. The Duplicate button creates a new plot window that is an exact copy of the current plot window. By default, the title for the duplicate plot is the same as the title of the existing plot window followed by (copy). The plot title can later be changed. Duplicate plots provide one way to ensure that the format of your plots is consistent from one plot to the next. Plot templates, described in the next section, provide an alternative way to maintain this formatting consistency.

Plot Templates

You may spend a substantial amount of time formatting a plot and making it look just right. In the Professional version of EES, your formatting preferences can be saved by selecting Save Template in the Information dialog shown in Figure 9-17. To illustrate this feature, save the formatting options used to generate a plot as the file template.ept (the file extension .ept stands for EES plot template). An .ept file is a text file that can be opened in a text editor such as Notepad and modified by the user. Now, duplicate a plot and adjust some of the formatting options; for example, change the font of the axis labels and/or text items, the size of the plot, or any other characteristic. Right-click on the tab for the duplicated plot and select Open Template in the Information dialog. Navigate to the file template.ept that you saved previously and select Open. Select OK and you should see that the plot format returns to its original style. This option is particularly useful if you are writing a report in which every plot should have a uniform format.

9.2 Time Sequence Plots

The Professional version of EES allows individual data series on a plot to be displayed sequentially in order to create an animation of the information. This capability is particularly useful if you are trying to display spatial information at different times.

As an example, consider the temperature distribution in a semi-infinite body that has been subjected to a pulse of energy at its surface at time zero. The solution to this problem is available using the function SemiInf4 in the Heat Transfer application library that is included in EES, as discussed in Section 12.9. The function Semiinf4 can be accessed by selecting Function Info from the Options menu and then selecting Transient Conduction from the drop down menu

at the lower right corner of the upper part of the window, as shown in Figure 9-19. Select Semi-Infinite Body from the drop down menu in the lower part of the window and then scroll to the correct solution, SemiInf4.

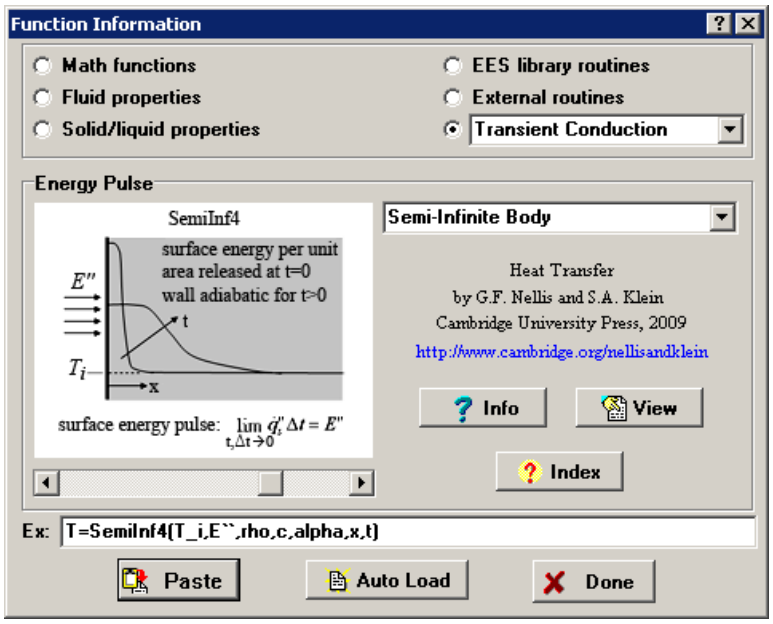


Figure 9-19: Access the function SemiInf4 in EES.

The calling protocol for the function SemiInf4 can be examined by selecting the Info button:

```
T=SemiInf4(T_i,E``,rho,c,alpha,x,t)
```

where T_i is the initial temperature, E`` is the surface energy per area, rho, c, and alpha are the density, specific heat capacity, and thermal diffusivity of the material, respectively, x is the position (relative to the surface), and t is the time (relative to the application of the pulse of energy).

We will animate the solution to this problem for the case where the initial temperature is T_i = 300 K, the surface heat pulse is E'' = 1x10^6 J/m^2, the material properties are ρ = 8933 kg/m^3, c = 385 J/kg-K, and α = 1.166x10^{-4} m^2/s.

```
$UnitSystem SI Mass J K Pa Rad
$TabStops 0.2 0.4 3.5

rho=8933 [kg/m^3]                    "density"
c=385 [J/kg-K]                       "specific heat capacity"
alpha=1.166e-4 [m^2/s]               "thermal diffusivity"
E``=1e6 [J/m^2]                      "pulse of energy per area"
T_ini=300 [K]                        "initial temperature"
```

Creating the Initial Plot

In order to create a time-sequence plot, we first need to make a plot that includes all of the data series that will eventually be shown one at a time. Here we will generate a matrix that includes the temperature at N = 41 axial locations ranging from 0 to 0.25 m for M = 20 different times ranging from near zero (i.e., immediately after the pulse is applied) to 60 s.

The array of x locations is generated:

```
x_sim=0.25 [m]                          "simulation extent"
n=41 [-]                                "number of spatial positions to simulate"
duplicate i=1,n
    x[i]=(i-1)*x_sim/(n-1)              "position"
end
```

and the array of times is generated:

```
time_sim=60 [s]                         "simulation time"
m=20 [-]                                "number of times to simulate"
duplicate j=1,m
    time[j]=j*time_sim/m                "time"
end
```

The temperature at each of these values of position and time is calculated using nested duplicate loops in order to create the matrix of temperatures.

```
duplicate j=1,m
    duplicate i=1,n
        T[i,j]=SemiInf4(T_ini,E``,rho,c,alpha,x[i],time[j])
    end
end
```

Solving leads to the Arrays Table shown in Figure 9-20.

Main

Sort	1 x_i [m]	2 $T_{i,1}$ [K]	3 $T_{i,2}$ [K]	4 $T_{i,3}$ [K]	5 T_i [K
[1]	0	308.8	306.2	305.1	
[2]	0.00625	308.5	306.1	305	
[3]	0.0125	307.8	305.9	304.9	
[4]	0.01875	306.8	305.5	304.7	
[5]	0.025	305.6	305	304.4	
[6]	0.03125	304.4	304.4	304	
[7]	0.0375	303.2	303.8	303.6	
[8]	0.04375	302.2	303.1	303.2	
[9]	0.05	301.5	302.5	302.8	
[10]	0.05625	300.9	302	302.4	
[11]	0.0625	300.5	301.5	302	

Figure 9-20: Arrays Table containing the data to be plotted.

Plot each of the temperature arrays as a function of the position array in order to generate the plot shown in Figure 9-21. All of the temperature arrays can be plotted at one time by selecting x[i] from the X-axis list and all of the temperature arrays from the Y-axis list. You will need to click on each temperature array to select it.

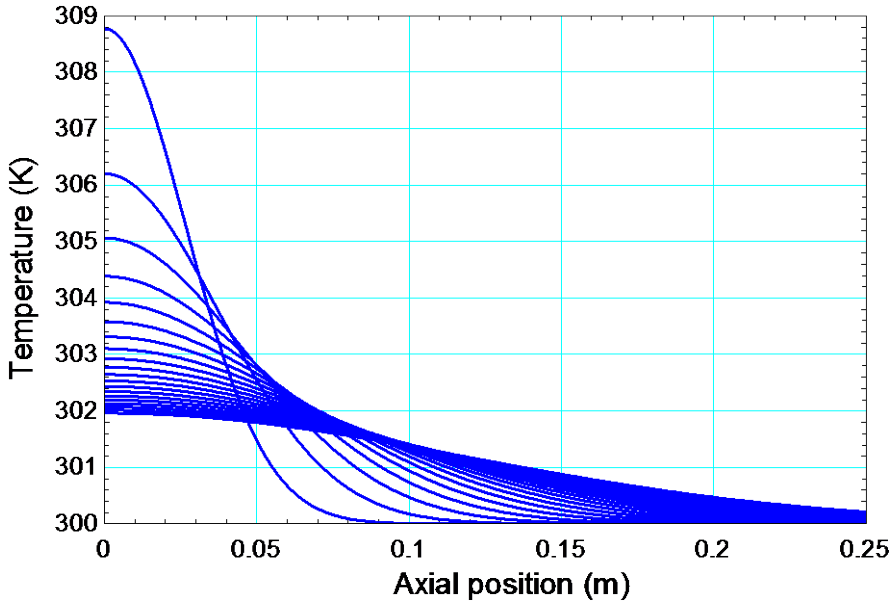


Figure 9-21: Plot showing all of the data.

Activating the Time Sequence Display

In order to activate the time sequence display, right click on the tab for the plot in order to bring up the Information dialog shown in Figure 9-22.

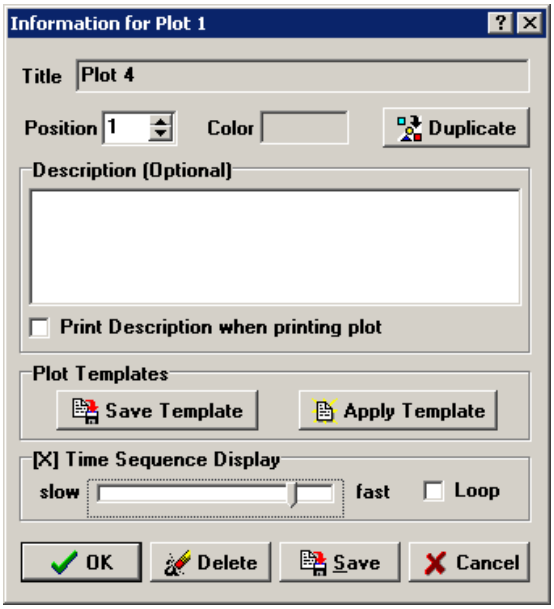


Figure 9-22: Information dialog showing the time sequence display.

The Time Sequence Display option is available at the lower part of the dialog. Select Time Sequence display and adjust the speed of the animation. Select Loop in order to replay the animation each time it completes. Select OK and a control panel will appear on the upper left corner of the plot that controls the time sequence display, as shown in Figure 9-23.

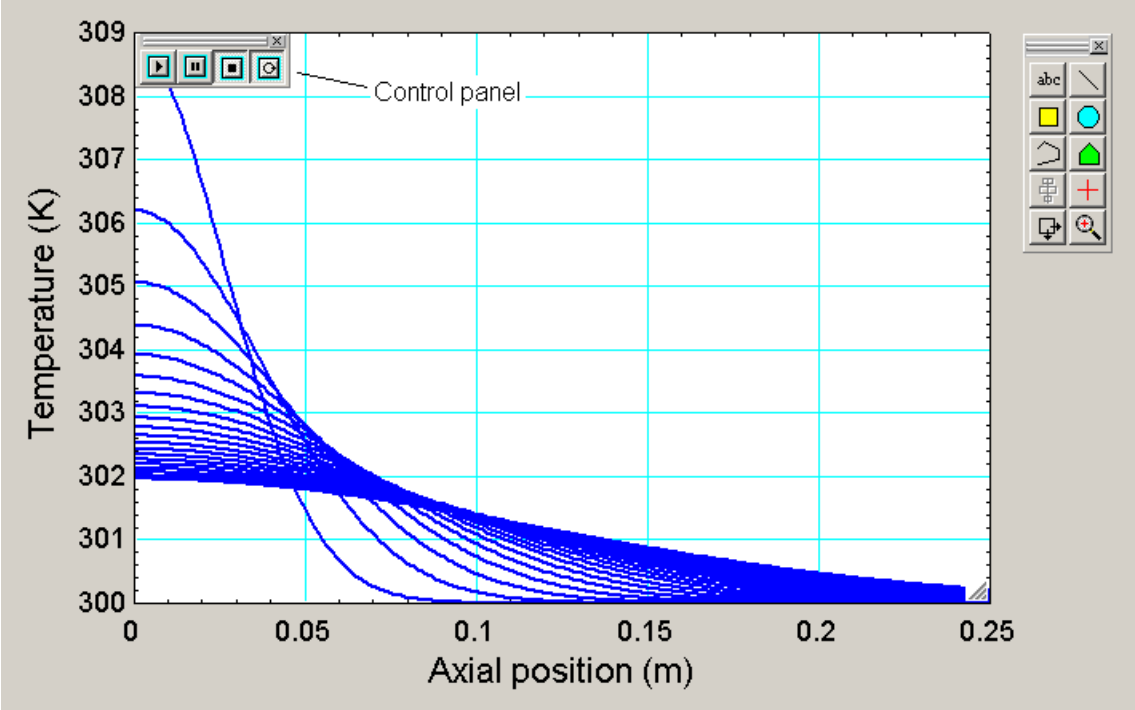


Figure 9-23: Plot showing the control panel for the time sequence display.

Each data series in the plot is assigned to a frame. Pressing the play button ( ) will activate the time sequence display. The pause button ( ) stops the display at the current frame and the stop button ( ) stops the display and resets it. The loop button ( ) causes the display to loop after it completes.

Assigning Frame Numbers to Data Series

Double click on the plot in order to access the Modify Plot dialog, shown in Figure 9-24.

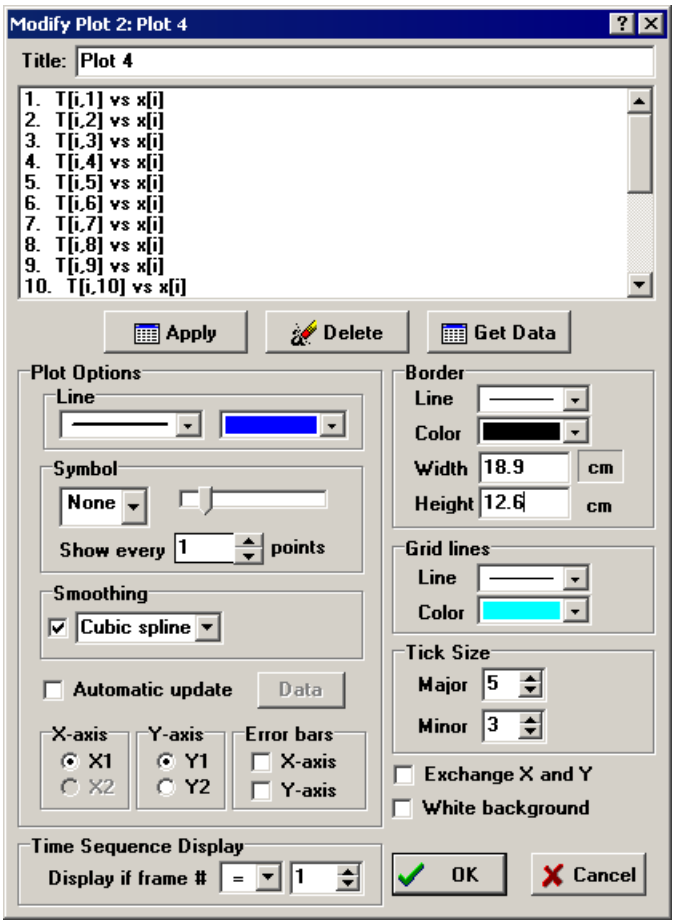


Figure 9-24: Modify Plot dialog showing the time sequence display frames.

At the bottom of the dialog is an area labeled Time Sequence Display. Each plot overlay is displayed in a separate frame. The plots can be displayed one at a time and then subsequently removed if "Display if frame #" is set to = followed by the appropriate frame number, as shown in Figure 9-24. Alternatively, the plots can be displayed one at a time and then persist if the Display if frame # is set to >=. The frame numbers are set by default to the order in which the plot overlays occur.

Assigning Frame Numbers to Objects

Text and graphic objects in the plot window can also be assigned frame numbers so that they appear and/or disappear during the time sequence display. For example, add a text object to the plot and modify the text to read “time > 30 s”. We would like this text item to only appear after the 10^{th} frame (corresponding to a time of 30 s) has been displayed. Therefore, modify the Time Series Display area at the bottom of the plot to read Display if frame # >= 11, as shown in Figure 9-25.

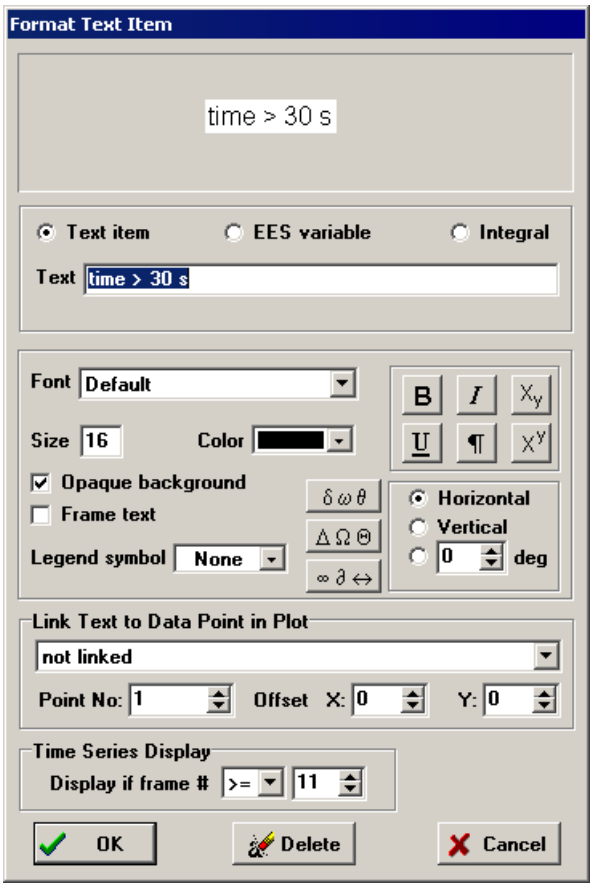


Figure 9-25: Format Text Item dialog showing the time series display.

Activate the time sequence display by selecting the play button on the control panel shown in Figure 9-23 and you should see the text item appear during the last half of the animation. By default, all plot objects and text items are set to display if frame >= 0 so that they persist during the entire animation.

Making a Movie

In order to make a movie that can be played from other software (e.g., Windows Media Player), it is necessary to save each frame of the plot as a separate image and then use some software (e.g., Windows Movie Maker) to assemble the images. Right click on the plot tab. When the Time Sequence display is enabled, the Save button in this dialog becomes the Movie button, as shown in Figure 9-26.

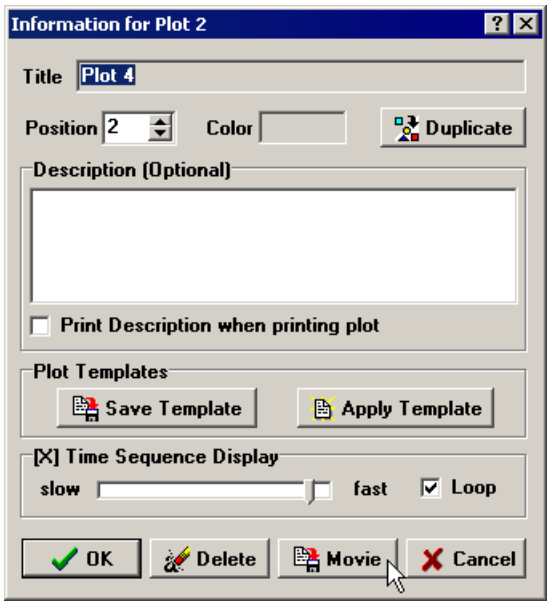


Figure 9-26: Save each image of the time sequence by selecting the Movie button.

Click the Movie button and the Save Each Frame in a Separate File for a Movie dialog will appear, as seen in Figure 9-27. It is best to create a new folder to hold all of the movie files. Here the folder name is Movie. Enter a file name (e.g., frame) in the File Name box. In this case, the file name extension is set to .emf, but it could be set .jpg, .tif, or .bmp. Click the Save button and a separate .emf file will be generated for each frame; each file will be given the name specified in the File Name box followed by _# where # is the sequential frame number.

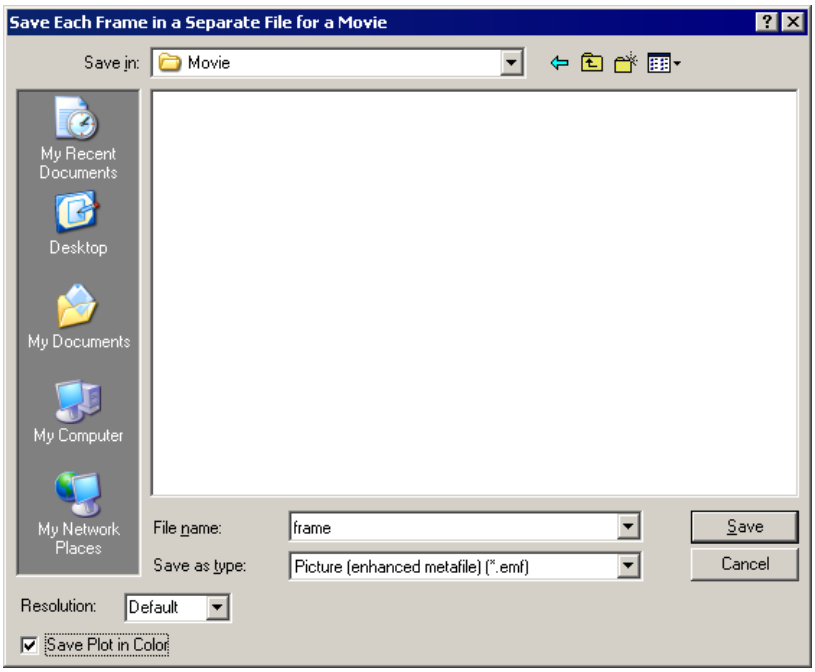


Figure 9-27: Save Each Frame in a Separate File for a Movie dialog.

Each type of movie-making software has its own protocol for assembling frames into a movie. In Windows Movie Maker, select Import pictures from the Capture Video heading and then select each frame that was saved. Choose Options from the Tools menu and then select the Advanced tab. Change the duration and transition times according to how fast you would like the video to play, as shown in Figure 9-28.

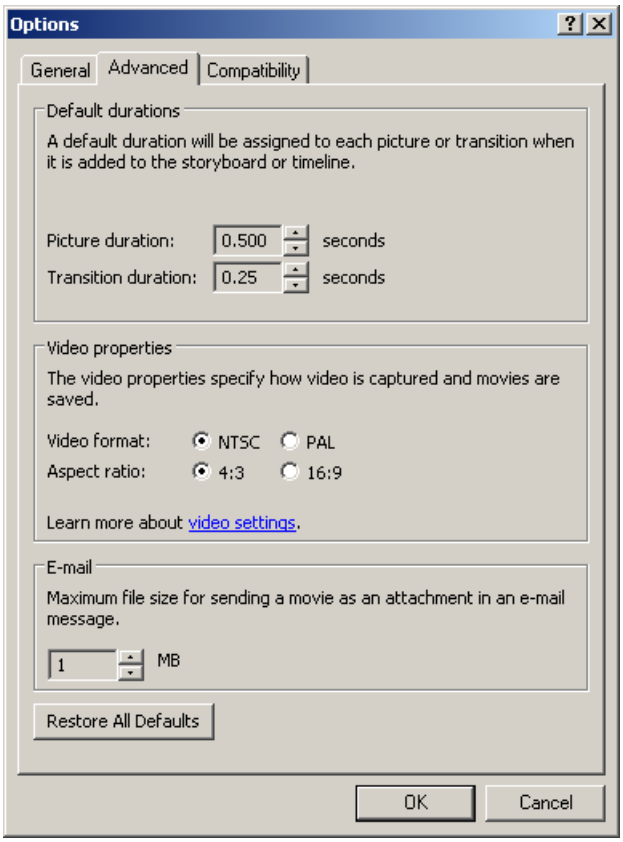


Figure 9-28: Options dialog in Windows Movie Maker.

Select all of the frames and drag them to the storyboard, as shown in Figure 9-29. Select the play button above the storyboard to preview the movie. Finally, select Finish Movie and Save to my computer in order to save the movie in a format that is playable by most software (e.g., Windows Media Player).

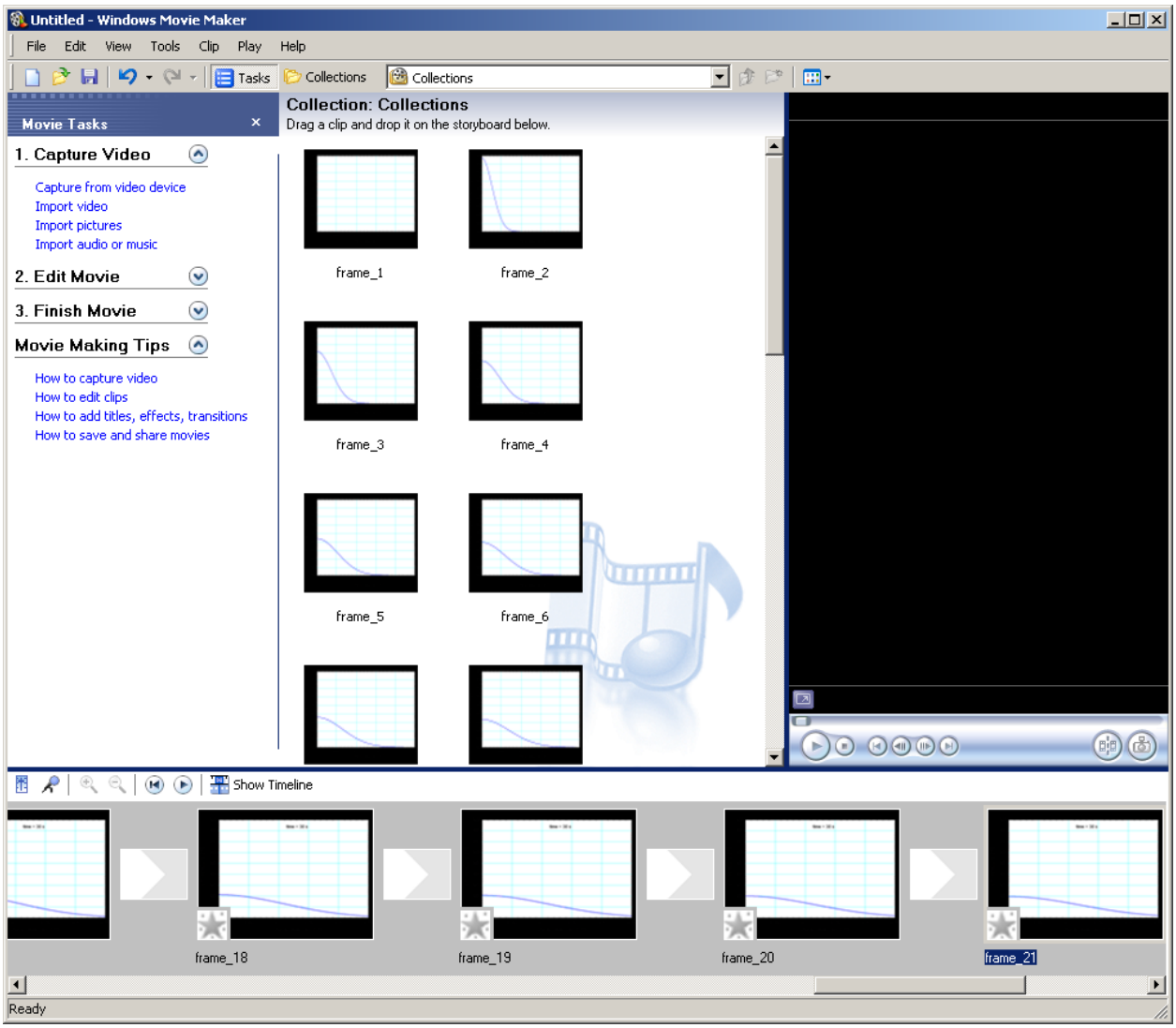


Figure 9-29: Frames dragged to the storyboard in Windows Movie Maker.

9.3 Three-Dimensional Plots

EES provides the capability to plot a dependent variable as a function of two independent variables in what are referred to as three-dimensional (or X-Y-Z) plots. EES can generate several types of 3-D plots including contour plots, gradient plots, 3-D surface plots and 3-D point points. These plots are discussed in this section in the context of the simple conduction problem shown in Figure 9-30.

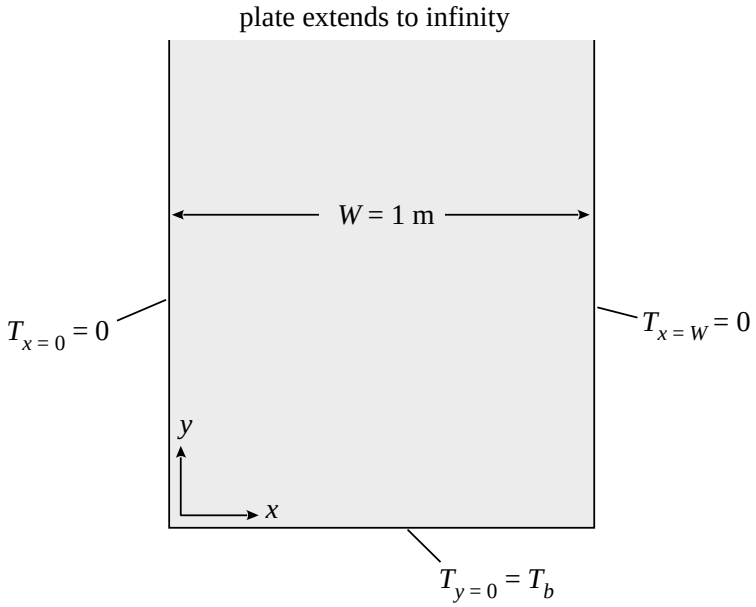


Figure 9-30: Plate subjected to fixed base and edge temperatures.

The temperature in a plate varies only in the x- and y-directions (i.e., the top and bottom of the plate are insulated). The plate is W = 1 m wide and extends infinitely into the y direction. The base of the plate (at y = 0) is subjected to a fixed temperature, T_b = 1 K. The edges of the plate (at x = 0 and x = W) are subjected to fixed temperature 0 K. The solution to this problem can be obtained analytically using separation of variables as discussed in Nellis and Klein (2009):

$$T = \sum_{i=1}^{\infty} \frac{-2T_b\left[-1+\cos\left(i\pi\right)\right]}{\left(i\pi\right)} \sin\left(\frac{i\pi x}{W}\right) \exp\left(-\frac{i\pi y}{W}\right) \tag{9-4}$$

The first N = 100 terms of the solution are programmed in EES and used to compute the temperature at x = 0.1 m and y = 0.1 m.

```
$UnitSystem SI Mass Rad Pa K J
W=1 [m]                                          "width of plate"
T_b=1 [K]                                        "base temperature"

x=0.1 [m]                                        "x-position"
y=0.1 [m]                                        "y-position"

N=100 [-]                                        "number of terms"
duplicate i=1,N
    T[i]=2*T_b*(1-cos(i*pi))*sin(i*pi*x/W)*exp(-i*pi*y/W)/(i*pi)
end
T=sum(T[1..N])
```

Solving these equations results in T = 0.4895 K.

Three Column Data

All of the 3-D plot types require three dimensional data. These data can either be provided in three column form or in the form of a 2-D table. In three column form, the data are provided as three separate columns corresponding to the two independent variables and the dependent variable. For this problem we can provide the data in three column form by creating a Parametric Table that includes the variables x, y, and T.

In three column form, the independent variables can be provided in any order and they do not have to be located on a regular grid. The first step that EES will do internally is interpolate/extrapolate the provided data to provide a suitable grid. To generate the data that will be plotted, we will vary the independent variable *x* from 0 to 1 m in 21 discrete values. The dependent variable *y* will be varied from 0 to 2 m in 41 discrete values. Therefore, the Parametric Table must contain 21x41 = 861 rows. To populate the column containing *x*, right click on the column header and select Alter Values to obtain the dialog shown in Figure 9-31(a). Vary *x* from 0 to 1 m and select Repeat pattern every 21 rows, as shown.

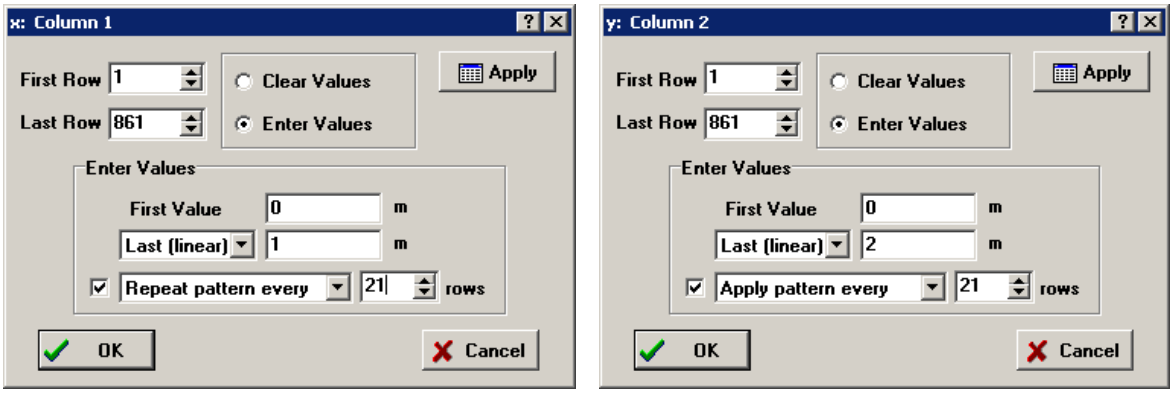


Figure 9-31: Alter values dialog for (a) *x* and (b) *y*.

To populate the column containing *y*, repeat the same process but vary *y* from 0 to 2 m and select Apply pattern every 21 rows, as shown in Figure 9-31(b). If you examine your Parametric Table you will find that the dependent variables form a 2-D grid over the computational domain $0 < x < 1$ m and $0 < y < 2$ m. Comment out the values of *x* and *y* that were previously entered in the Equations Window:

```
{x=0.1 [m]                                    "x-position"
y=0.1 [m]                                     "y-position"}
```

and solve the table to obtain a set of data in three column format.

Isometric Lines Contour Plot

Select New Plot Window from the Plots menu and then select X-Y-Z Plot to access the X-Y-Z Plot Setup dialog shown in Figure 9-32. A contour plot illustrates lines of constant values of the dependent variable (i.e., isometric lines) in the parameter space defined by the two independent variables. In order to generate a contour plot, select Isometric Lines from the list of plot types. Select Label contours in order to apply a label to each contour line.

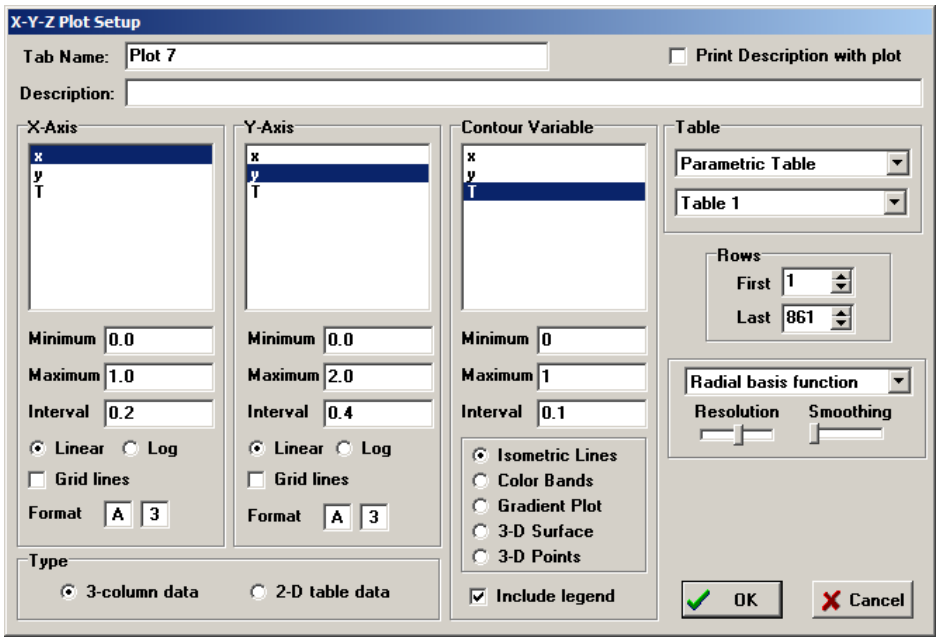


Figure 9-32: X-Y-Z Plot Setup dialog.

For this problem, select the variables x and y as the independent variables and T as the contour variable. The minimum and maximum values of the contours and the interval between contours can be defined. The data in the columns must be interpolated/extrapolated by the internal plotting algorithms so as to provide values of the independent variable over the entire selected range of the dependent variables at relatively fine intervals. This process will be done using either Radial basis functions or Bi-quadratic polynomials depending on the selection from the drop-down menu in the lower right corner of the dialog. The grid size is controlled by the resolution slider. The smoothing slider is useful for experimental data or for a very sparse data set where outliers or large gaps may be present; normally the smoothing slider should be set to zero. Select OK in order to generate the contour plot shown in Figure 9-33.

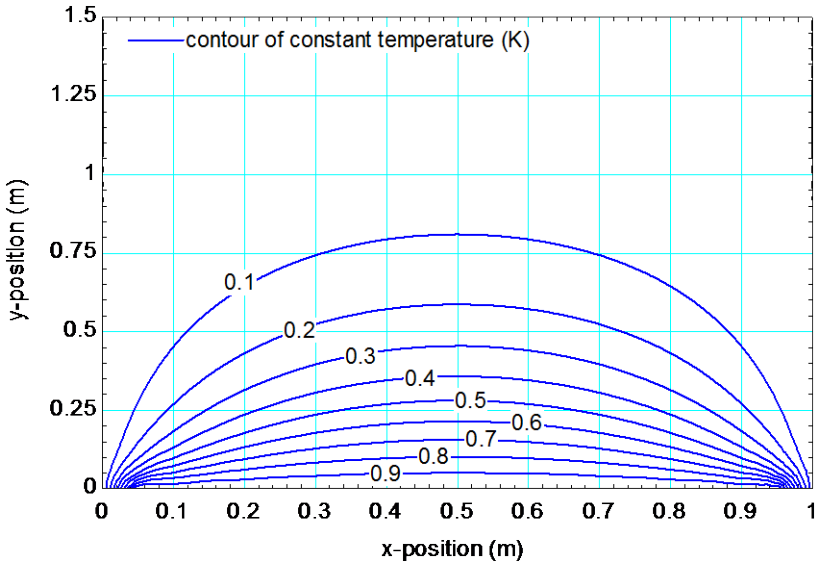


Figure 9-33: Isometric lines contour plot showing temperature as a function of *x* and *y*.

Color Bands Contour Plot

The Color Bands option in the X-Y-Z Plot Setup dialog shown in Figure 9-32 produces the plot shown in Figure 9-34. Each contour is indicated by a color (e.g., the range from 0.9 to 1 in Figure 9-34 is indicated by the region in red).

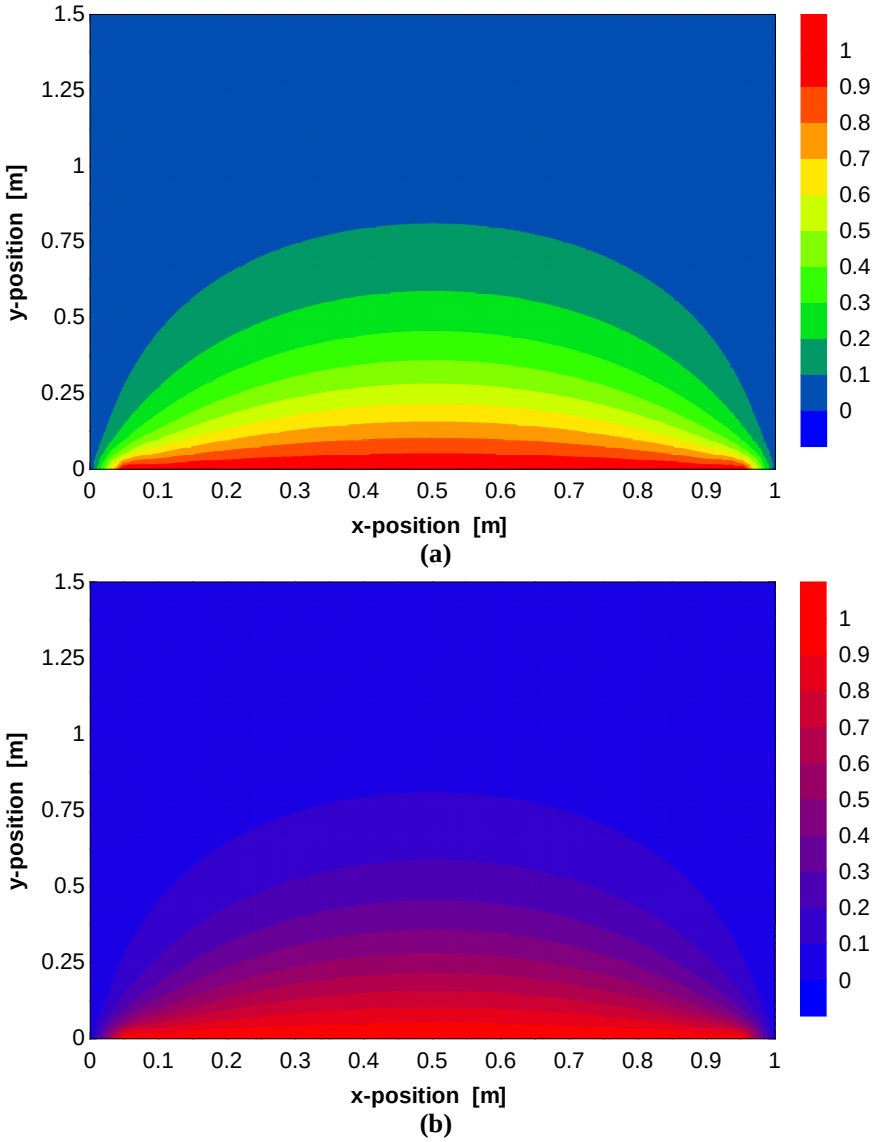


Figure 9-34: Color band contour plot using (a) full spectrum color and (b) blue to red color.

Two color schemes are available. The color scheme shown in Figure 9-34 is called the Full Spectrum color scheme and it uses all of the colors of the rainbow. The alternative scheme uses colors ranging from blue to red. In some cases, this color scheme is more easily associated with a value. To change the color scheme, click the right mouse button anywhere within the plot rectangle in order to display the Modify Plot dialog shown in Figure 9-35(a). The color scheme can be changed to blue to red by unchecking the Full spectrum check box. Controls are also provided to hide or show the legend and to superimpose gradient arrows, which are discussed in the following section.

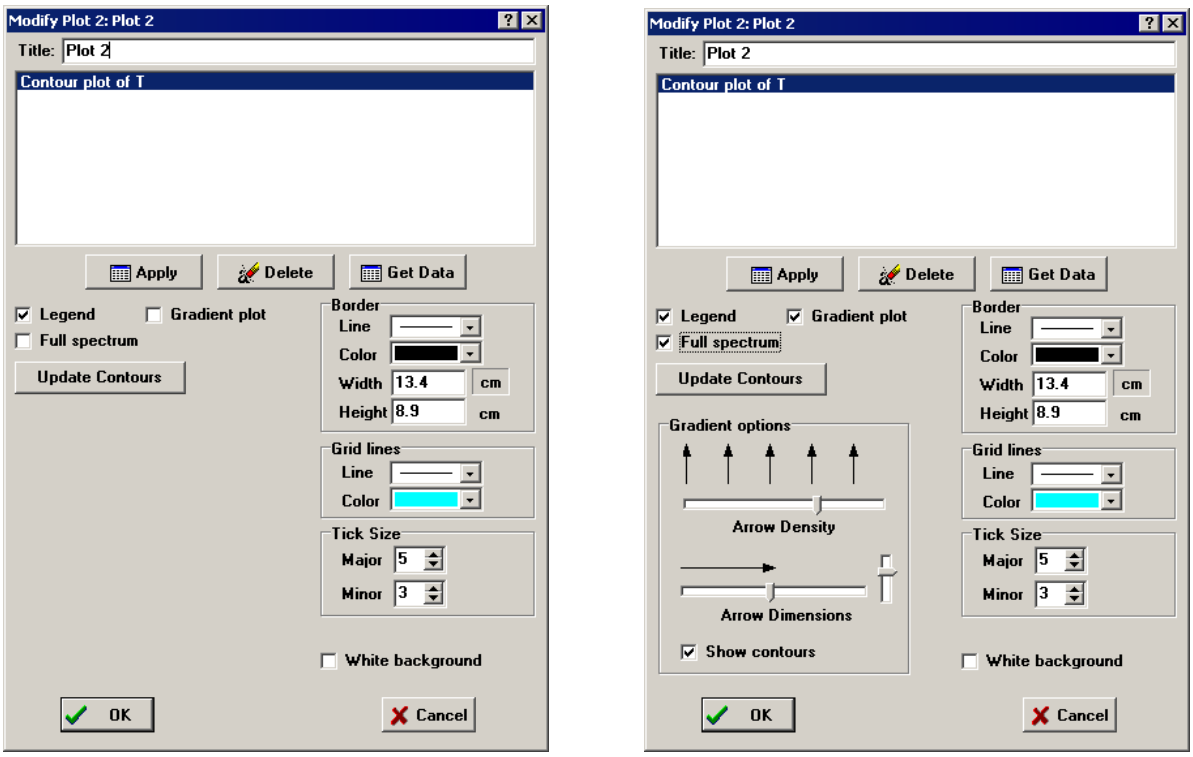


Figure 9-35: Modify Plot dialog for (a) a color band contour plot and (b) a gradient plot

Gradient Plot

If Gradient plot is selected from the list in the X-Y-Z Plot Setup dialog in Figure 9-32 then a Color Band plot is generated that includes arrows indicating the direction and magnitude of the gradient of the independent variable, as shown in Figure 9-36. The frequency and maximum length of the arrows, as well as the size of the arrowhead, can be modified by right-clicking on the plot. This action will bring up the Modify Plot diagram shown in Figure 9-35(b).

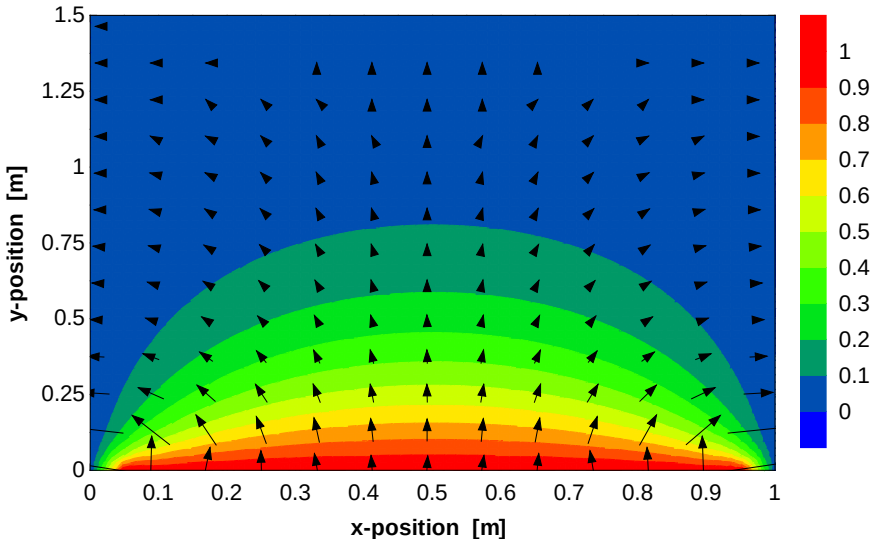


Figure 9-36: Gradient plot.

Three-Dimensional Surface Plot and Control Panel

There are two versions of EES provided in the installation program. One version is capable of generating 3-D surface and point plots that can be rotated. This version will be installed only if the 3-D plotting option is selected during the EES setup; by default, this option is not selected. Be sure that you are using the 3-D version before attempting to create 3-D surface and point plots. If you are unsure of what version you are using, select the About EES menu item in the Help menu. The startup screen will appear and -3D will be appended to the version number if you are using a 3-D version.

A 3-D surface plot can be created using the temperature as a function of x and y position data generated in the previous section. Select New Plot and then X-Y-Z from the Plots menu. Select 3-D Surface from the list in the X-Y-Z Plot Setup dialog shown in Figure 9-32 in order to display a three-dimensional, rotatable projection of the surface. The resulting plot is shown in Figure 9-37.

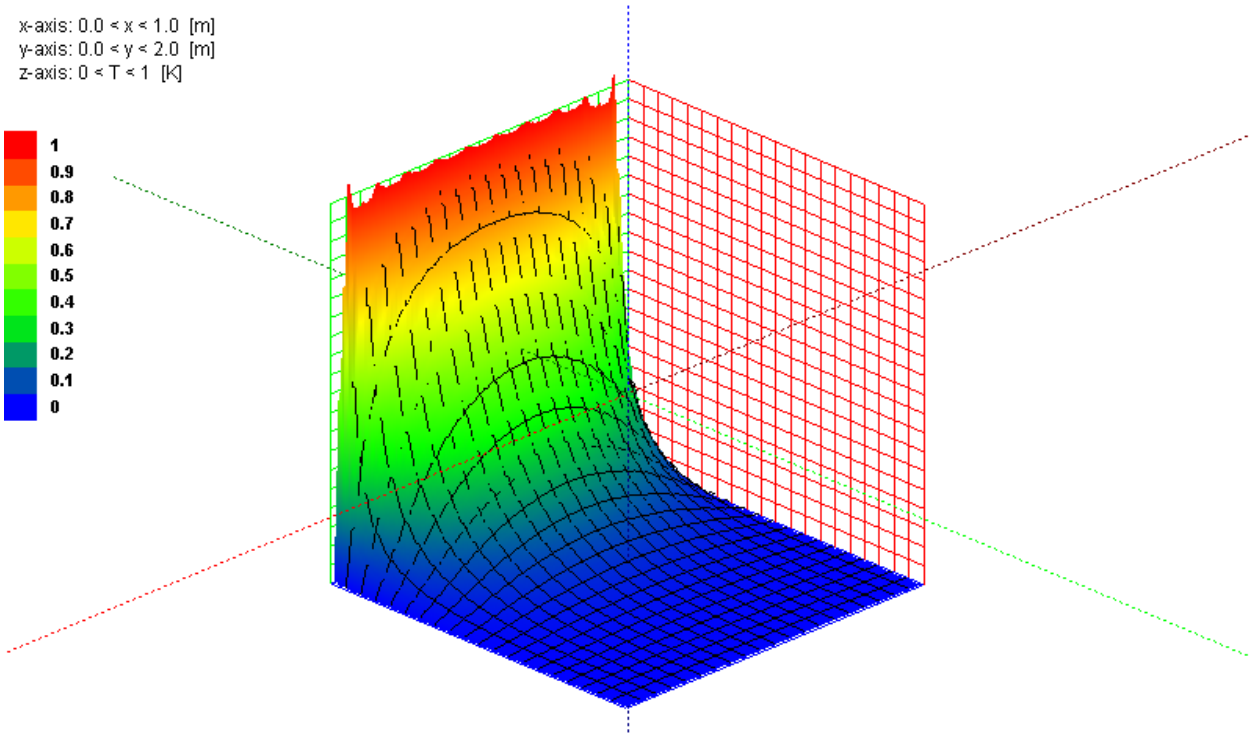


Figure 9-37: Three-dimensional plot.

The control panel shown in Figure 9-38 is provided at the bottom of the plot window. If the panel is not visible, click the right mouse button anywhere in the plot and it will appear. The same action will make it disappear. Alternatively, you can use the Show/Hide Tool Bar menu item in the Plots menu to control the visibility of the control panel.

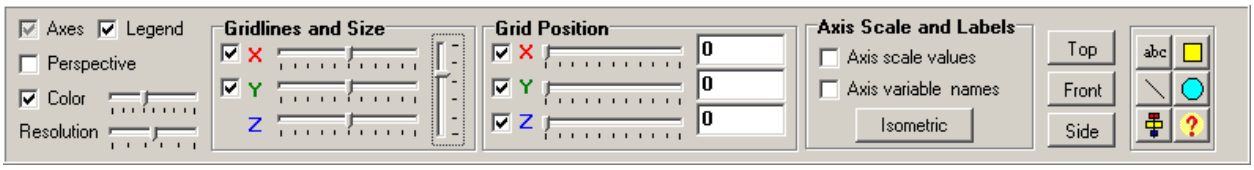


Figure 9-38: Control Panel for 3-D Surface Plots.

The Axes check box controls the visibility of the three axis lines. This check box has three positions that are activated by clicking the box. If it is not checked, the axis lines are not visible. If it is checked with a gray check, the axis lines are drawn without labels, as shown in Figure 9-37. Clicking the box one more time will show the axis lines with axis labels. The legend displayed at the left of the plot window in Figure 9-37 can be made visible or hidden with the Legend check box. The Perspective check box, if selected, distorts the display so that objects that are further back appear smaller. The Resolution slider controls the size of the polygons used to form the surface plot. High resolution (slider bar to the right) results in a crisp plot, but rendering and rotating may be slower

The Color check box has three positions that engage with a mouse click. The three possibilities are black and white (no check); color display with colors ranging from blue to red (gray check); and color display with full spectrum color (black check). Normally, the color (or shade of gray for black and white plots) is a visual indicator of the Z-axis variable. However, it is possible to have the color provide a totally different piece of information, which effectively provides a 4-D plot.

To demonstrate this capability, we will assign the color of the surface plot to a variable C, which is included in the equations used to generate T as a function of x and y. The variable C is set to 0 if the temperature is less than 0.5 K and 1 if it is greater than 1 K.

```
C=if(T,0.5,0,0,1)
```

Add a fourth column for the variable C to the Parametric table and resolve the table. Select 3-D Surface from the list in the X-Y-Z Plot Setup dialog in Figure 9-32, as was done before. Select T to be the Z-axis variable. Next click on Z-Axis Variable located above the Z-axis list. It will change to Color Variable and the list will be displayed in yellow. Note that if you click Color Variable then it will return the display to Z-Axis Variable. Select variable C, as shown in Figure 9-39, and the click the OK button. The plot shown in Figure 9-40 will be generated and it will indicate with its color the areas in which the temperature is greater than 0.5 K. Note that Figure 9-40 shows colors that indicate values that are between 0 and 1 even though all of the original data for variable C were either 0 or 1. This effect occurs because EES must grid the data to create the plot and it uses interpolative procedures to do so. The effect of interpolated variables can be minimized using the color slider control shown at the left of Figure 9-38.

The Gridlines and Size box provides slider controls to separately scale the X, Y and Z axes. Moving the slider to the right increases the scale factor for the selected axis. The slider can be moved with the mouse or with the left/right arrow keys. The check boxes for X and Y control the visibility of surface grid lines. The vertical slider scales the X, Y, and Z directions uniformly. Moving the slider down increases the apparent size of the plot in the window. This

slider can also be controlled with the up/down arrow keys. The lower and upper limits of the plotting range for the surface can be changed by right-clicking on the slider control for the X and Y axes; doing so will bring up a small dialog in which you can enter the lower and upper limit for which the surface plot will be drawn.

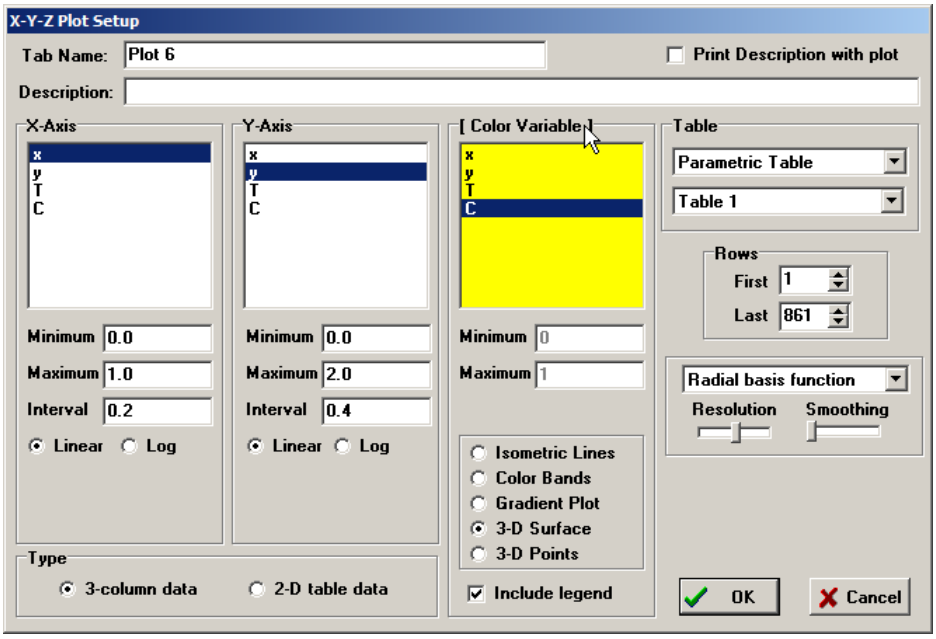


Figure 9-39: X-Y-Z Plot Setup dialog with the color variable assigned to variable C.

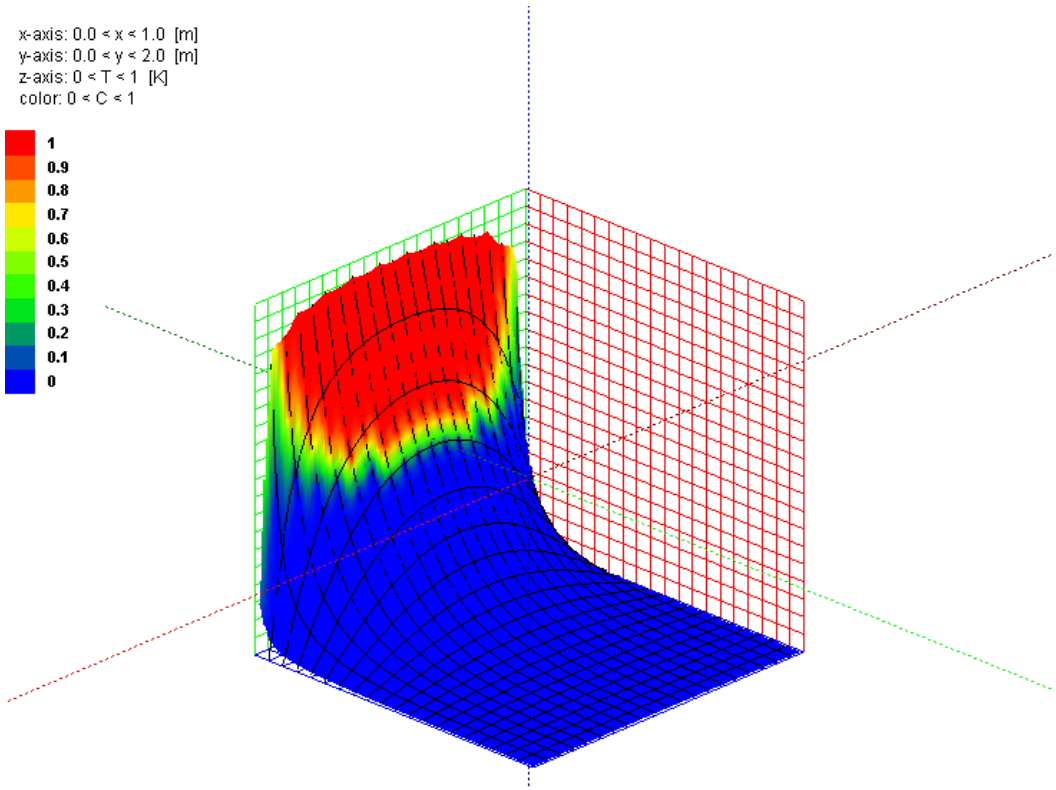


Figure 9-40: 3-D plot of T as a function of x and y. The color is set to variable C.

The Grid Position box provides controls to display X, Y and Z orthogonal grids. The visibility of these grids is controlled with the checkboxes. For example, the grid in the YZ plane will be visible if the checkbox preceding X is checked. The grid can be moved to any position within the range of allowable values using the slider control. The slider controls can be controlled using the mouse or the left/right arrow keys. The current location of the grid is shown in the box to the right of the slider. A value can be directly entered into this box in order to move the grid to the specified value. By default, the grid spans the entire range for each variable. However, this default can be changed. For example, right click on the slider for the X grid. A small dialog will appear that allows the minimum and maximum X values for the grid to be changed. Change the default values of 0 and 1 to 0.5 and 1, as shown in Figure 9-41. Click the OK button and the 3-D surface plot will be redrawn as shown in Figure 9-42

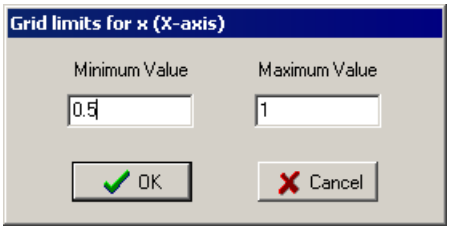


Figure 9-41: Specifying minimum and maximum values for a grid.

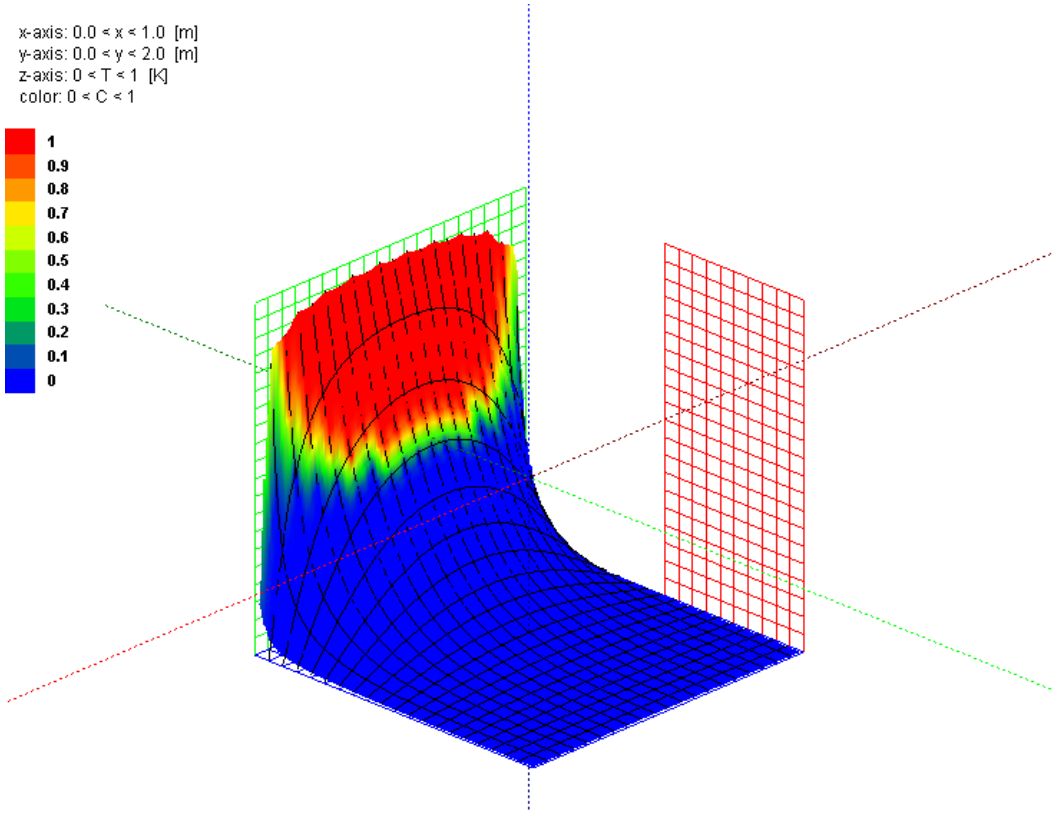


Figure 9-42: 3-D plot of *T* as a function of *x* and *y* with the X grid drawn between 0.5 and 1.

The Axis Scale and Labels box provides controls to label the axes. If the Axis scale values check box is checked, the lower and upper limits of the scales for the X, Y, and Z axes will be displayed. If the Axis variable names check box is checked, the names of the variables plotted on the X, Y and Z axes will be displayed. Clicking the Isometric button will orient the axes in an isometric manner that best shows the axis scale values and axis names. Clicking the Top button will rotate the display so that the view is of the X-Y plane looking down the positive Z-axis towards the origin. The Front button presents a Y-Z plane viewed along the positive X axis. The Side button displays the X-Z plane from the positive Y axis.

The tool bar provides buttons to place text, lines, boxes, and circles on the plot in the same manner as for X-Y plots. An align button is also provided. The help button accesses the online help page with information on the use of the control panel.

Holding the Shift and Ctrl keys down will cause the 3-D coordinates to be displayed in the window caption, provided that the X and Y-axis grids are visible. The information displayed is the current locations of the X and Y grids, followed by the value of Z on the surface for the specified values X and Y. If the color corresponds to a variable other than Z, its value on the surface at the specified X and Y coordinates is also shown. If the X and Y grids are moved using the mouse to manipulate the slider control while the Shift and Ctrl keys are held down, the information in the window caption will be automatically updated. This option provides the most convenient way to accurately read the 3-D coordinates.

Two-Dimensional Table Data

The three-dimensional data required by an X-Y-Z plot can also be provided in the form of a 2-D table. To illustrate this format, let's return to the transient conduction problem that was used in Section 9.2 in order to demonstrate time sequence plots.

```
$UnitSystem SI Mass J K Pa Rad
$TabStops 0.2 0.4 3.5

rho=8933 [kg/m^3]                          "density"
c=385 [J/kg-K]                             "specific heat capacity"
alpha=1.166e-4 [m^2/s]                     "thermal diffusivity"
E``=1e6 [J/m^2]                            "pulse of energy per area"
T_ini=300 [K]                              "initial temperature"

x_sim=0.25 [m]                             "simulation extent"
n=41 [-]
duplicate i=1,n
    x[i]=(i-1)*x_sim/(n-1)                 "position"
end

time_sim=60 [s]                            "simulation time"
m=20 [-]                                   "number of times to simulate"
duplicate j=1,m
    time[j]=j*time_sim/m                   "time"
end

duplicate j=1,m
```

```
    duplicate i=1,n
        T[i,j]=SemiInf4(T_ini,E``,rho,c,alpha,x[i],time[j])
    end
end
```

Running this simulation will result in an Arrays Table that includes 20 columns corresponding to the spatial temperature distribution at each of the simulated times in addition to the column time that includes the times simulated (3 s to 60 s) and the column x that includes the locations simulated (0 m to 0.25 m). Select the Sort button at the upper right of the Arrays Table window and the temperature information will be placed in rows 1 through 41 of columns 1 through 20.

Select New Plot Window from the Plots Window and then X-Y-Z plot. Select 2-D table data under type in order to access the dialog shown in Figure 9-43.

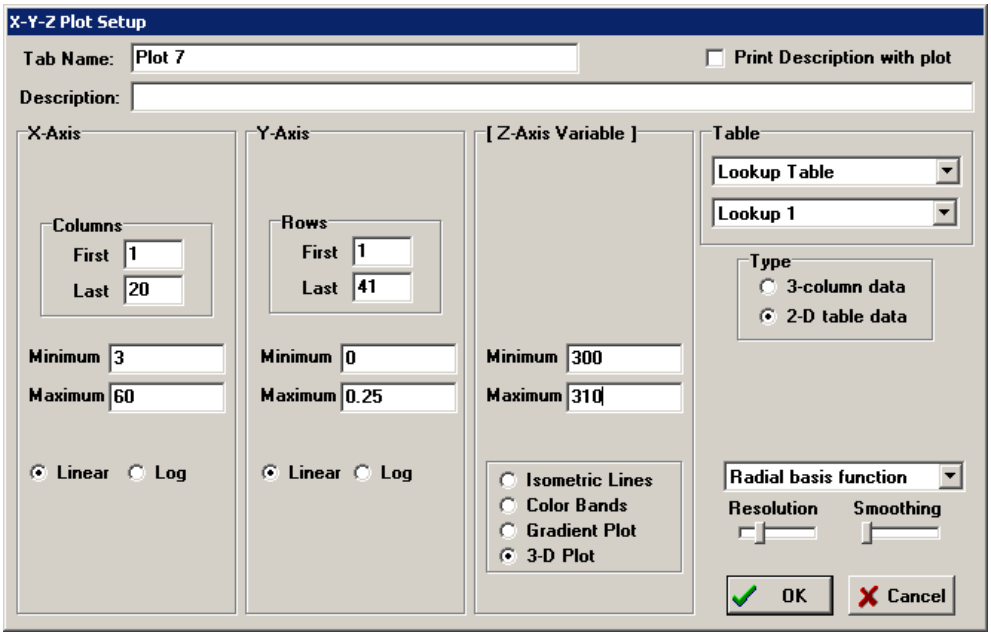


Figure 9-43: X-Y-Z Plot Setup dialog using 2-D table data.

Select 3-D plot and then specify the columns (1-20) and rows (1-41) that contain the data. Specify the range for the X-Axis (3 s to 60 s) and the range for the Y-Axis (0 m to 0.25 m). The resulting plot is shown in Figure 9-44. Note that text to label the axes has been added using the text tool in the control panel in the same manner as for X-Y plots.

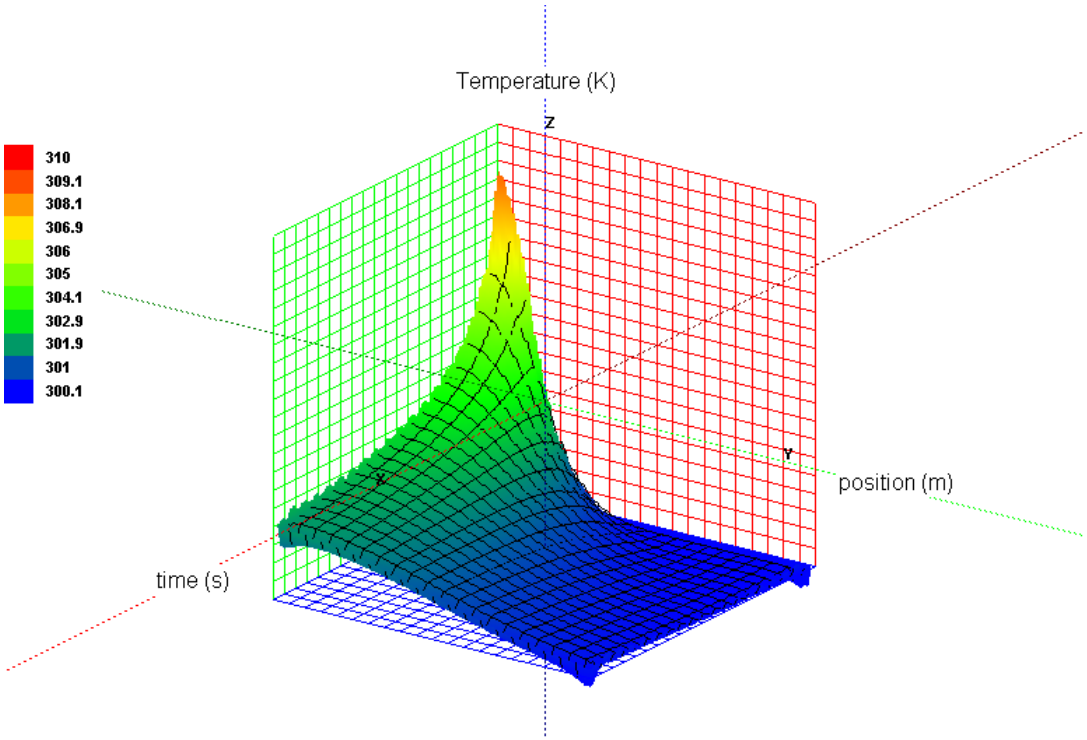


Figure 9-44: 3-D graph of temperature as a function of time and position.

3-D Points Plots

The Professional version can plot 3-dimensional data as a set of points in 3-D space, either by itself or superimposed on a 3-D surface plot. To illustrate this capability, we will plot the vapor dome on temperature-pressure-specific volume coordinates for water. Enter the following short program, which calculates the specific volume of saturated liquid water and saturated vapor water as a function of temperature for temperatures between 300 K and the critical temperature. The saturation pressure is also calculated. Because the specific volume and pressure vary by orders of magnitude, we will plot the natural logarithms of these quantities which are calculated in this program.

```
$UnitSystem SI K kPa kJ
$TabStops 0.2 3.5 in
N=100                                   "number of data point"
T_low=300 [K]                           "lowest temperature considered"
T_c=T_crit(water)                       "critical temperature"
duplicate i=1, N
    T[i]=T_low+(T_c-T_low)*i/N          "temperatures between T_low and T_c"
    P[i]=pressure(Water,T=T[i],x=1)     "saturation pressure"
    vg[i]=volume(Water,T=T[i],x=1)      "specific volume of saturated vapor"
    vf[i]=volume(Water,T=T[i],x=0)      "specific volume of saturated liquid"
    lvg[i]=ln(vg[i])                    "natural log of saturated vapor volume"
    lvf[i]=ln(vf[i])                    "natural log of saturated liquid volume"
    lnP[i]=ln(P[i])                     "natural log of pressure"
end
```

Solve the program (select F2) and the select New Plot Window and then X-Y-Z plot from the Plots menu. Select 3-D Points as the plot type and choose the X, Y, and Z-Axis variables as indicated in Figure 9-45. The limits for the Z-Axis Variable (log of the specific volume) are selected so that both saturated liquid and vapor will be displayed on the plot.

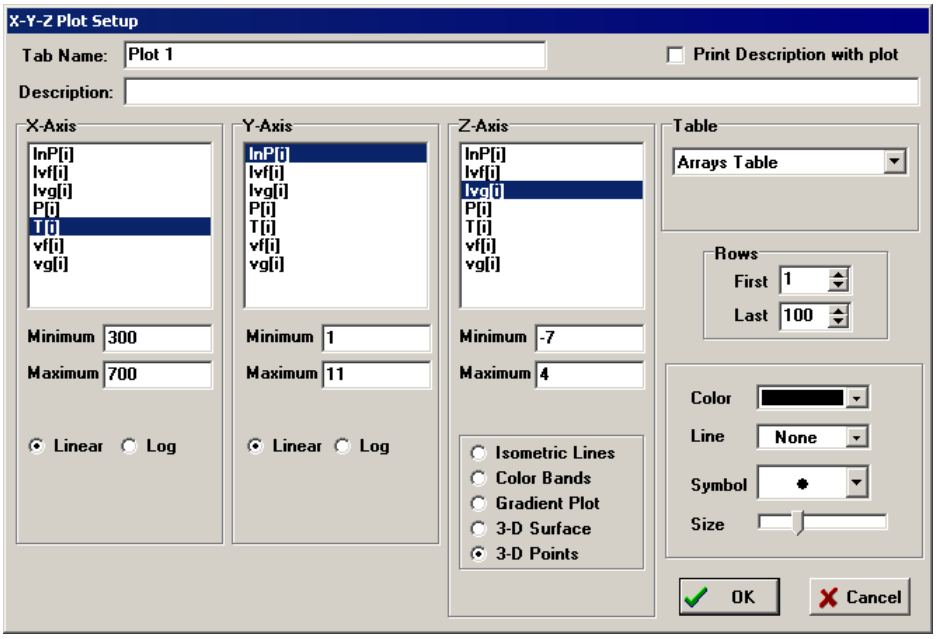


Figure 9-45: Plot Setup for a 3-D Points plot of the vapor dome for water.

Click OK and the plot will be generated showing the saturated vapor line in 3-D space. Next, select Overlay Plot from the Plots menu. Select the arrays T[i] and lnP[i] for the X-axis and Y-axis variables, as before. Select lvg[i] for the Z-axis variable. Change the color to red so that the saturated liquid line is distinct from the saturated vapor line. Click OK. The plot will appear as shown in Figure 9-45. The plot can be rotated to allow views from any direction. The control panel that appears below the plot operates in the same manner as for the 3-D surface plots. One difference is that the control panel provides a Modify Plot button (see Figure 9-47) that allows the color, symbol, symbol size, line type and other information to be modified for any of the overlaid 3-D plots. It is also possible to delete a plot within the Modify-Plot dialog.

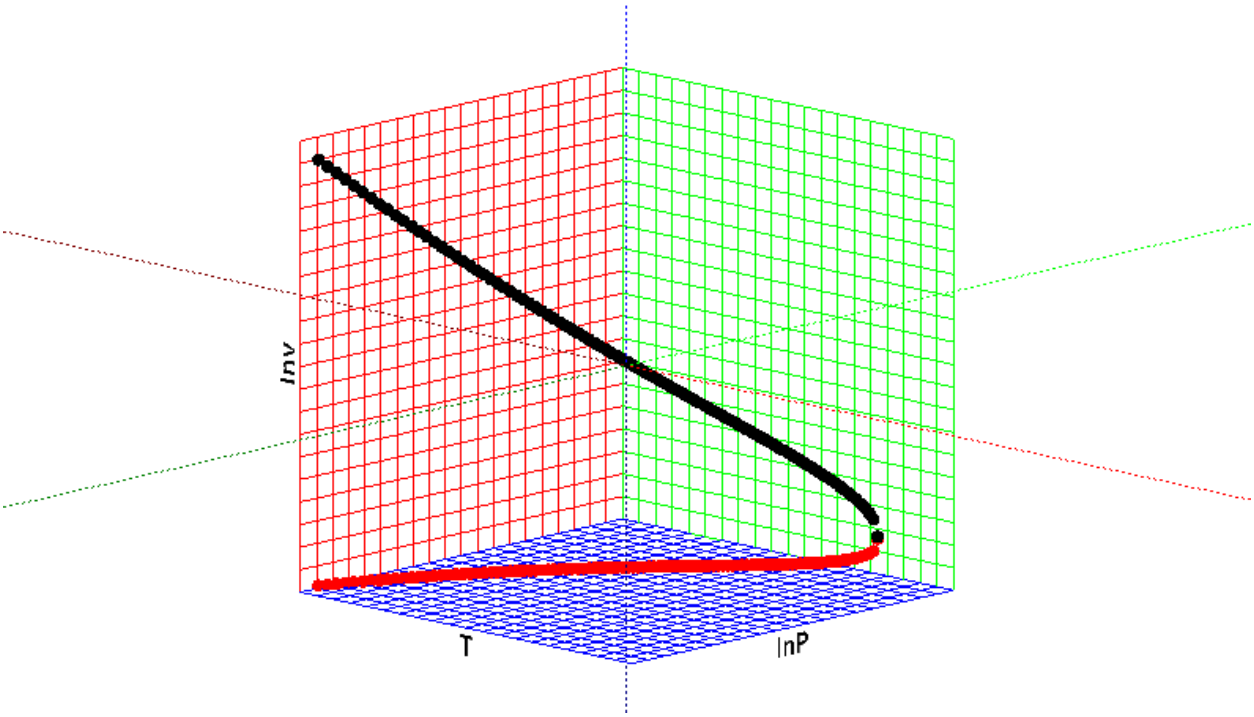


Figure 9-46: 3-D plot of ln(specific volume) for saturated liquid and vapor as a function of temperature and ln(pressure).

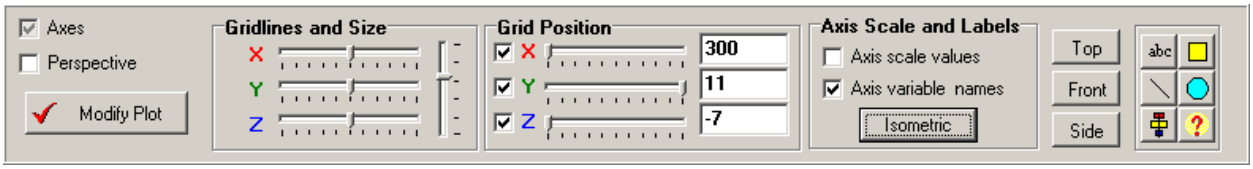
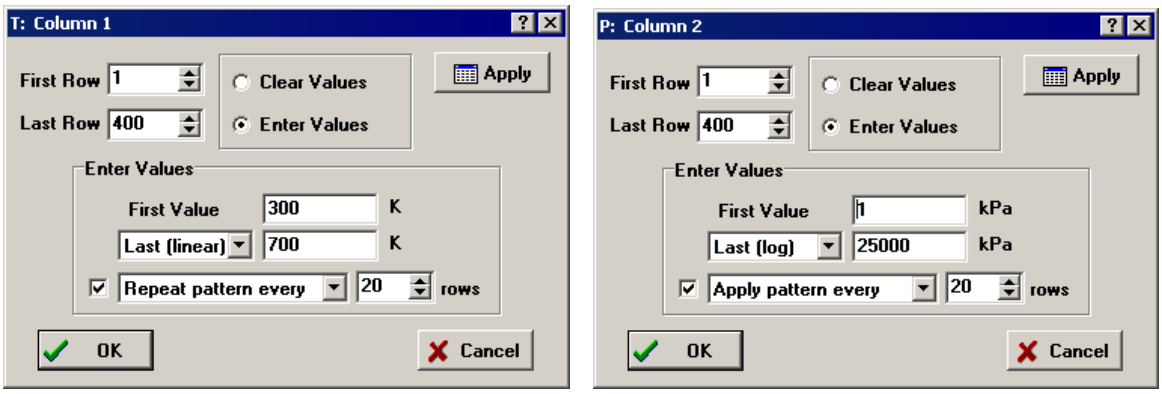


Figure 9-47. Control panel for 3-D points plots.

It is also possible to overlay 3-D point plots on a 3-D surface plot. The 3-D surface plot must be created first. To illustrate, we will overlay the saturation specific volume plots in Figure 9-46 on a surface plot of temperature, ln(pressure) and ln(specific volume). Add the following equations to the Equations Window.

```
v=volume(Water,P=P,T=T)          "specific volume as a function of T and P"
lnv=ln(v)                        "natural log of specific volume"
lnP=ln(P)                        "natural log of pressure"
```

Create a Parametric Table with 400 runs and columns for the variables T, P, v, lnP and lnv. Fill the Parametric Table with temperature data ranging from 300 to 700 K using the Repeat pattern every options for 20 rows, as shown in Figure 9-48(a). Fill in the pressure column with values between 1 and 25000 kPa. Use the Last (log) option to space the pressures logarithmically and Apply the pattern every 20 rows, as shown in Figure 9-48(b). Solve the Parametric Table (using the F3 key).



(a) **(b)**

Figure 9-48: Filling the Parametric table with (a) temperature and (b) pressure data for the 3-D plot.

Select New Plot Window and then X-Y-Z plot from the Plots menu. Select 3-D Surface as the plot type and choose the X, Y, and Z Axis variables to be T, lnP and lnv, respectively. Select OK and then overlay the saturated specific vapor volume and saturated liquid specific volume data from the Arrays Table onto this plot, following the same procedure used to create Figure 9-46. The plot will appear as shown in Figure 9-49. The saturated liquid line shown in red is partially obscured by the surface plot, but it can be viewed as the plot is rotated.

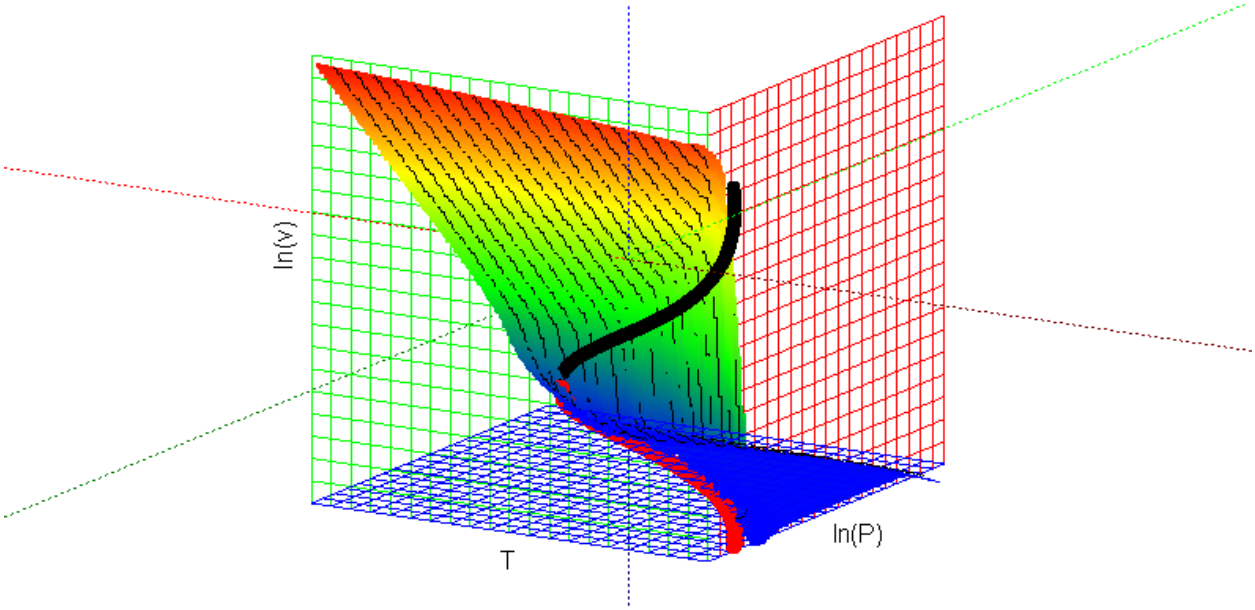


Figure 9-49: 3-D plot of ln(specific volume) as a function of temperature and ln(pressure)

References

Nellis, G.F. and S.A. Klein, *Heat Transfer*, Cambridge University Press, New York, (2009).

10 SUBPROGRAMS and MODULES

Subprograms are internal EES programs that can be called from the main EES program or from other subprograms. Modules are similar to subprograms, but they are implemented in a manner that results in the equations in the module being integrated and solved together with the equations in the Equations Window. Most problems can be successfully solved using either a subprogram or a module, but the convergence properties of the two approaches differ. The format of a module or subprogram is similar to that of an internal procedure, discussed in Chapter 3. The module or subprogram is supplied with inputs and it calculates and returns outputs. Like a procedure, a module or subprogram statement can use a colon in the argument list in order to separate inputs from outputs; however, the colon is optional for a module or subprogram (although it is recommended). This chapter describes how modules and subprogram can be effectively used within an EES program.

10.1 Subprograms

A subprogram is a stand-alone EES program that can be called from the main EES program. It can also be called from a function, procedure or another subprogram. Subprograms are not recursive; i.e., they cannot call themselves.

Format of a Subprogram

The format of a subprogram is shown below[3]:

```
SubProgram SubProgram_Name(Input 1, Input 2, ..., Input N: Output 1, Output 2, ..., Output M)

    Equation(s) - note that these equations must be sufficient to compute the outputs given the inputs

End
```

The format of a subprogram is similar to that of a procedure, as described in Section 3.3. Both types of code segments must be entered in the Equations Window before the first equation in the main EES program appears and both must provide an argument list that includes the input and output variables. However, there are significant differences between a subprogram and procedure. The major difference is that the statements that appear in a subprogram are equations like those that are used in the main EES program. Therefore, the equations can be entered in any order and it is not necessary to isolate the variable that is being determined on the left side of the equal sign. The statements that appear within a procedure are, in contrast, assignment statements as discussed in Section 3.1. Logic constructs such as If-Then-Else and Repeat-Until statements can be used in a procedure but not within a subprogram.

[3] Note that the list separator is a semicolon (instead of a comma) if the computer is configured to operate in the European system in which the decimal separator is a comma.

Each subprogram has its own variable information page in the Variable Information dialog and its own tab in the Solution Window. Subprograms can use array variables. Each subprogram will have its own tab in the Arrays Table window if the $Arrays On directive appears in the subprogram. Each subprogram can have up to 6,000 equations in the Commercial version of EES and 12,000 equations in the Professional version.

The argument list for a subprogram normally includes a colon to separate inputs from outputs; however, the colon is optional. If a colon is not provided then EES will determine the required number of inputs from examination of the number of equations in the subprogram. Inputs always appear before outputs in the argument list, regardless of whether or not a colon is used. Variables that are defined as being inputs differ from those defined as being outputs in the manner in which the guess values and limits are assigned, as explained subsequently. The input-output designation of the arguments does not restrict the use of a subprogram in that it is usually possible to specify the values of one or more outputs so as to calculate a corresponding number of inputs.

Calling a Subprogram

A subprogram may be called from the main EES program, a function, a procedure or another subprogram using the Call statement. The Call statement for a subprogram has the same format as for a procedure. For example:

```
Call SubProgram_Name(Input 1, Input 2, ..., Input N: Output 1, Output 2, ..., Output M)
```

Note that if the subprogram header employs a colon to separate inputs and outputs, the Call statement must also use the colon. Use of the colon is recommended unless the subprogram is intended to have a varying number of inputs. The number and type (string or numerical) of the variables passed to the subprogram must exactly match the number and type declared in the subprogram header statement. Variable information can also be passed into the subprogram from the main EES program using a $Common directive, as described in Section 14.3.

Subprogram Execution

When a Call statement for a subprogram is executed, EES clears its variable space, loads the subprogram and attempts to solve the equations in the subprogram. The first thing EES will do is to check to see if the number of equations in the subprogram plus the number of inputs to the subprogram is equal to the total number of variables. If this is not true then the problem cannot be solved.

Consider as a simple example, the following subprogram with one input (X) and two outputs (Y and Z). The subprogram consists of a single equation:

$$X = 2Y + 3Z \tag{10-1}$$

```
Subprogram DOF(X: Y, Z)
    X=2*Y+3*Z
end
```

When EES attempts to solve this subprogram it will realize that there are three variables (X, Y, and Z) but the sum of the number of inputs (1) and the number of equations (1) is only two; therefore, the problem is under-specified. For example, enter the equations:

```
A=5
Call DOF(A: B, C)
```

in the Equations Window. The equations in the subprogram cannot be solved and therefore the error message shown in Figure 10-1 is displayed.

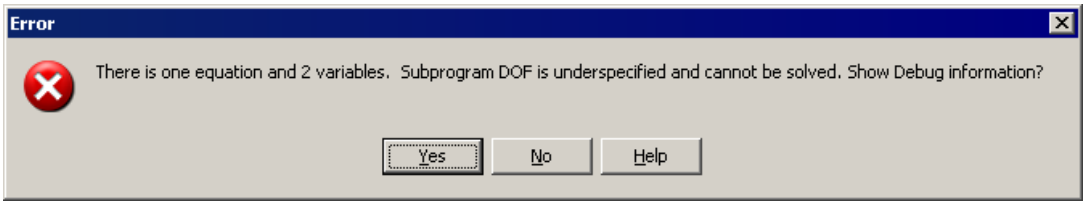


Figure 10-1: Error message resulting from trying to solve the under-specified problem.

If a second equation is provided in the body of the subprogram,

$$X^2 = Y^2 + Z^2 \tag{10-2}$$

```
Subprogram DOF(X: Y, Z)
    X=2*Y+3*Z
    X^2=Y^2+Z^2
end

A=5
Call DOF(A: B, C)
```

then the problem will solve, returning the values of B and C to the Main program.

Solution and Residuals Windows

The Solution Window will include a tab with the name of the subprogram. The Solution Window for the subprogram will display the local values of the variables, in the subprogram for the last time it was called, as shown in Figure 10-2.

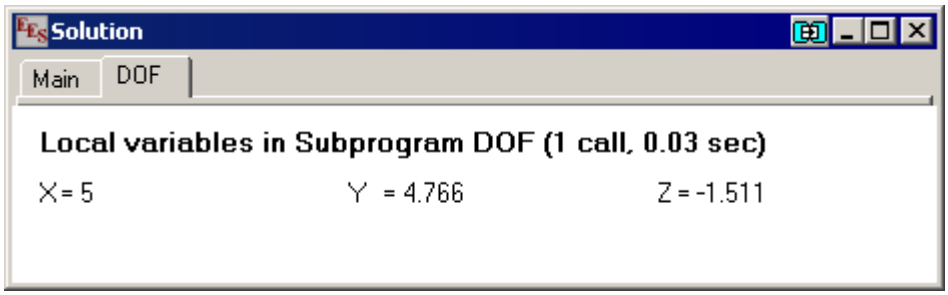


Figure 10-2: Solution window for subprogram DOF.

The Residuals Window will also provide information for the equations in the subprogram, as seen in Figure 10-3. In this case, the Residuals window shows that variables Y and Z were determined simultaneously using the two equations in the subprogram. The variable X is an input whereas the variables Y and Z are outputs and are thus shown in bold. The value of X and the two equations in the Subprogram are used to solve for the values of Y and Z.

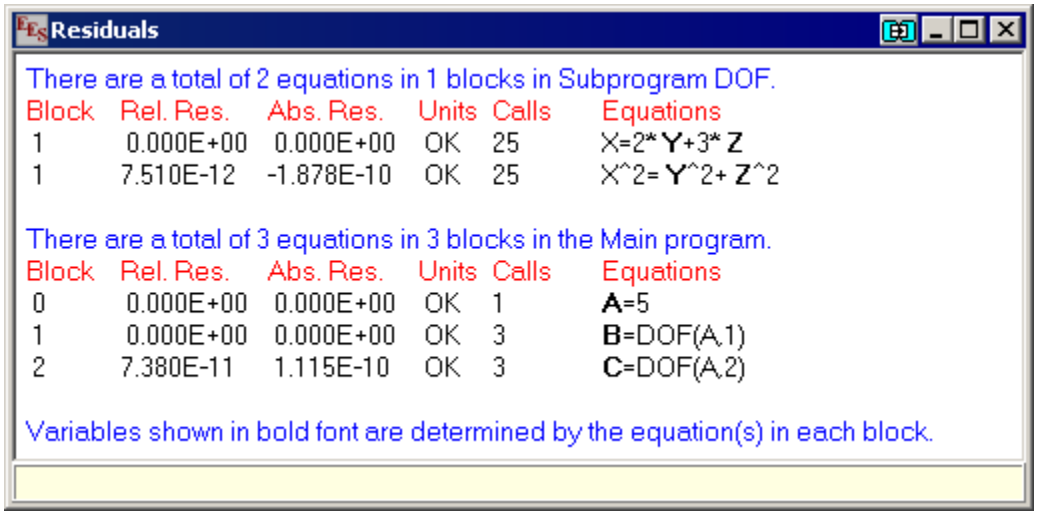


Figure 10-3: Residuals window showing results for subprogram DOF.

One of the advantages of a subprogram is that it can be called multiple times. Another is that the values of one or more of the outputs can be specified in order to calculate one or more inputs. For example, we could add another call to the subprogram DOF at the bottom of the Equations Window:

```
Subprogram DOF(X: Y, Z)
    X=2*Y+3*Z
    X^2=Y^2+Z^2
end

A=5
Call DOF(A: B, C)

B2=6
Call DOF(A2: B2, C2)
```

Note that in the second call to the subprogram DOF the value of B2 is specified even though it is still considered to be an output of the subprogram. EES will automatically adjust the value of the variable A2 (which is the input to the subprogram) and repeatedly call the subprogram DOF until the calculated value of B2 is equal to the specified value (6). With the two Call statements in the program, the values of variables B, C, A2, and C2 will all be determined, as shown in the Solution Window for the Main program, Figure 10-4. The Solution and Residuals Window for the subprogram DOF will only show the results for the last call to the subprogram.

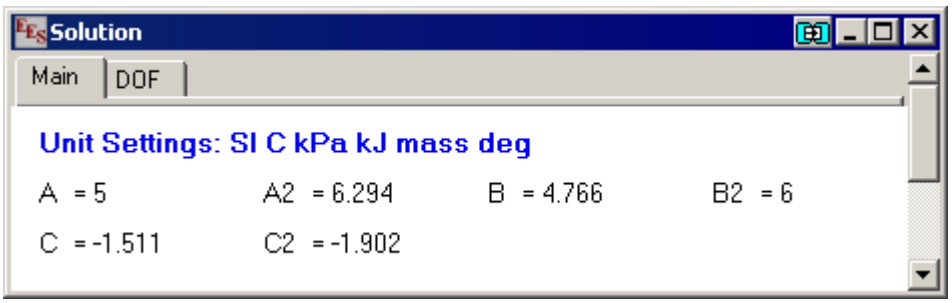


Figure 10-4: Solution Window showing results for the Main program

Guess Values and Limits of Inputs and Outputs

Each variable appearing in a subprogram has an associated guess value and lower and upper limits as well as a display format and units just as the variables in the main program do. This information can be viewed or changed by selecting Variable Info from the Options menu. Select the subprogram of interest using the drop-down list control at the top center of the Variable Information dialog window shown in Figure 10-5.

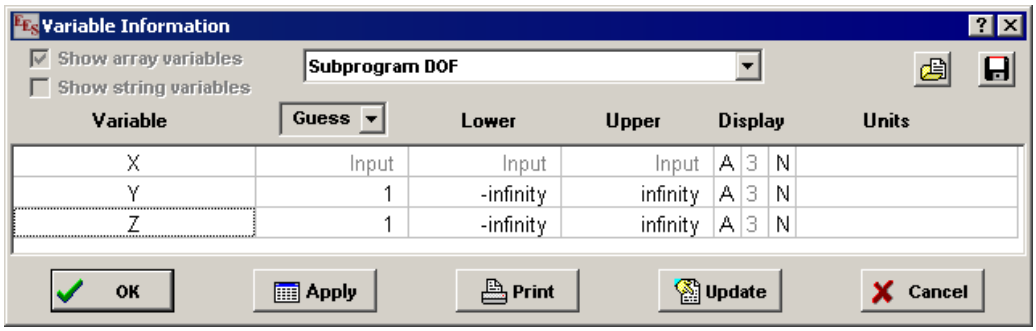


Figure 10-5: Variable Information dialog for the subprogram DOF.

The equations in the subprogram DOF are non-linear and thus they can have more than one solution. The solution that results when B2 in the main program (and therefore Y in the subprogram DOF) is set to 6 is shown in Figure 10-4: A2 = 6.294 in the main program (and X = 6.294 in the subprogram DOF). However, what if you knew that the variable A2 in the Main program (and therefore the variable X in subprogram DOF) must have a negative value? In this case, it seems logical to try to set the guess value for the variable X in the subprogram DOF to a negative value and also constrain it to have an upper limit of zero using the Variable Information window. This situation illustrates the distinction between inputs and outputs to a subprogram. The guess value and limits of the inputs to a subprogram assume the values that are set for those parameters in the calling program and therefore these values cannot be adjusted using the Variable Information dialog for the subprogram. Figure 10-5 illustrates the Variable Information dialog for the subprogram DOF; notice that the word Input is displayed in gray for the guess value and lower/upper limit fields for the input variable X. These fields are disabled.

In order to limit the value of A2 (and therefore the value of X during the second call to DOF) to a negative value, the guess value and limits of the variable A2 in the calling program must be set. In this example, the calling program is the main program and the Variable Information dialog for

the main program is shown in Figure 10-6 with the guess value upper limit of A2 set to -1 and zero, respectively. After this change is made, solving will result in A2 = -9.294, and C2 = -7.098.

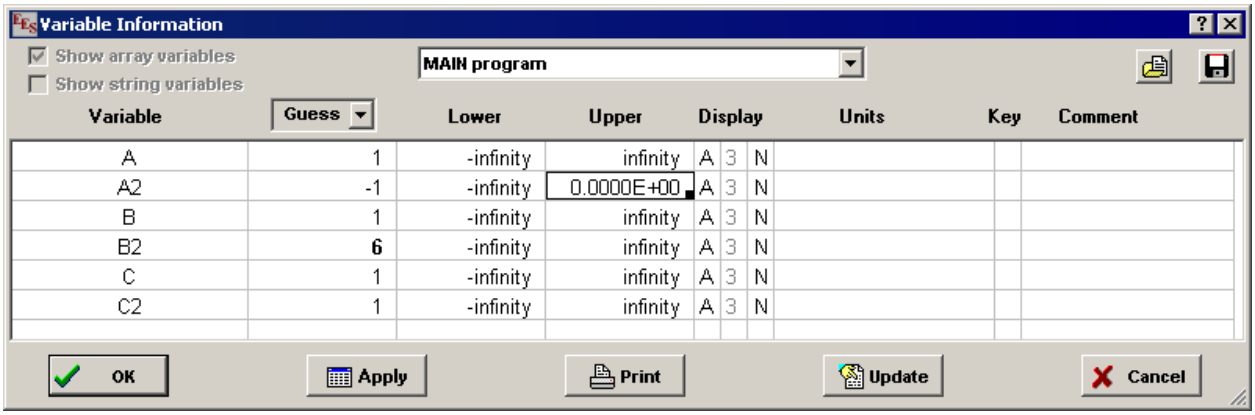


Figure 10-6: Variable Information dialog for variables in the Main program.

Subprograms with the Integral Command

Subprograms can be helpful when you wish to solve a set of equations that is somewhat independent of the equations in the main program. The use of subprograms for this purpose results in an organized code that is easy to understand. Subprograms are particularly useful when used with the equation-based Integral function, discussed in Section 7.2. When the Integral function is used in the main program, EES solves all of the equations as the integration variable is adjusted between its lower and upper limits in order to carry out the numerical integration.

The manner in which the Integral function is implemented places a number of restrictions on the use of integrals within the main program. For example, it is not possible to have two or more integrals that use the same integration variable with different limits in the main program. It is not possible to directly determine the limit of an integral that provides a specified value of the integrated value. Finally, EES may not be able to solve problems efficiently when two or more Integral functions are used with different integration variables. These limitations can be overcome by using subprograms to implement the Integral function.

As an example, we will approximate the pressure of a fluid (P) as a function of its specific volume (v) and temperature (T) using the Peng-Robinson equation of state, which is given in Eq. (10-3). (See Klein and Nellis (2011) for more information about this and other equations of state.)

$$P = \frac{RT}{(v-b)} - \frac{a}{v(v+b)+b(v-b)} \tag{10-3}$$

For this fluid, the ideal gas constant is R = 188.9 J/kg-K and the constants in Eq. (10-3) are a = 70.89 N-m^4/kg^2 and b = 0.0006059 m^3/kg. We wish to determine the work per unit mass of fluid that is required to isothermally compress from P_1 = 1x10^5 Pa to P_2 = 5x10^6 Pa at T = 350 K. We also would like to determine the work per unit mass that is required to isothermally compress the fluid from P_2 to P_3 = 1x10^7 Pa at the same temperature. The inputs are entered in EES:

```
$UnitSystem SI Mass J K Pa Rad

R=188.9 [J/kg-K]                    "gas constant for fluid"
a=70.89 [N-m^4/kg^2]                "Peng-Robinson parameter a"
b=0.0006059 [m^3/kg]                "Peng-Robinson parameter b"
T=350 [K]                           "temperature during compression"
P[1]=1e5 [Pa]                       "pressure at state 1"
P[2]=5e6 [Pa]                       "pressure at state 2"
P[3]=1e7 [Pa]                       "pressure at state 3"
```

The specific volumes of the fluid at states 1, 2, and 3 are evaluated using Eq. (10-3).

$$P_1 = \frac{RT}{(v_1 - b)} - \frac{a}{v_1(v_1 + b) + b(v_1 - b)} \tag{10-4}$$

$$P_2 = \frac{RT}{(v_2 - b)} - \frac{a}{v_2(v_2 + b) + b(v_2 - b)} \tag{10-5}$$

$$P_3 = \frac{RT}{(v_3 - b)} - \frac{a}{v_3(v_3 + b) + b(v_3 - b)} \tag{10-6}$$

```
P[1]=R*T/(v[1]-b)-a/(v[1]*(v[1]+b)+b*(v[1]-b))    "determines v[1]"
P[2]=R*T/(v[2]-b)-a/(v[2]*(v[2]+b)+b*(v[2]-b))    "determines v[2]"
P[3]=R*T/(v[3]-b)-a/(v[3]*(v[3]+b)+b*(v[3]-b))    "determines v[3]"
```

The compression work per unit mass required to go from state 1 to state 2 is given by the integral:

$$\frac{W_{12}}{m} = \int_{v_1}^{v_2} P\, dv \tag{10-7}$$

where the integrand, P, is evaluated at T = 350 K and the integration variable is v. The integral in Eq. (10-7) is evaluated using the Integral command.

```
P=R*T/(v-b)-a/(v*(v+b)+b*(v-b))    "determines integrand"
W_12=Integral(P,v,v[1],v[2])       "compression work to go from state 1 to 2"
```

The work per unit mass required for the compression process is W_{12}/m = -258,370 J/kg; the negative sign indicates that work is required for this process.

Next, we wish to determine the additional work that is required to compress the fluid from P_2 to P_3 isothermally at T.

$$\frac{W_{23}}{m} = \int_{v_2}^{v_3} P\, dv \tag{10-8}$$

It is tempting to solve Eq. (10-8) using another numerical integration, this time with limits v_2 and v_3:

```
W_23=integral(P,v,v[2],v[3])                    "compression work to go from state 2 to 3"
```

Trying to solve these equations will result in the error message shown in Figure 10-7. The equation set fails to solve because the two Integral statements use the same integration variable (v) with different limits.

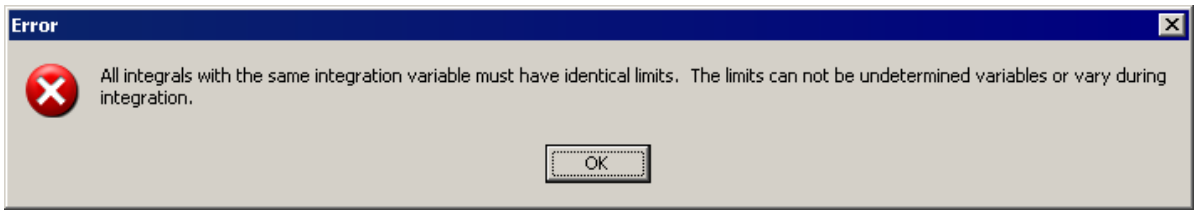


Figure 10-7: Error message that results when two integrals with the same integration variable are evaluated using different limits.

A solution to this problem could be obtained by introducing a second integration variable (v_x) and relating a second integrand (P_x) to v_x using the Peng-Robinson equation:

```
Px=R*T/(vx-b)-a/(vx*(vx+b)+b*(vx-b))            "determines integrand"
W_23=integral(P,vx,v[2],v[3])                   "compression work to go from state 2 to 3"
```

which leads to W_{23} = -32,515 J/kg.

This problem could be solved more elegantly using subprograms. A subprogram named PR is developed in order to solve the Peng-Robinson equation of state with specific volume and temperature (v and T) provided as inputs and pressure (P) returned as the output:

```
Subprogram PR(v,T: P)
    R=188.9 [J/kg-K]                            "gas constant for fluid"
    a=70.89 [N-m^4/kg^2]                        "Peng-Robinson parameter a"
    b=0.0006059 [m^3/kg]                        "Peng-Robinson parameter b"
    P=R*T/(v-b)-a/(v*(v+b)+b*(v-b))             "Peng-Robinson equation of state"
end
```

A second subprogram named W_c calculates the work per unit mass associated with the isothermal compression process. The inputs to the subprogram W_c are the temperature, initial pressure, and final pressure (T, P_i, and P_f). The output is the work per unit mass (W).

```
Subprogram W_c(T, P_i, P_f: W)
```

The specific volumes at the beginning and end of the compression process are computed by calling the Subprogram PR.

```
    call PR(v_i,T: P_i)                         "determine v_i"
    call PR(v_f,T: P_f)                         "determine v_f"
```

The compression work per unit mass required to go the initial to the final state is given by the integral in Eq. (10-7) where the integrand, P, is evaluated at T and the integration variable v using the subprogram PR. The integral is evaluated using the Integral command.

```
    call PR(v,T: P)                    "determine pressure at given v, T"
    W=integral(P,v,v_i,v_f)            "compression work"
end
```

The inputs are entered in the main program:

```
"Inputs"
T=350 [K]                              "temperature during compression"
P[1]=1e5 [Pa]                          "pressure at state 1"
P[2]=5e6 [Pa]                          "pressure at state 2"
P[3]=1e7 [Pa]                          "pressure at state 3"
```

The work per unit mass associated with the compression from states 1 to 2 (W_{12}) and from states 2 to 3 (W_{23}) are both obtained using the subprogram W_c.

```
call W_c(T,P[1], P[2]: W_12)           "compression work from P[1] to P[2]"
call W_c(T,P[2], P[3]: W_23)           "compression work from P[2] to P[3]"
```

When you attempt to solve these equations, you will likely receive the error message in Figure 10-8.

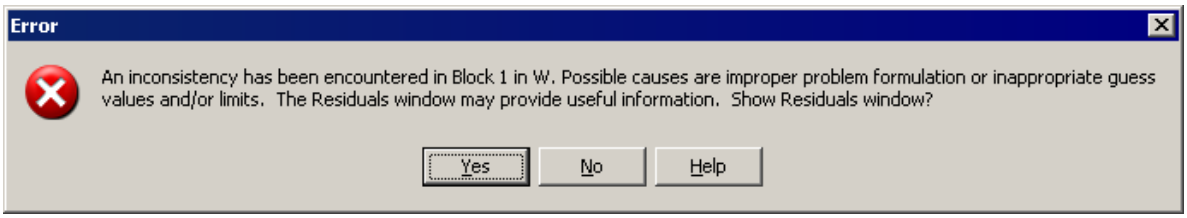
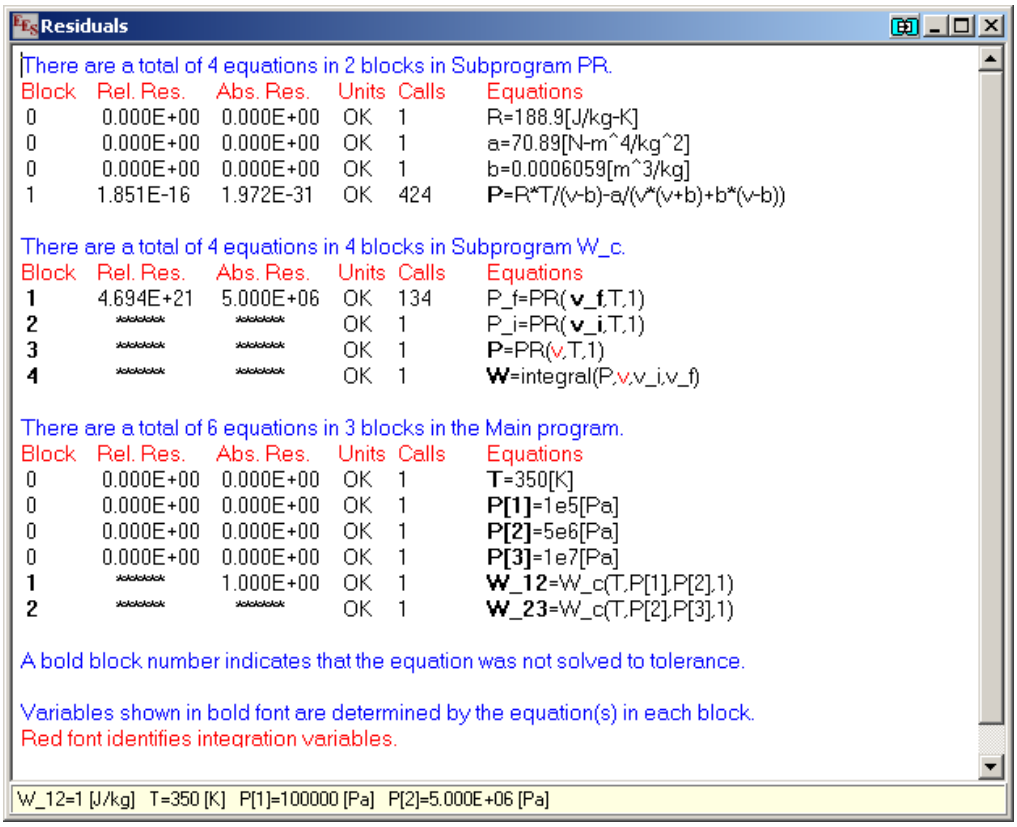
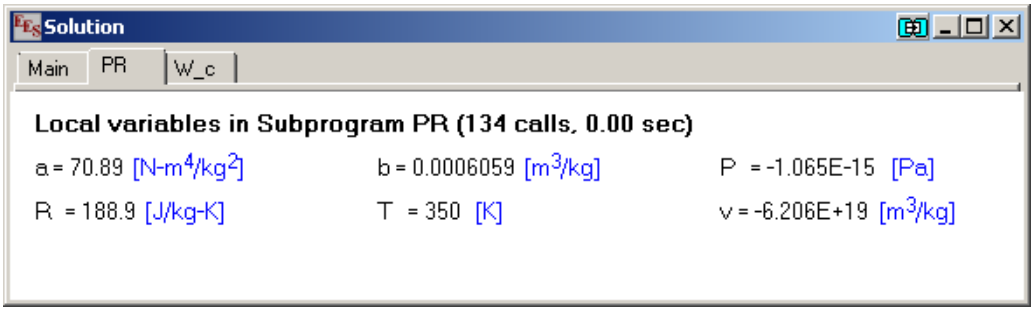


Figure 10-8: Error message when solving without first entering limits for specific volume.

It is necessary to specify guess values and/or limits on some of the variables in order to solve this problem. The Residuals Window is shown in Figure 10-9(a) and the Solution Window for the subprogram PR is shown in Figure 10-9(b).



(a)



(b)

Figure 10-9: (a) Residuals Window and (b) Solution Window for the subprogram PR.

Notice that the Residuals Window shows that the pressure in subprogram PR is not solved to within tolerance and the Solution Window shows that the specific volume in the last call to the PR subprogram is a large negative number. Clearly this is not a physically possible solution. It is necessary to place a lower limit of zero on variable v in the subprogram PR. Select Variable Info from the Options menu and then use the drop-down list in the top center of the dialog to select the variable information for subprogram PR, as shown in Figure 10-10.

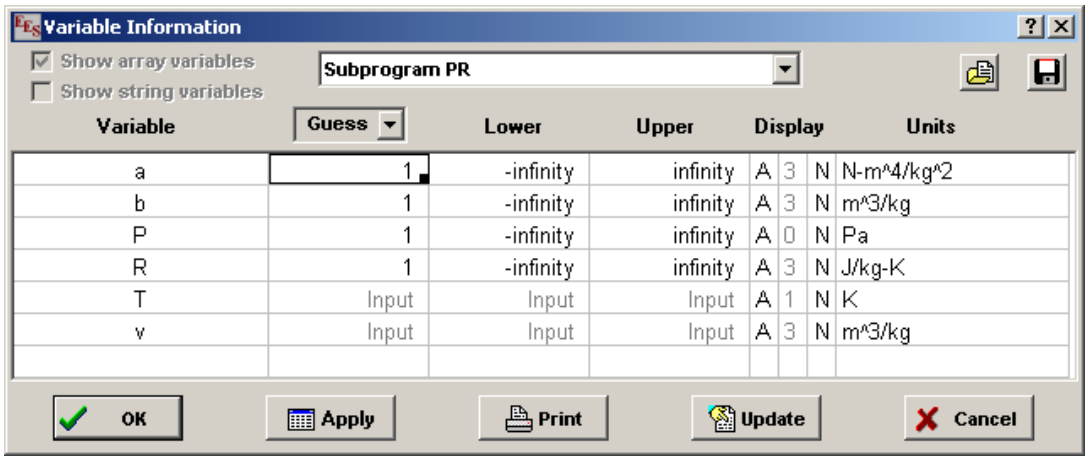


Figure 10-10: Variable Information dialog for the subprogram PR.

The Variable Information Window shows that the variable v is an input to subprogram PR. Therefore, we cannot set the guess value or limits for this variable because they are controlled by the calling program. The calling program in this case is subprogram W_c. Use the drop-down list to display the variable information for subprogram W_c, as shown in Figure 10-11.

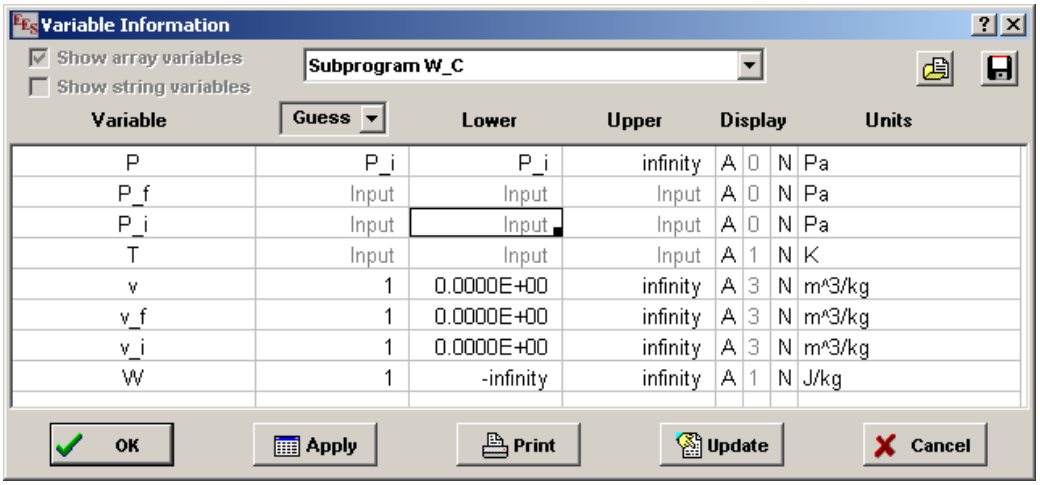


Figure 10-11: Variable Information dialog for the subprogram W_c.

Notice in Figure 10-11 that the variables v, v_1, and v_2 are not inputs and therefore it is possible to adjust their guess values and bounds. Enter lower bounds of zero for the variables v, v_1 and v_2. Although it is unnecessary, we can also enter a better guess value and lower bound for the variable P. The lowest possible value for P is P_i so this value is used as the guess value and lower limit. Click the OK button and then solve the equations. Both integrals will be now be evaluated without problem.

Subprograms to Determine the Limit of an Integral

Subprograms can be used to determine the limit of an integral evaluated using the Integral command that is required to obtain a certain integrated result. As a very simple example, suppose you wish to determine the upper limit (*a*) for the integral in Eq. (10-9) such that:

$$\int_0^a Y^2 dY = 33 \tag{10-9}$$

Because EES solves implicit equations iteratively, it would seem like Eq. (10-9) should be relatively easy to solve directly using the equation:

```
Integral(Y^2,Y,0,a)=33                    "attempting to directly solve for the limit of an integral"
```

However, solving will lead to the error message shown in Figure 10-12.

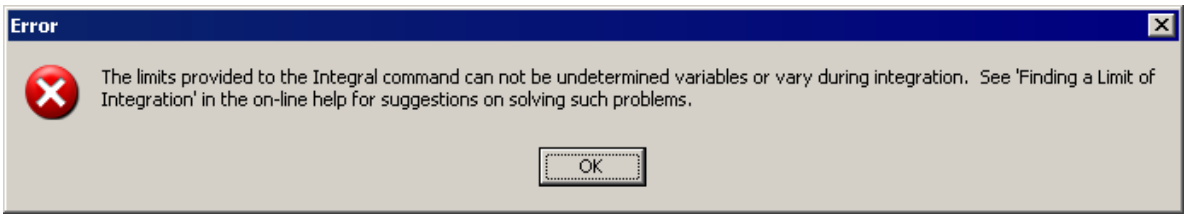


Figure 10-12: Error message associated with attempting to solve for the limits of an integral.

EES is not able to determine the derivative of the residual of the Integral command with respect to variable a, and therefore it cannot solve this problem numerically.

It is possible to determine the limit of the integral using a subprogram. A subprogram named Limit is developed that takes as its input the value of a and provides as its output the integrated value, X:

$$\int_0^a Y^2 dY = X \tag{10-10}$$

```
Subprogram Limit(a: X)
    Integral(Y^2,Y,0,a)=X
End
```

Subprogram Limit is called from the main EES program by specifying the output (X = 33) and allowing EES to iterate in order to identify the associated value of the input variable a.

```
Call Limit(a: 33)
```

When the Solve command is applied, EES will iteratively call subprogram Limit with different values of a until the value of a that results in an integrated value of X = 33 is found. The value identified by EES, a = 4.626, agrees with the analytical solution of $\sqrt[3]{99}$.

Note that the calculation would fail to solve if the order of the arguments in subprogram Limit was reversed. As shown above, a is an input to subprogram Limit and its value is adjusted in the calling program. If X were the input, then the variable a would be considered an output and the subprogram Limit would then try to adjust its value in order to solve the equation set; this situation would result in the same error message that is shown in Figure 10-12.

10.2 Modules

A module serves the same purpose as a subprogram. Both code segments provide a means to compartmentalize a set of equations within an EES program and both use equations rather than assignment statements. The equations can be entered in any order in implicit form, just as they can within the main section of an EES program. Because they use equations, neither subprograms nor modules can directly use logic statements, such as If-Then-Else or Repeat-Until constructs. Subprograms and modules are both accessed using the Call statement.

Module Execution

The difference between subprograms and modules is subtle and is related to how their equations are solved. The solution process will be demonstrated with the simple program shown below. Note that this is exactly the same program that was used to demonstrate subprograms in Section 10.1, except that the Subprogram keyword has been changed to Module.

```
Module DOF(X: Y, Z)
    X=2*Y+3*Z
    X^2=Y^2+Z^2
end

A=5
Call DOF(A: B, C)

B2=6
Call DOF(A2: B2, C2)
```

As noted in Section 10.1, when a Call statement for a subprogram is executed, EES clears its variable space, loads the subprogram and then solves the equation set contained in the subprogram. When the calculations in the subprogram are completed, EES will reload all of the information that was in memory before the subprogram was called and proceed with the solution process.

The process is very different when a module is called. EES compiles all of the equations in the Equations Window before it attempts to solve them. When EES encounters a Call statement for a module, it transparently grafts the equation set associated with the module into the equations in the main EES program. The steps necessary for this process are as follows. First, every variable in the module, including each input and output variable in the module statement, is renamed with a unique qualifier that EES can recognize. In the example above, the variables within the first call to module DOF are renamed DOF\1.X, DOF\1.Y, and DOF\1.Z. Then EES adds one equation for each input that sets the value of the input parameter in the calling program to the value in the module. In the example above, this process is equivalent to writing:

```
DOF\1.X = A
```

for the first call to the module. All of the equations in the module, with their renamed variables, are merged with the equations in the main EES program. For this example, the first call to the module will result in:

```
DOF\1.X=2*DOF\1.Y+3*DOF\1.Z
DOF\1.X^2=DOF\1.Y^2+DOF\1.Z^2
```

Finally, EES adds one equation for each output that sets the value of the output variable in the module to the corresponding EES variable in the argument list.

```
DOF\1.Y = B
DOF\1.Z = C
```

If the module is called a second time then the process described above is repeated, but with a different qualifier for the variable names in the module. The second call to the module in this example will lead, internally, to:

```
DOF\2.X = A2
DOF\2.X=2*DOF\2.Y+3*DOF\2.Z
DOF\2.X^2=DOF\2.Y^2+DOF\2.Z^2
DOF\2.Y = B2
DOF\2.Z = C2
```

The net effect is that a copy of all of the equations in the module is merged into the main EES program each time a Call statement to a module is encountered. Solving this example is therefore equivalent to solving:

```
A=5
DOF\1.X = A
DOF\1.X=2*DOF\1.Y+3*DOF\1.Z
DOF\1.X^2=DOF\1.Y^2+DOF\1.Z^2
DOF\1.Y = B
DOF\1.Z = C

B2=6
DOF\2.X = A2
DOF\2.X=2*DOF\2.Y+3*DOF\2.Z
DOF\2.X^2=DOF\2.Y^2+DOF\2.Z^2
DOF\2.Y = B2
DOF\2.Z = C2
```

EES currently allows up to 6,000 equations (12,000 in the Professional version) so rather large problems can be developed in this manner. EES then uses its efficient blocking techniques to reorganize all of the equations for an optimal solution.

The Solution Window

The Solution Window that results after solving the module example problem is shown in Figure 10-13. As for a subprogram, a separate tab in the Solution Window is provided for each module. However, unlike a subprogram, local results are provided in the Solution Window for each Call to the module. Module DOF was called twice in the example program and therefore the results for both calls appear in Figure 10-13.

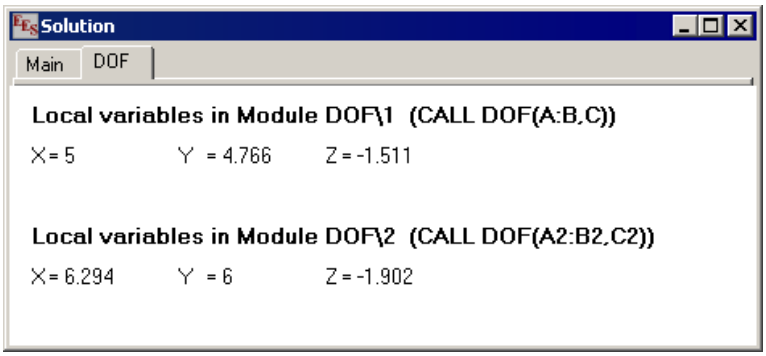


Figure 10-13: Solution window showing the local results for both calls to the module DOF.

The Residuals Window

Since the equations in a module are integrated with the equations in the main program and subsequently blocked, the equations in the module may not necessarily be called in the sequence that they were written. In fact, the equations are almost always re-ordered internally by EES in order to accomplish the solution most efficiently. This is one reason that the convergence properties of a module differ from those of a subprogram. The subprogram equations must necessarily be solved all at one time whereas the equations in a module can be distributed as needed with the other equations in the main EES program. You can view the equations in the order that EES has rearranged them in the Residuals Window. The Residuals Window for this example appears in Figure 10-14.

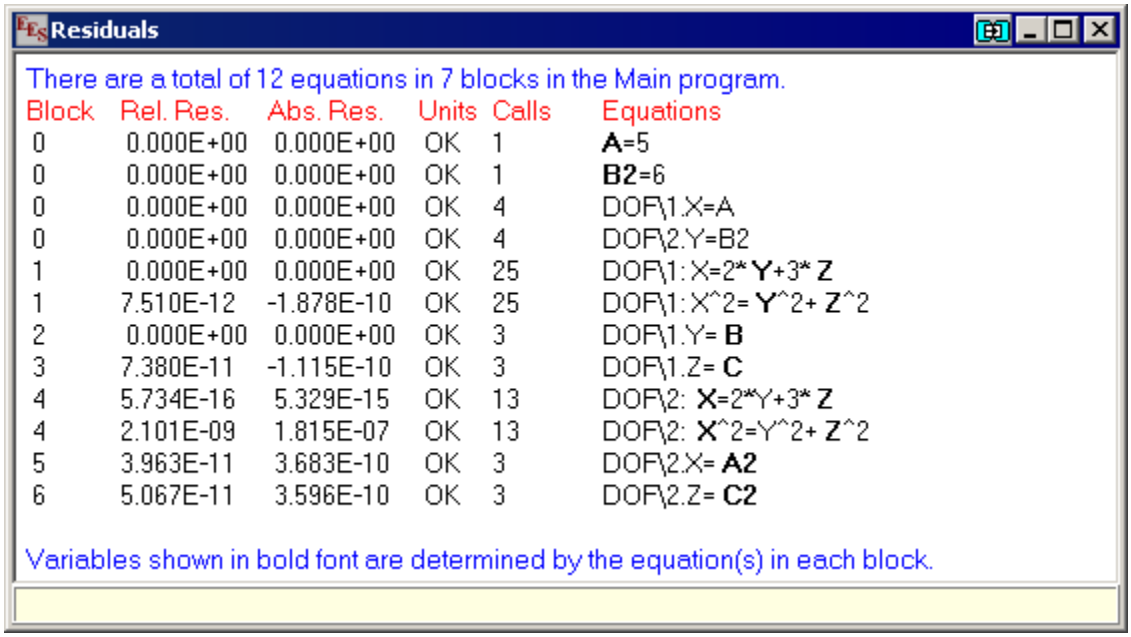


Figure 10-14: Residuals Window showing the order in which the equations are called.

Figure 10-14 shows how the module statements are included with equations in the Main program and blocked together with these equations. Equations from a module are identified with the module name followed by a backslash and then the call index number. For example, the equation:

```
DOF\1: X=2* Y+3* Z
```

in Block 1 originated from the first call to module DOF and it was used to determine the variables DOF\1.Y and DOF\1.Z. The equation:

```
DOF\2: X=2*Y+3* Z
```

in Block 4 originated from the second call to module DOF and it was used to determine the variables DOF\2.X and DOF\2.Z.

Guess Values and Limits of Inputs and Outputs

As for subprograms, the guess value and limits associated with the inputs to a module are controlled by the calling program. It is not possible to provide limits on the input variables for a module using the Variable Information dialog for the module. For example, the Variable Information dialog for the module DOF is shown in Figure 10-15; notice that it is not possible to provide limits or guess values for the variable X, which is an input.

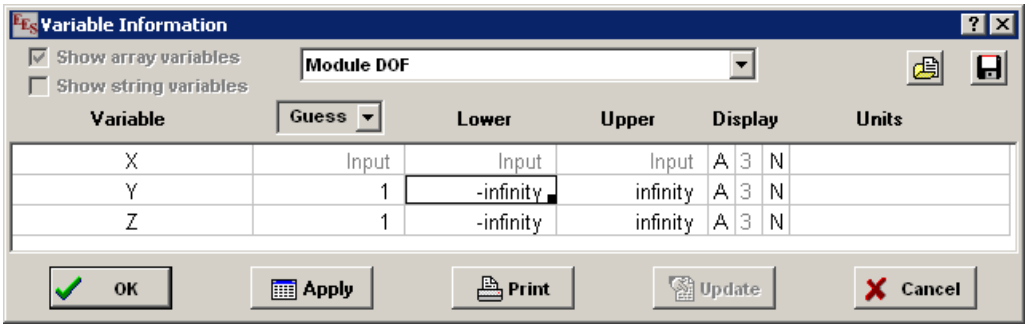


Figure 10-15: Variable Information dialog for the module DOF.

10.3 Should you use Subprograms or Modules?

The main difference between subprograms and modules is how they are implemented in the solution method. So should you use a module or a subprogram? In our experience, modules tends to be more robust and result in a more computationally efficient solution than subprograms. However, this result is problem specific. You may wish to test both subprograms and modules in order to determine which is better for your application. Other considerations that affect your choice are as follows.

1. A major advantage of a subprogram is that it can be called from a function or a procedure. Conditional statements, such as the If-Then-Else, GoTo, and Repeat-Until constructs are only supported in functions and procedures. Therefore, if the operation provided must be controlled using logic then it will likely be necessary to use a subprogram rather than a module.
2. Modules provide local information for each Call statement in the Solution Window and the Residuals Window; this information may be helpful, particularly for diagnostics.

3. If Arrays are used in a subprogram, they can be optionally displayed in the Arrays Table by placing a $Arrays On directive after the Subprogram declaration statement. Once they are available in the Arrays Table, array values can be plotted or saved to a file. However, only the array values in the last call to the subprogram are accessible in this way. The $Arrays On directive can also be used for modules. In this case, a separate tabbed Arrays Table is provided for every call to the module.
4. Modules necessarily add equations and variables to the Main body of an EES program. There is an upper limit of 6,000 variables in the Commercial version and 12,000 variables in the Professional version of EES. If you are solving a large program, the use of subprograms may be preferable to using modules. Since each Subprogram provides its own variable space, the variables used in a subprogram do not contribute to the count of variables in the Main program.
5. Subprograms can call other subprograms. However, a module cannot call another module. EES will not display an error in this case, but it will automatically modify the second module so that it operates as a subprogram.
6. Complex numbers, discussed in Chapter 13, can be used in a subprogram by placing a $Complex On directive within the subprogram. It is not necessary to have complex numbers enabled in the main program in order to use them in a subprogram. Complex numbers can not be used in modules.
7. Subprograms and modules can both be stored as library files, just like internal functions and procedures, as described in Chapter 11. However, it is recommended that subprograms, rather than modules, be used for library files. The equations in modules are integrated with those in the Main program and the combined set of equations may not always solve as expected, depending on the equations contained in the user's program. The subprogram equations are solved separately, providing some isolation from the main program.
8. Both subprograms and modules can significantly increase the capabilities of your EES program.

References

Klein, S.A. and Nellis, G.F., *Thermodynamics*, Cambridge University Press, New York, (2011).

11 INTERNAL LIBRARY FILES

Internal functions and procedures are EES code units that are composed of assignment statements rather than equations, as described in Chapter 3. The assignment statements used in functions and procedures are similar to those used in most programming languages. Subprograms and modules are callable code units that use equations, like those used in the main EES program, as described in Chapter 10. EES allows files containing one or more functions, procedures, and/or subprograms to be saved as library files with a .lib filename extension. These files can be loaded manually or automatically when EES is started. The code units in these files operate just like the built-in functions in EES and can therefore be accessed conveniently and transparently by the user. Online help files can be prepared in several formats and associated with the library files. The library file concept is among the most powerful features of EES because it allows the user to easily write customized and reusable code for personal use or for use by others. Library files can also be written in compiled languages, such as C++ or FORTRAN. These library files are called external library files and they are described in Chapter 19. This chapter also describes how a user-defined menu can be designed in order to allow access to EES programs from the menu bar.

11.1 Writing and Saving Library (.lib) Files

There are four types of EES code units that can appear in an EES program. Functions and procedures, described in Chapter 3, are composed of assignment statements. Their major advantage is that they allow logic statements, such as If-Then-Else and Repeat-Until constructs to be used. Subprograms and modules, described in Chapter 10, utilize equations and can be called from the main EES program or from other code units. All four types of code units can make an EES program more efficient and more understandable. Another major advantage of these code units is that they can be carefully written, debugged, and validated and then reused by your or others with confidence.

The key to making a code unit reusable is to save it in a manner in which it can later be used easily by other EES programs. Of course, it would be possible to simply open a previously saved EES program, copy the code unit of interest, and then paste it into another EES program. However, aside from being somewhat inconvenient, this manner of moving a code unit from one EES program to another would not copy the variable information, such as the guess values, limits, display format and units associated with each variable. A much better alternative is to save the EES file containing the code unit of interest as a library file having a .lib file name extension. Library files save the information in the Equations Window as well as any associated variable information.

An Example Library File Function

As an example, we will develop a function that returns the thermodynamic property called specific exergy. Specific exergy has the significance of being the maximum useful work per unit mass that can be obtained from a fluid that is not in equilibrium with the atmosphere (i.e., it is not at the "dead" state). Klein and Nellis (2011) show that specific exergy, *x*, is related to other thermodynamic properties as indicated by Eq. (11-1):

$$x = (u - u_o) - T_o(s - s_o) + P_o(v - v_o) + \frac{\tilde{V}^2}{2} + g\,z \tag{11-1}$$

where u is specific internal energy, s is specific entropy, v is specific volume, T is absolute temperature, P is pressure, $\tilde{V}$ is velocity (relative to the dead state), g is the gravitational acceleration, and z is the elevation relative to the dead state (i.e., the atmosphere). The subscript o in Eq. (11-1) indicates the value of the quantity at the dead state (e.g., T_o is the dead state temperature).

EES does not provide a built-in function for specific exergy. Therefore, we will write a general function that returns specific exergy as a function of temperature and pressure. To be useful, our function must work in any unit system and for any fluid in the EES database. The name of the function is Exergy and it is declared according to:

```
Function Exergy(F$, T, P, Vel, z,  T_o, P_o)
```

The inputs to the function include the fluid name (in the string F$), the temperature, pressure, velocity, and elevation of the fluid (T, P, Vel, and z), and the temperature and pressure of the dead state (T_o and P_o). The Exergy function will use the built-in property functions that are described in Chapter 4.

Units in Library File Functions

It is easy to write the function so that it operates in one assumed set of units. However, in order for it to be easily usable when called from an arbitrary EES program it must operate properly in any set of units. A strategy for doing this is to convert all of the values of the input variables to standard SI units, evaluate specific exergy using Eq. (11-1), and then convert the computed value of specific exergy back to the user's units. The units of the input arguments that are properties (i.e., temperature and pressure) can be determined using the built-in UnitSystem$ function, which returns a string that contains the units that EES is currently set to use for a specified EES property.

```
"Obtain the units of the inputs"
T$=UnitSystem$('Temperature')          "T$ is the temperature units EES is set to"
P$=UnitSystem$('Pressure')             "P$ is the pressure units EES is set to"
```

The strings T$ and P$ will contain the units for temperature and pressure that are consistent with EES' current unit system settings. The UnitSystem function is used to determine if the current unit system setting is SI or English and the units for length (L$) and velocity (Vel$) are set accordingly. The UnitSystem function takes a string argument corresponding to a unit system

setting and returns true (1) or false (0) depending on whether the argument is consistent with current unit system settings. For example, UnitSystem('Deg') will return true (1) if EES is currently configured to use degrees in its trigonometric functions and false (0) otherwise.

```
if (UnitSystem('SI')=1) then
    L$='m'                                  "L$ is the length units EES is set to"
    Vel$='m/s'                              "Vel$ is the velocity units EES is set to"
else
    L$='ft'
    Vel$='ft/s'
endif
```

The inputs are each converted to their base SI unit values using the Convert and ConvertTemp functions; note that string variables are used within these conversions.

```
"Convert inputs to base SI units"
T_SI=ConvertTemp(T$,K,T)                    "temperature in K"
T_o_SI=ConvertTemp(T$,K,T_o)                "dead state temperature in K"
P_SI=P*convert(P$,Pa)                       "pressure in Pa"
P_o_SI=P_o*convert(P$,Pa)                   "dead state pressure in Pa"
Vel_SI=Vel*convert(Vel$,m/s)                "velocity in m/s"
z_SI=z*convert(L$,m)                        "elevation in m"
```

The properties required to compute specific exergy include the specific internal energy, specific entropy, and specific volume of the fluid at the dead state. The specific internal energy is computed differently depending on whether the fluid is an ideal gas or not, as determined using the IsIdealGas function in EES. The specific internal energy of an ideal gas is a function of temperature alone. The specific internal energy of a real fluid is a function of temperature and another property such as pressure. Note that the units of the property returned by the IntEnergy function are consistent with the unit settings in EES; these units are obtained using the UnitSystem$ function and contained in the string u$. The specific internal energies are converted to base SI units.

```
"Compute properties and convert to SI units"
If (IsIdealGas(F$)) Then
    u_o=IntEnergy(F$,T=T_o)                 "ideal gas specific internal energy at dead state"
    u=IntEnergy(F$,T=T)                     "ideal gas specific internal energy"
Else
    u_o=IntEnergy(F$,T=T_o,P=P_o)           "specific internal energy at dead state"
    u=IntEnergy(F$,T=T,P=P)                 "specific internal energy"
EndIf
u$=UnitSystem$('Energy')                    "specific internal energy units that EES is set to"
u_SI=u*convert(u$,J/kg)                     "specific internal energy in J/kg"
u_o_SI=u_o*convert(u$,J/kg)                 "specific internal energy at dead state, in J/kg"
```

The same process is used to determine the specific entropies and specific volumes and they are converted to base SI units.

```
s_o=entropy(F$,T=T_o,P=P_o)                 "specific entropy at dead state conditions"
s=entropy(F$,T=T,P=P)                       "specific entropy"
s$=UnitSystem$('Entropy')                   "specific entropy units that EES is set to"
```

```
s_SI=s*convert(s$,J/kg-K)                    "specific entropy in J/kg-K"
s_o_SI=s_o*convert(s$,J/kg-K)                "specific entropy at dead state in J/kg-K"

v_o=volume(F$,T=T_o,P=P_o)                   "specific volume at dead state conditions"
v=volume(F$,T=T,P=P)                         "specific volume"
v$=UnitSystem$('Volume')                     "specific volume units that EES is set to"
v_SI=v*convert(v$,m^3/kg)                    "specific volume in m^3/kg"
v_o_SI=v_o*convert(v$,m^3/kg)                "specific volume at dead state in m^3/kg"
```

Equation (11-1) is used to compute the specific exergy in base SI units. The specific exergy is converted to units that are consistent with those set in EES.

```
g_SI=9.81 [m/s^2]                            "gravitational acceleration in SI units"
x_SI=u_SI-u_o_SI-T_o_SI*(s_SI - s_o_SI)+P_o_SI*(v_SI - v_o_SI)+Vel_SI^2/2+g_SI*z_SI
                                             "specific exergy in J/kg"
Exergy=x_SI*convert(J/kg,u$)                 "specific exergy in u$ units"
```

The units of each variable are set in the Variable Information dialog, as shown in Figure 11-1. Note that the string variables are used for the units that are consistent with the current EES settings. For example, the temperature units that EES is set to are provided in string variable T$, which was set according to T$=UnitSystem$('Temperature').

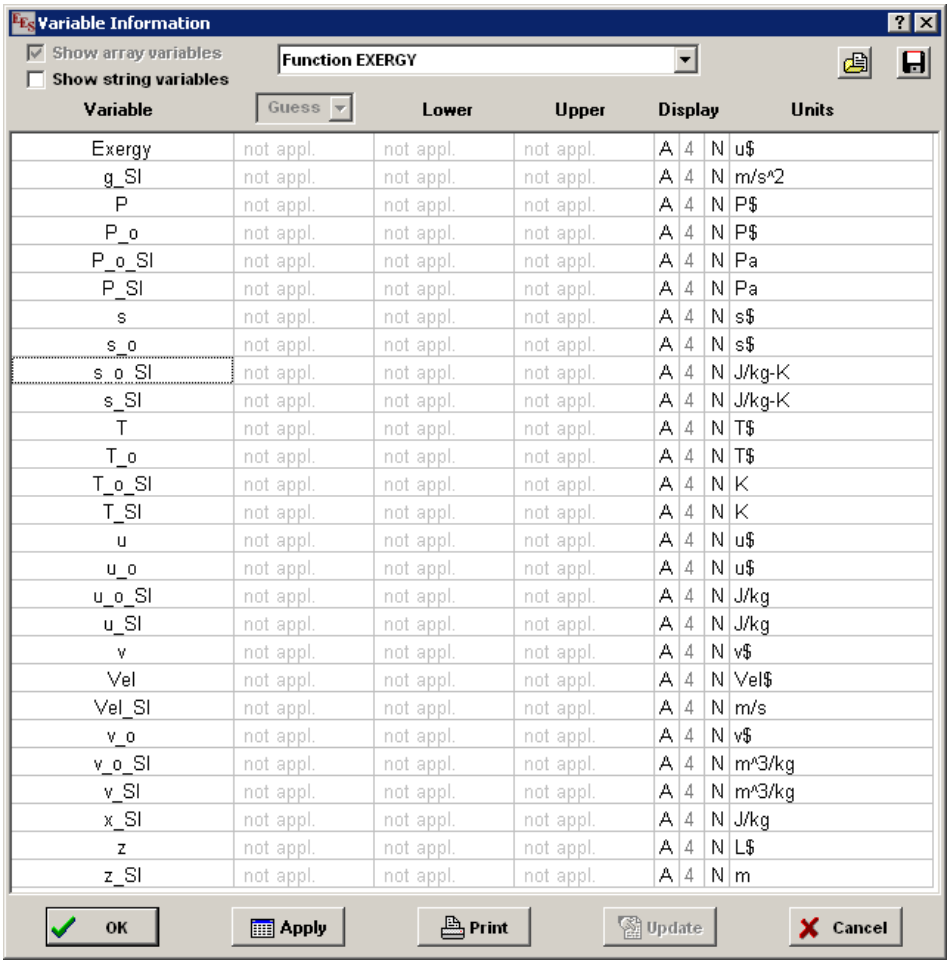


Figure 11-1: Variable Information dialog showing unit settings for the variables in the Exergy function.

Testing a Library File Function

A function should be thoroughly tested before it is saved as a library file. We can test the function Exergy by calling it for different fluids and using different unit systems, ensuring each time that the results are correct. The following equations test the function for water at temperature T = 400°C, pressure P = 100 kPa, velocity $\tilde{V}$ = 100 m/s, and elevation z = 4 m.

```
$UnitSystem SI Mass kJ C kPa
F$='Water'                                  "fluid name"
T=400 [C]                                   "temperature"
P=100 [kPa]                                 "pressure"
T_o=20 [C]                                  "dead state temperature"
P_o=100 [kPa]                               "dead state pressure"
Vel=100 [m/s]                               "velocity"
z=4 [m]                                     "elevation"
x = Exergy(F$, T, P, Vel, z,  T_o, P_o)     "specific exergy"
```

The result is x = 781.5 J/kg with no reported unit warnings. Change the unit system and the units of the variable in the main program to the English system:

```
$UnitSystem English Mass Btu F atm
F$='Water'                                  "fluid name"
T=ConvertTemp(C,F,400 [C])                  "temperature"
P=100 [kPa]*convert(kPa,atm)                "pressure"
T_o=ConvertTemp(C,F,20 [C])                 "dead state temperature"
P_o=100 [kPa]*convert(kPa,atm)              "dead state pressure"
Vel=100 [m/s]*convert(m/s,ft/s)             "velocity"
z=4 [m]*convert(m,ft)                       "elevation"
x = Exergy(F$, T, P, Vel, z,  T_o, P_o)     "specific exergy"
x_2=x*convert(Btu/lbm,kJ/kg)                "for comparison with previous result"
```

in order to obtain x = 336 Btu/lb_m which is consistent with 781.5 kJ/kg and again, there are no unit warnings. Additional tests for different fluids indicate that the function is working properly. Now it can be saved as a library file.

Saving Functions as a Library File

All of the equations in the main program should be commented out or deleted before saving the library file. Otherwise, they will be saved in the library file and reloaded into the main program when the library file is reopened, which is not a good idea. For this example, the equations that were used to test the function Exergy should be commented out:

```
{$UnitSystem English Mass Btu F atm
F$='Water'                                  "fluid name"
T=ConvertTemp(C,F,400 [C])                  "temperature"
P=100 [kPa]*convert(kPa,atm)                "pressure"
T_o=ConvertTemp(C,F,20 [C])                 "dead state temperature"
P_o=100 [kPa]*convert(kPa,atm)              "dead state pressure"
Vel=100 [m/s]*convert(m/s,ft/s)             "velocity"
z=4 [m]*convert(m,ft)                       "elevation"
x = Exergy(F$, T, P, Vel, z,  T_o, P_o)     "specific exergy"
```

```
x_2=x*convert(Btu/lbm,kJ/kg)                    "for comparison with previous result"}
```

This library file is saved with the name Exergy.lib using the Save As command in the File menu, as shown in Figure 11-2. The file can be saved in any directory. However, if the library file is saved in the Userlib folder within the EES directory, as shown, it will be automatically loaded when EES is started.

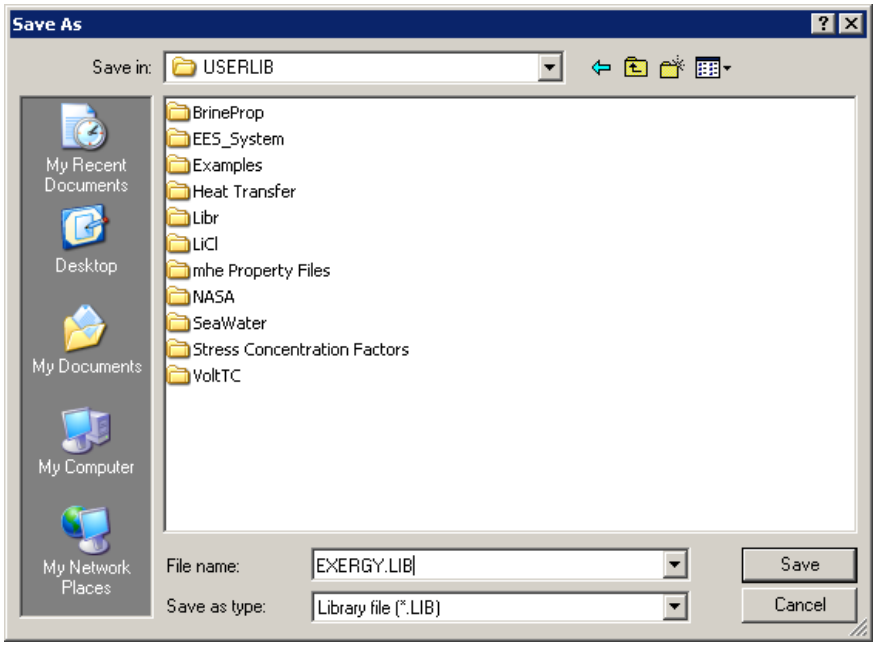


Figure 11-2: Use of the Save As command to save the Exergy.lib.

Note that, in general, one EES library file can contain any number of functions, procedures, or subprograms.

11.2 Loading and Using Library Files

A library file can be saved anywhere. If it is saved in the Userlib folder within the directory that EES is installed, the library file will be automatically loaded the next time EES is started and the functions, procedures, and subprograms within the library will be available for use in any new EES program. The contents of the library file will not be displayed in the Equations Window, but you can confirm that the library file has been loaded by using the Function Information dialog.

EES Library Routines in the Function Information Dialog

If you saved the Exergy.lib file in the Userlib folder, then close EES and restart it in order to load the library. Select Function Information from the Options menu and then click on the EES library routines button at the upper right. Scroll down the list and you should see a folder named Exergy.lib, as shown in Figure 11-3. Click on the folder and you will see all of the functions, procedures, and subprograms that are contained in this file. In this case, there is only one

function: Exergy. The fact that the function appears in this list indicates that it has been loaded and it is ready to be used.

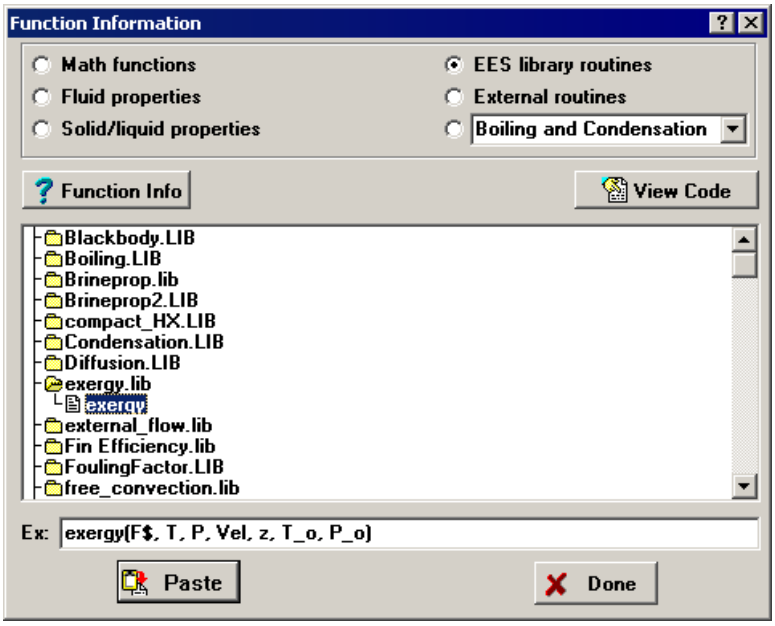


Figure 11-3: Function Information dialog showing that the function Exergy is loaded.

Enter the following equations into the empty Equations Window.

```
$UnitSystem SI C kPa kJ

F$='air'
T=76.85 [C]                          "temperature"
T_o=25 [C]                           "dead state temperature"
P=200 [kPa]                          "pressure"
P_o=Po#                              "dead state pressure"
Vel=44 [m/s]                         "velocity"
z=180 [m]                            "elevation"
x=exergy(F$,T, P, Vel, z, T_o, P_o)  "specific exergy for state"
```

Solve and you should see that x = 15.49 kJ/kg. The function Exergy can be used in the same way as any built-in function.

Libraries in the Userlib Folder

The contents of a typical Userlib folder are shown in Figure 11-4; your folder may differ somewhat from that shown, depending on what libraries are installed in your version of EES. Notice that the Exergy.lib file that we saved appears in this folder. We could have saved the file in a subfolder within the Userlib folder or within a nested folder. EES will find the library if it is saved anywhere within the Userlib folder and automatically load it at start up. Note that one of the subfolders in the Userlib folder is named EES_System. EES will automatically load the library files in the EES_System folder before it loads any other library files. Therefore, if you have developed a library file that is used by other library files you may want to put it in the EES_System folder.

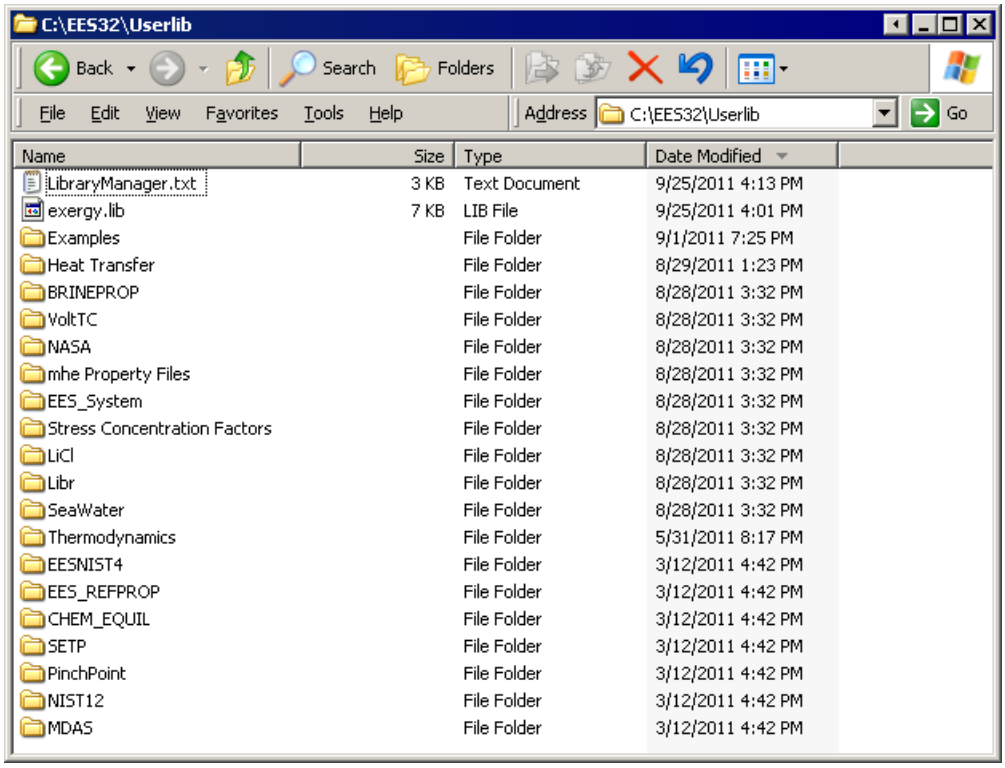


Figure 11-4: Contents of the Userlib folder.

Manually Loading Library Files

The disadvantage of saving a library file in the Userlib directory is that it is loaded every time you start EES, even for EES programs that do not use the library file. This is not really a major disadvantage, as EES loads library files quickly and most require very little memory. However, if the library is rarely used then you may wish to save it somewhere other than in the Userlib directory. In this case, the library file will need to be manually loaded before you use it.

There are two ways to manually load a library file. The direct way to load the library file is by selecting the Load Library command from the File menu. After selecting this command, specify the location of the library file and click the Open button. You can confirm that the library has been loaded by selecting the Function Information command from the Options menu, as illustrated in Figure 11-3.

The second way to load a library file is to use the $Include directive. The use of this and other directives is described in Chapter 14. For example, if the Exergy.lib file had been saved in the folder C:\temp\ then it would be loaded if the following statement appears in the EES Equations Window:

```
$Include C:\temp\Exergy.lib
```

It is not necessary to include the directory information if the library file is located in the directory that EES is installed in.

11.3 Help for Library Files

If a library file is to be useful to persons other than its developer, it is necessary to provide some documentation for the functions, procedures and subprograms contained in the file. If help information is available, it can be accessed using either the Function Info button that appears in the Function Information dialog or the Help menu item in the main menu bar titled Help for External Libraries. For example, if you click on the exergy item within the Exergy.lib folder shown in Figure 11-3, the Function Info button will be activated. No help information has thus far been entered for the Exergy function and therefore the help dialog appears as shown in Figure 11-5 when the Function Info button in the Function Information dialog is clicked.

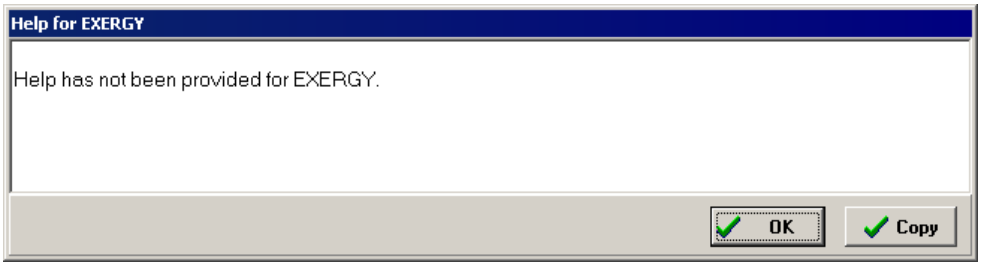


Figure 11-5: Result of clicking the Function Info button before help information has been provided.

Help Embedded within the Library

EES provides a number of ways to allow the developer of a library to provide help information. The simplest way to provide information is to embed it within the library file itself. Open the Exergy.lib file that was created in Section 11.1 using the Open command from the File menu. Set the filter box so that only files having a .lib file name extension are displayed. After the file is opened, enter the following text anywhere between the declaration line having the Function keyword and the termination line containing the End keyword. Note that the text is entered as a comment within braces and it is therefore ignored by EES during the processing of the equations. The first line must contain a $ followed by the name of the function, procedure, or subprogram on a line by itself, as shown below:

```
{$Exergy
Exergy(F$, T, P, Vel, z, T_o, P_o) returns the specific exergy of a fluid.
F$ is the name of the fluid
T is the temperature in EES temperature units
P is the pressure in EES pressure units
Vel is the velocity in m/s (SI) or ft/s (Eng)
z is the elevation in m (SI) or ft (Eng)
T_o is the dead state temperature in EES temperature units
P_o is the dead state pressure in EES pressure units
}
```

Now save the library file, close EES and restart it. If you saved the library file in the Userlib folder, it will automatically load. If not, load it with the Load Library command from the File menu or by using the $Include directive. Select the Function Information command from the Options menu and find the Exergy function, as shown in Figure 11-3. Select Exergy in the

Exergy.lib folder and click the Function Info button; you should see the help information appearing in Figure 11-6.

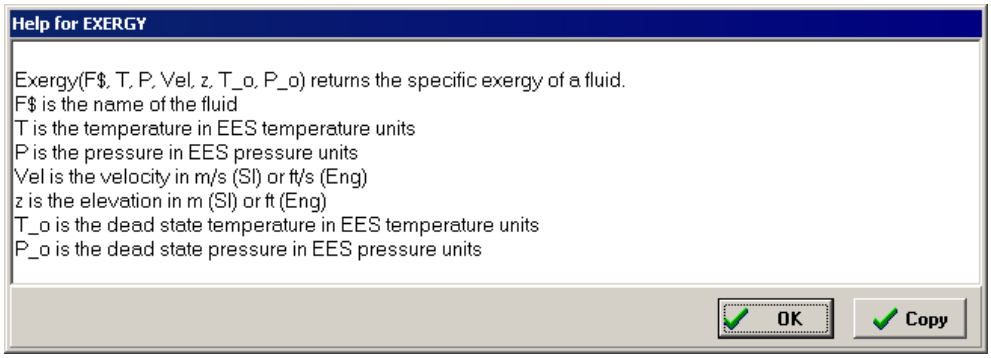


Figure 11-6: Function information provided in the Equations Window.

Using a Dedicated Help File

Embedding documentation in the library file is very easy, but it is also limited. It is not possible to incorporate fancy colors or fonts using this technique. Hyperlinks to other information cannot be included. Much more elaborate and effective help can be provided by creating a dedicated help file for the library file. The help file must have the same name as the library file (e.g., Exergy) but have a file name extension of .pdf, .htm, or .chm. Help files can be provided for both the internal library files discussed in this chapter (which have a .lib extension) or the external library files that are described in Chapter 19.

Help files having an .htm or .pdf format can be generated in several ways. Perhaps the easiest way is to use a word processor. For example, Microsoft Word allows a file to be saved with an .htm file name extension. A portable document format (.pdf) file can be obtained by printing a file from a word processor to a pdf printer, which is provided by Adobe Acrobat or other software. Figure 11-7 shows help information for the Exergy function entered into Microsoft Word, including a hypertext link to a reference. Save this file as Exergy.htm in the same directory as the Exergy.lib file. EES will load the Exergy.htm help file at the same time that the Exergy.lib file is loaded. The help information in the Exergy.htm file can be accessed from the Help for External Libraries command in the Help menu or by clicking the Function Info button in the Function Information dialog in Figure 11-3. Either method will start your browser program and load the help file, as shown in Figure 11-8. A .pdf file could be used in exactly the same manner.

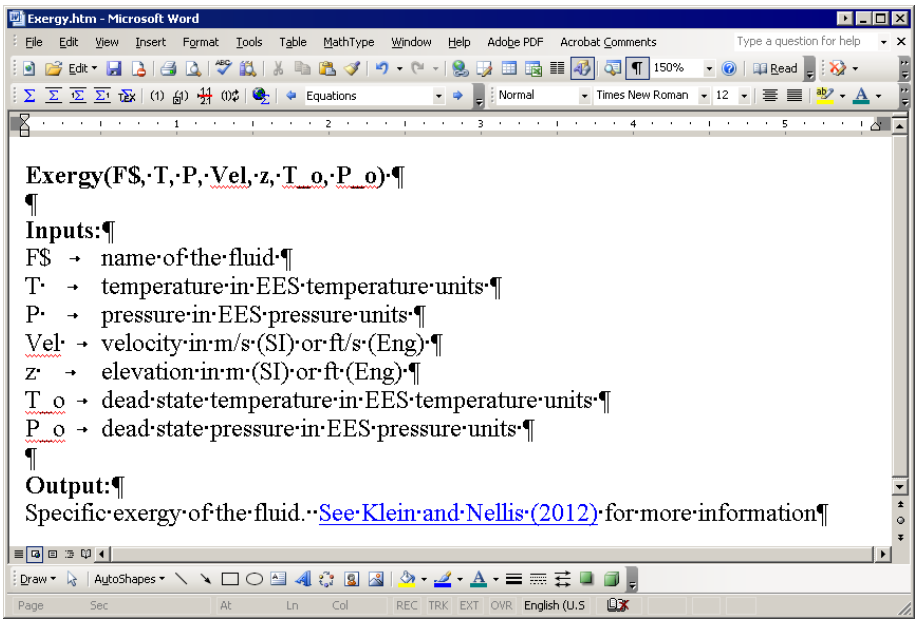


Figure 11-7: Help information for the Exergy function entered in Microsoft Word.

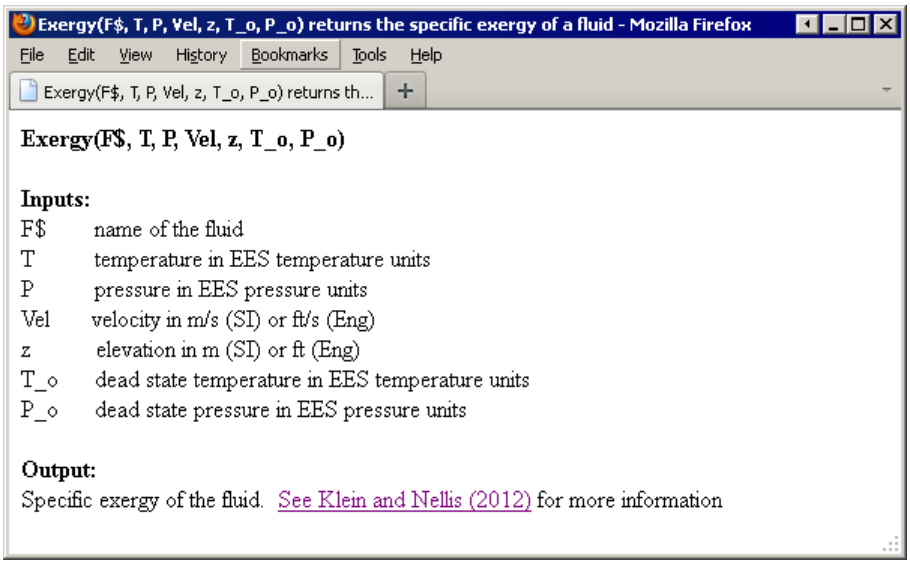


Figure 11-8: Browser program showing help information for the Exergy function.

A .chm file (compiled html) is a binary help file that contains hyperlinked contents. This is a common type of help file that is used to provide the online help for many programs in newer Microsoft Windows operating systems. (The .chm format has replaced the older .hlp help file format that was previously the default format for help files.) A help compiler program such as the HTML Help Workshop or Help Scribble (http://www.helpscribble.com/) can be used to build a .chm file. EES will recognize the .chm file if it has the same name as the library file and it will display the help information when the Help for External Libraries command is selected from the Help menu or the Function Info button in the Function Information dialog is activated.

11.4 Application Library Files

Application library files provide graphical and text information that is displayed in the Function Information dialog when one of the application radio buttons at the bottom of the top panel is selected. Application libraries provide a means of organizing a series of related libraries by associating a descriptive graphic, an example, and help for each function, procedure or subprogram that is contained in a library file. An example of the information in an application library is shown in Figure 11-9 for the Boiling and Condensation library that is provided as part of the Heat Transfer application libraries.

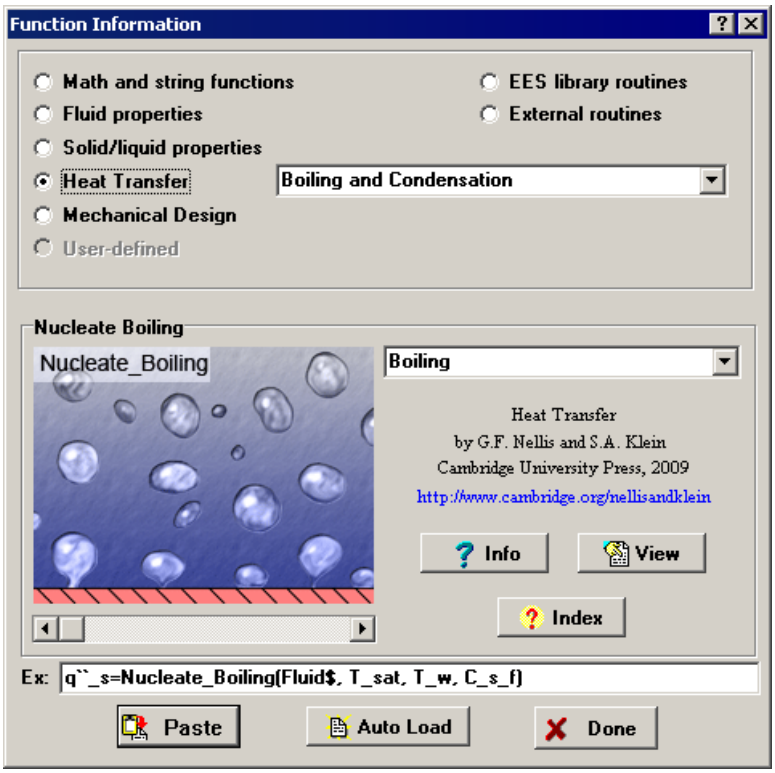


Figure 11-9: Application library information for the Boiling and Condensation library.

Contents of an Application Library Folder

The available application libraries are selected from the drop-down list to the right of the application radio buttons in the top panel (in Figure 11-9, the Boiling and Condensation library is selected from the Heat Transfer application radio button). Currently application libraries are available for Heat Transfer and Mechanical Design; there are folders associated with each of these topics in the UserLib folder. In addition, users can create their own User-Defined application libraries. Within each folder is a subfolder that is associated with each entry in the dropdown list. The contents of the associated subfolder must include:

- A table of contents (.toc) file that provides information about the organization and calling protocol of the library code units.
- One or more library (.lib) files that provide the functions, procedures, or subprograms.
- One or more .htm, .pdf, or .chm files that provide help for the code in the library file.

- One or more bitmap (.bmp) files that provide the figures that are shown in the Function Information dialog.

Table of Contents File

The required information and format for an application library can be explained by viewing the contents of the EES\Userlib\Heat Transfer\Boiling and Condensation folder shown in Figure 11-10 and the Boiling and Condensation.toc file that is contained in that folder and shown in Figure 11-11. Line numbers have been added in Figure 11-11 to simplify the following discussion. Note that each entry (i.e., each line number) in the table of contents file shown in Figure 11-11 exists on a separate line. However, some word-wrapping and indentation have been used to display lines that are too long to fit on the page.

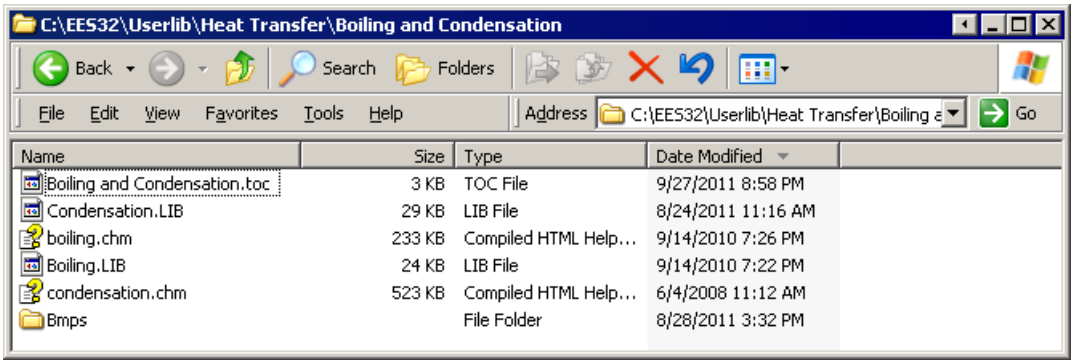


Figure 11-10: Contents of the Boiling and Condensation folder in the Heat Transfer library.

Line 1 of the .toc file provides the name of the library file; this name will appear in the drop down list next to the radio button in the bottom right of the top panel, as shown in Figure 11-9 (Boiling and Condensation). Line 2 provides reserved information. Currently this line should display -1 for the heat transfer application libraries, -2 is used for the mechanical design application libraries, and -3 for user-defined libraries. Lines 3-6 provide user-supplied information that is displayed to the right of the picture in Figure 11-9. Line 6 can contain a URL web link and if it does, clicking on the link will start the default browser and open it to the specified page. Following these six lines is the information that is required to provide the visual information in the Function Information dialog. Blank lines are ignored. Lines that begin with > indicate a new subheading; the subheadings are the choices that are provided in the drop-down list to the right of the picture. For example, lines 8, 15, and 25 are the subheading choices that appear in the drop-down list (Boiling, Condensation, and Pressure Drop).

Following each subheading line is a line that provides information for each code unit (i.e., function, procedure or subprogram) in that subgroup. The code unit is selected using the scroll bar that is located below the picture. The | character is used to separate the information items on each line.

```
1:  Boiling and Condensation
2:  -1
3:  Heat Transfer
4:  by G.F. Nellis and S.A. Klein
5:  Cambridge University Press, 2009
6:  http://www.cambridge.org/nellisandklein
7:
8:  >Boiling
9:  Nucleate Boiling|Boiling.LIB|Boiling.CHM@2010|BMPS\nucleate_boilings.bmp|
        q``_s=Nucleate_Boiling(Fluid$, T_sat, T_w, C_s_f)
10: Film Boiling|Boiling.LIB|Boiling.CHM@2510|BMPS\film_boiling.bmp|
        q``_s=Film_Boiling(Fluid$, Geom$, T_sat, T_s, D, epsilon)
11: Flow Boiling|Boiling.LIB|Boiling.CHM@3010|BMPS\flow_boiling.bmp|
        call Flow_Boiling(Fluid$, T_sat, G, d, x, q``, 'Horizontal': h, T_w)
12: Flow Boiling - Average|Boiling.LIB|Boiling.CHM@3020|BMPS\flow_boiling_avg.bmp|
        h_bar=Flow_Boiling_avg(Fluid$, T_sat, G, d, x_in, x_out, q``, 'Horizontal')
13: Critical Heat Flux|Boiling.LIB|Boiling.CHM@1010|BMPS\critical_heat_flux.bmp|
        q``_max=Critical_Heat_Flux(Fluid$,Geom$, L, T_sat)
14:
15: >Condensation
16: Inside a Horizontal Tube - Local|Condensation.LIB|Condensation.CHM@2010|BMPS\horizontal_tubes.bmp|
        Call Cond_HorizontalTube(Fluid$, m_dot,  x, T_sat, T_w, D : h_m, F$)
17: Inside a Horizontal Tube - Average|Condensation.LIB|Condensation.CHM@2070|BMPS\
        horizontal_tube_avgs.bmp|Call Cond_HorizontalTube_avg(Fluid$, m_dot, T_sat, T_w, D, x_1, x_2 : h_m)
18: Vertical Plate|Condensation.LIB|Condensation.CHM@2020|BMPS\vertical_plates.bmp|
        Call Cond_vertical_plate(Fluid$, L, W, T_w, T_sat :h_m, Re_L, q, m_dot)
19: Horizontal Cylinder|Condensation.LIB|Condensation.CHM@2030|BMPS\horizontal_cylinders.bmp|
        Call Cond_horizontal_Cylinder(Fluid$, T_sat, T_w, D:h_m, Nusselt_m)
20: N Horizontal Cylinders|Condensation.LIB|Condensation.CHM@2031|BMPS\horizontal_N_cylinders.bmp|
        Call Cond_horizontal_N_Cylinders(Fluid$, T_sat, T_w, D, N:h_m, Nusselt_m)
21: Horizontal Finned Tube|Condensation.LIB|Condensation.CHM@2040|BMPS\finned_tubes.bmp|
        Call Cond_finned_tube(Fluid$, d_r, d_o, t, p, T_w, T_sat, k_f:h_m)
22: Flat Horizontal Plate Facing Up|Condensation.LIB|Condensation.CHM@2050|BMPS\horizontal_ups.bmp|
        Call Cond_horizontal_up(Fluid$, L, T_w, T_sat:h_m, Nusselt_m)
23: Flat Horizontal Plate Facing Down|Condensation.LIB|Condensation.CHM@2060|BMPS\
        horizontal_downs.bmp|Call Cond_horizontal_down(Fluid$, T_w, T_sat: h_m, Nusselt_m)
24:
25: >Pressure Drop
26: 2-Phase Pressure Drop|Boiling.LIB|Boiling.CHM@4010|BMPS\
        DELTAP_2phase_horiz.bmp|DELTAP=DELTAP_2phase_horiz(Fluid$, G, P_i, d, L, x_in, x_out)
```

Figure 11-11: Listing of the Boiling and Condensation.toc file.

The required information on each of the separate lines corresponding to the code units is as follows:

1. The name of the code unit (e.g., Nucleate Boiling in Figure 11-11; this is the name that is displayed above the picture).
2. The name of the library file that the code unit is located in (e.g., Boiling.lib in Figure 11-11). This library file must be contained in the same folder as the .toc file.
3. The name of the help file that is used to document the code unit (e.g., Boiling.chm@2010 in Figure 11-11). This file can be a .pdf, .htm or .chm file. If a .chm file is used then the item context number may be included by following the filename with the @ symbol and the help

file context number (e.g., @2010). The item context allows the help file to be opened to a particular page when the help file includes several pages.

4. The name of the bitmap file that holds a picture representative of the selected code unit (e.g., BMPS\nucleate_boilings.bmp – note that the BMPS\ indicates that the bitmaps are contained in a subfolder named BMPS). The bitmap file has a maximum width of 212 pixels and a maximum height of 160 pixels. Bitmaps will be scaled with a constant aspect ratio if they are larger in either of these dimensions.
5. An example that appears in the Example text box at the bottom of the Function Information dialog for the selected code unit (e.g., q``_s=Nucleate_Boiling(Fluid$, T_sat, T_w, C_s_f) in Figure 11-11). This is the text that will be pasted into the Equations Window if the Paste button is selected.

The Boiling and Condensation.toc file shown in Figure 11-11 is an example of a library that already exists and is included in the Heat Transfer application library. It is possible to add your own libraries to the Heat Transfer or Mechanical Design libraries or add libraries to the third, user-defined selection in Figure 11-9 by developing .toc files and changing the value in the second line to -3.

11.5 The Textbook Menu

The Textbook menu is a user-defined menu that can be added to the standard EES menu bar in order to allow easy access to existing sets of EES files. The Textbook menu has been used to provide a convenient means to access EES problems associated with a textbook, which explains its name. However, the Textbook menu can be used to organize any set of EES files. The information required to create a Textbook menu is contained in a textbook index file. The Textbook menu can be displayed either by opening a textbook index file with the Load Textbook command from the File Menu or by placing the textbook index file in the Userlib subdirectory, in which case the menu will be created automatically when EES is started.

A textbook index file is a text file that is identified with the filename extension (.txb). When EES reads a textbook index file, it creates a menu at the far right of the menu bar, as shown in Figure 11-12. Each menu item opens a dialog window from which one or more EES programs can be selected. The menu thus provides easy access to a large number of EES programs and is useful for organizing programs that are related to a specific topic. This menu is used by default to provide easy access to example problems for a text book; however, it can be configured by a user for any purpose.

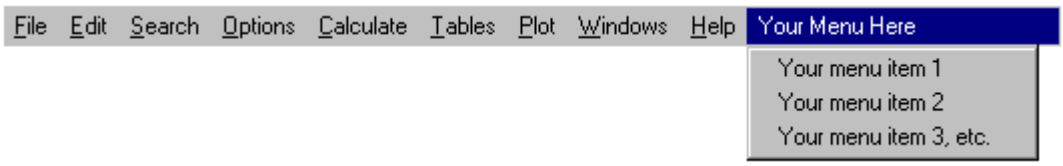


Figure 11-12: User-created Textbook menu, is placed at the far right of the menu bar.

The format of the textbook index file is quite simple. The menu shown in Figure 11-12 was created with the textbook index file that is shown in Figure 11-13. Note that the line numbers have been added for reference, but they are not part of the information in the file.

```
1:    Your Menu Here
2:    1
3:    General information line 1
4:    General information line 2
5:    General information line 3
6:    Reserved
7:    >Your menu item 1
8:    Descriptive problem name1 | FileName1.EES | HelpFile1.HLP | NO.BMP
9:    Descriptive problem name2 | FileName2.EES | HelpFile2.HLP | NO.BMP
10:   Descriptive problem name3 | FileName3.EES | HelpFile3.HLP | NO.BMP
11:   >Your menu item 2
12:   Descriptive problem name4 | FileName4.EES | HelpFile4.HLP | NO.BMP
13:   Descriptive problem name5 | FileName5.EES | HelpFile5.HLP | NO.BMP
14:   Descriptive problem name6 | FileName6.EES | NO.HLP | NO.BMP
15:   >Your menu item 3, etc.
16:   etc ….
```

Figure 11-13: Listing of the .txb file used to generate the Textbook menu shown in Figure 11-12.

Line 1 of the .txb file provides the menu title, which will appear in the menu bar to the right of the Help menu. Line 2 is reserved for a version number used internally by EES. Enter a 1 on this line. Lines 3, 4, and 5 provide user-supplied information that will be displayed in the dialog that appears whenever one of the menu items is selected. The information on line 3 is considered to be a title and it is displayed in bold red font, as shown in Figure 11-14.

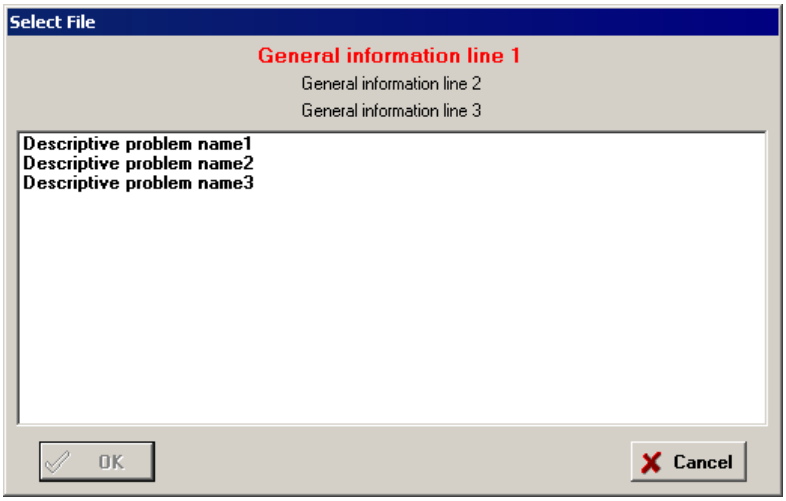


Figure 11-14: Dialog that appears when the first menu item in the textbook menu is selected.

Line 6 is reserved for future use. Enter the word reserved on this line. The remaining lines provide a set of information in groups. Each group begins with a menu item name preceded by the > character (e.g., line 7 in Figure 11-13) and then one or more problem descriptions. The menu item name is the name that will appear in the Textbook menu. The characters >- together will place a break line in the menu. Note that a second level of menu nesting is supported. If a

menu item name is preceded with >> then this menu item will become a 'flyout' menu that is displayed when the parent menu item is chosen.

Each problem description line contains four pieces of information that are separated by the | character. The first item is a descriptive name for the problem, which may be up to 128 characters. This is the name that will appear in the dialog when the menu item is chosen, as seen in Figure 11-14. The second item is the filename corresponding to the EES program file that will be opened if the problem is selected. This filename may be partially or fully qualified with directory information, e.g., C:\myBook\Chapter1\Problem1.ees or ..\Chapter1\Problem1.ees. The partial qualification is relative to the directory that EES is installed in. The third item is an optional help file that is to be associated with the EES file. The help file can be an .htm, .pdf, or .chm file. If no help file name is provided, you should enter no.hlp. If help is provided, then the help file can be accessed from an additional menu item that is placed in the Help menu. The final item was intended to be the file name for a figure associated with the EES file. However, EES does not currently use the filename, so enter no.bmp as a placeholder. The textbook index file should be placed in a sub-directory that includes all of the referenced EES programs and associated help files.

EES currently uses the Textbook menu to display examples in the default installation. The Examples menu provides convenient access to a number of instructive EES programs that may be of use to a new user. You can remove this menu by deleting or renaming the Examples.txb file; any other existing .txb file in the Userlib folder can be deleted as well.

11.6 The Library Manager

Normally, all files placed in the EES Userlib folder are automatically loaded when EES is started. These files can include internal library files with the file name extension .lib. Library files can also be written in compiled languages such as FORTRAN, C, and Pascal, as described in Chapter 19. These external library files can have file name extensions of .fdl, .dlf, .dlp, or .dll, and they are also loaded at startup if they are placed in the Userlib folder. Textbook menu files having a .txb filename extension are also automatically loaded, but only one .txb file can be active at a time. The user can control which files are loaded at startup by moving files into and out of the Userlib folder. However, a more convenient alternative is to use the Library Manager which is provided in the Professional version of EES.

The Library Manager is accessed by clicking the Auto Load button at the bottom of the Function Information dialog when the EES library routines, External Routines, or an application library button is selected. For example, notice the Auto Load button in Figure 11-9. Selecting the Auto Load button will bring up a dialog window having the appearance shown in Figure 11-15. Each file in the Userlib folder (or its subfolders) appears in the list. The check box to the left of the list controls whether or not the file is automatically loaded when EES is started. If the checkbox is gray then the function is located in the EES_System directory and therefore its load status cannot be changed in the Library Manager. Changing the loading status of any file has an immediate effect. If a file is not currently loaded, clicking the check box will automatically load the file and also ensure that it is loaded the next time EES is started. Un-checking a box will

remove a loaded file from memory and prevent it from loading the next time EES is started. Information is written by the Library Manager into the file LibraryManager.txt in the Userlib folder. Deleting this file will cause all files to be automatically loaded.

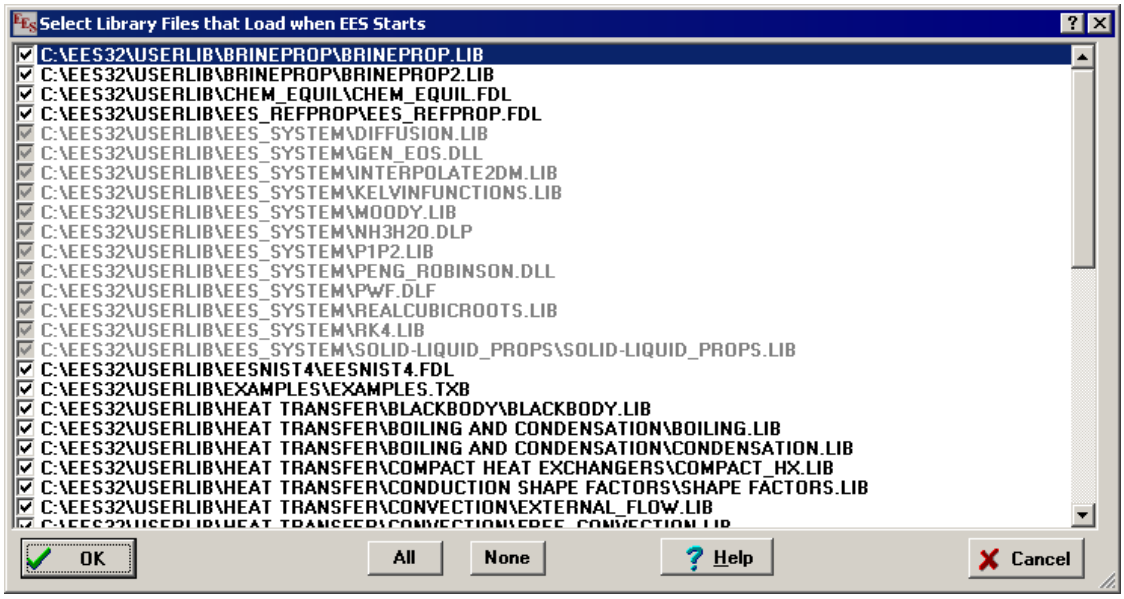


Figure 11-15: Library Manager dialog window.

References

Klein, S.A. and Nellis, G.F., *Thermodynamics*, Cambridge University Press, New York, (2011).

Nellis, G.F. and S.A. Klein, *Heat Transfer*, Cambridge University Press, New York, (2009).

12 THE HEAT TRANSFER LIBRARY

The heat transfer library in EES was developed during the writing of *Heat Transfer* by Nellis and Klein (2009). The objective of the library is to automate the reference materials that are commonly used to solve heat transfer problems so that it is not necessary to obtain information from figures, tables or equations. The heat transfer library includes functions that are capable of calculating quantities that are frequently required when solving heat transfer problems. Examples include view factors, heat exchanger effectiveness-*NTU* solutions, convection coefficients, friction factors, conduction shape factors and related information. The functions are easy to navigate through as they are organized into an application library, as discussed in Section 11.4. The various heat transfer library categories are discussed in this chapter. Note that the categories and functions may change as new features are continuously being added. One of the advantages of the heat transfer library is that it is easy to add functions or change existing functions. For more information on the underlying theory that is used to develop the heat transfer library please see Nellis and Klein (2009).

12.1 Boiling and Condensation

The Boiling and Condensation library contains functions that are used to compute heat transfer and pressure drop for liquid-vapor phase change heat transfer processes. The functions provided in the Boiling and Condensation library are listed in Table 12-1. Detailed information about these functions is available in EES from the Function Information dialog (Options menu).

As an example, we will use the Boiling and Condensation library to calculate the average heat transfer coefficient for R134a that is evaporating in a 2.0 cm diameter tube. The R134a steadily enters the tube at 260 K with a quality of 0.36 at a mass velocity of 200 kg/s-m^2 and exits as saturated vapor (quality = 1). The average heat flux for the tube is 1600 W/m^2.

Select Function Information from the Options menu and click on the radio button titled Heat Transfer. If it is not already selected, click in the list to right of the radio button to select the Boiling and Condensation library, as shown in Figure 12-1. This library provides functions for the subjects of Boiling, Condensation and Pressure Drop. The Boiling subject should be selected by default, but if it is not, click the drop-down list that is shown on the right of the picture to select the Boiling category. The available functions for the Boiling category are each shown with a picture using the application library tools described in Section 11.4. Use the scroll bar under the picture to select different boiling heat transfer corrections. Clicking the Info button in this dialog will provide specific information for the selected function. The heat transfer library code is written as an EES library. Clicking the View button will display the EES code associated with this function in the library. Clicking the Index button will open the help file to an index page that displays information for all of the functions in this library. An example of the function call appears in the Example box. Click the Done button to close the dialog window.

Table 12-1: Organization of the Boiling and Condensation heat transfer library.

Menu	Function	Description
Boiling	Nucleate_Boiling	returns the heat flux associated with nucleate boiling
	Film_Boiling	returns the heat flux associated with film boiling
	Flow_Boiling	returns the local heat transfer coefficient & wall temperature for flow boiling
	Flow_Boiling_avg	returns the average heat transfer coefficient for flow boiling over a range of quality
	Critical_Heat_Flux	returns the critical heat flux
Condensation	Cond_HorizontalTube	returns the local heat transfer coefficient & flow regime for flow condensation
	Cond_HorizontalTube_avg	returns the average heat transfer coefficient for flow condensation
	Cond_vertical_plate	returns the heat transfer coefficient for film condensation from a vertical plate
	Cond_horizontal_Cylinder	returns the heat transfer coefficient for film condensation from the outer surface of a horizontal cylinder
	Cond_horizontal_N_Cylinders	returns the heat transfer coefficient for film condensation from the outer surface of a bank of horizontal cylinders
	Cond_finned_tube	returns the heat transfer coefficient for film condensation from the outer surface of a horizontal cylinder with annular fins
	Cond_horizontal_up	returns the heat transfer coefficient for film condensation from an upward facing plate
	Cond_horizontal_down	returns the heat transfer coefficient for film condensation from a downward facing plate
Pressure Drop	DELTAP_2phase_horiz	returns the pressure drop for two-phase flow through a tube over a range of qualities

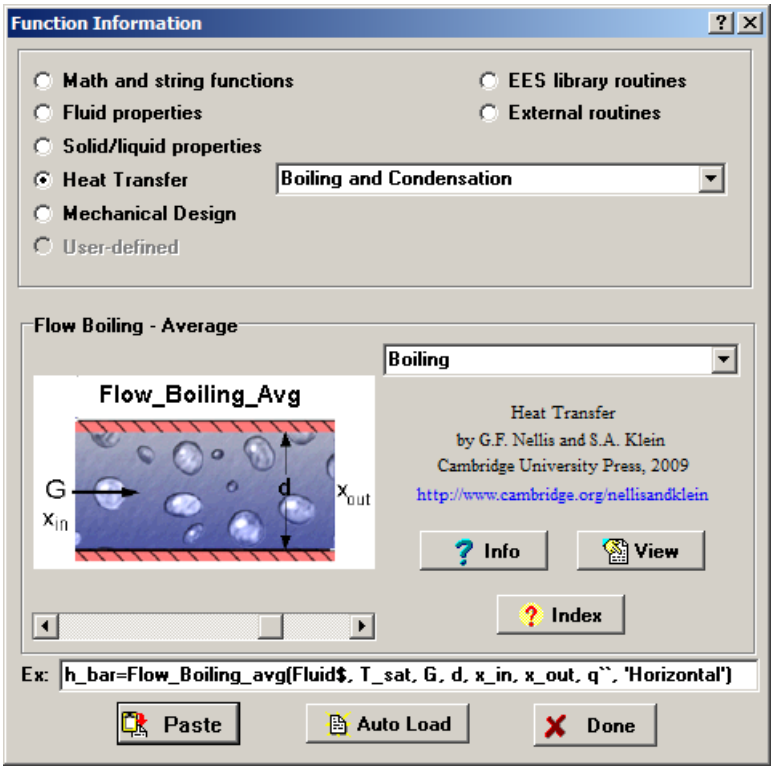


Figure 12-1: Function Information dialog showing information for the Flow_Boiling_Avg function.

Enter the known information for this problem into an empty Equations Window.

```
$UnitSystem SI K Pa J
$TabStops 3 in
Fluid$='R134a'
T_sat=260 [K]                    "boiling saturation temperature"
G=200 [kg/m^2-s]                 "mass velocity"
d=0.020 [m]                      "tube inner diameter"
x_in=0.36                        "quality at inlet"
x_out=1.00                       "quality at outlet"
q``=1600 [W/m^2]                 "surface heat flux"
```

Open the Function Information dialog and if necessary, select the Boiling and Condensation library and scroll to the Flow_Boiling_Avg function. Click the Paste button to paste the contents of the examples edit box into the Equations Window.

```
h_bar=Flow_Boiling_avg(Fluid$, T_sat, G, d, x_in, x_out, q``, 'Horizontal')
```

All of the inputs to the function have been specified. Select the Solve command. The Solutions Window should show that the average heat transfer coefficient is 2350 W/m^2-K.

12.2 Compact Heat Exchangers

The Compact Heat Exchanger library arranges much of the data presented in the book *Compact Heat Exchangers* by Kays and London (1984) in functional form in order to facilitate simulating heat exchangers with common core geometries.

Compact Heat Exchanger Data

The *Compact Heat Exchangers* book presents non-dimensional charts, typically in the form of Colburn j_H factor and friction factor as a function of Reynolds number as shown in Figure 12-2. Each chart is developed using data for a specific heat exchanger core. For example, Figure 12-2 shows data for the finned circular tube core fc_tubes_s80-38T. There are a large number of such cores available in the *Compact Heat Exchanger* book and these data are very useful because the air flow through the finned side of a compact heat exchanger is a complex combination of internal flow in passages and external flow over tubes for which generic correlations are not available. However, it is challenging to decipher the dimensionless data shown in Figure 12-2 in order to provide useful engineering quantities like heat transfer coefficient and pressure drop. The Compact Heat Exchanger library in EES simplifies this process.

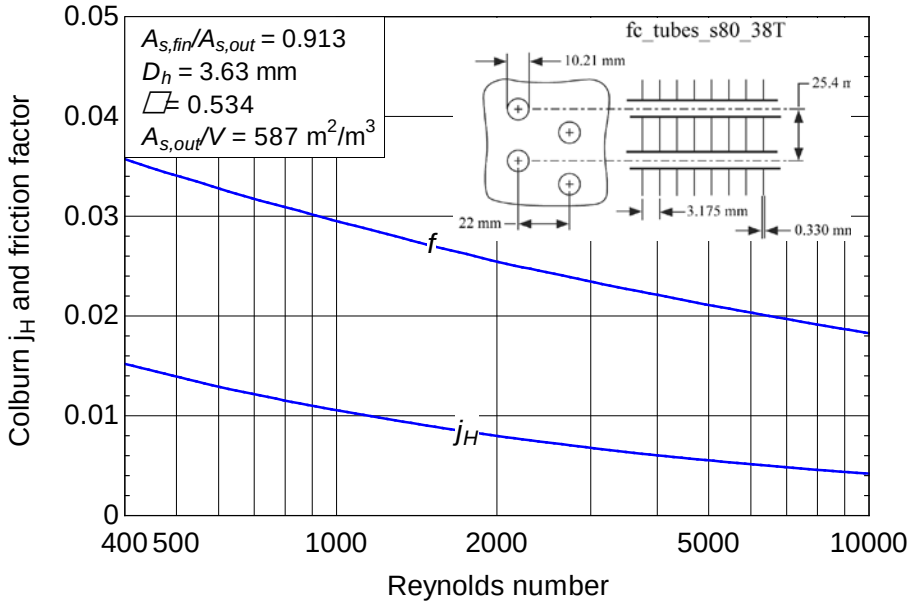


Figure 12-2: Typical compact heat exchanger chart showing the Colburn j_H factor and friction factor as a function of Reynolds number.

Organization of the Compact Heat Exchanger Library

Figure 12-3 illustrates the Function Information dialog for the Compact HX category in the heat transfer library.

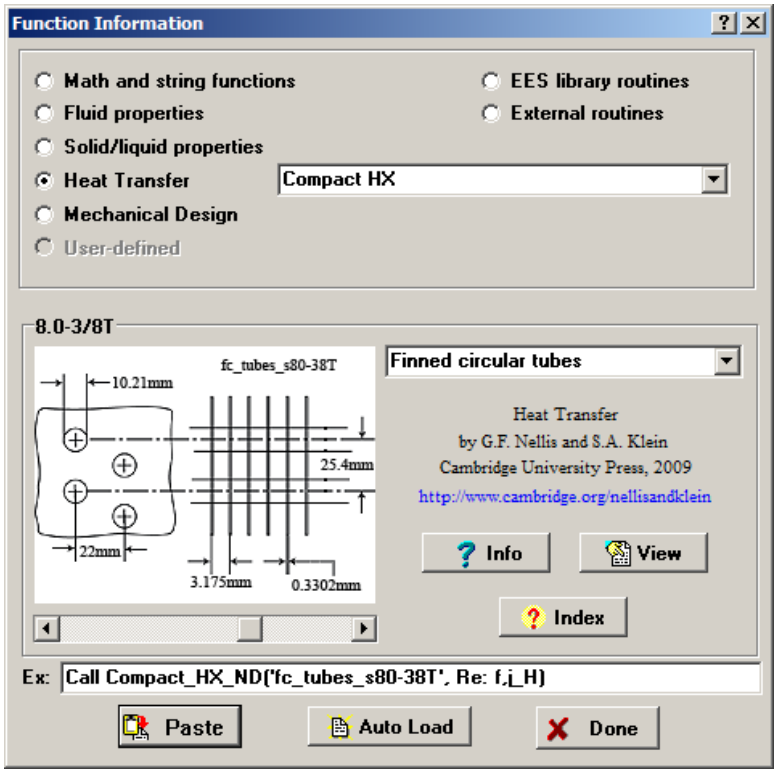


Figure 12-3: Function Information dialog for the Compact HX heat transfer library.

When the drop down menu to the right of the figure in the lower panel is selected, the user will see four categories of functions labeled Non-dimensional, Geometry, Coefficient of Heat Transfer, and Pressure Drop. Under each category there is a list of different types of cores. Selecting a certain type of core allows the user to scroll through the various specific geometries for which data are available.

Non-Dimensional Functions

Non-Dimensional functions provide the dimensionless parameters (j_H and f) as a function of Reynolds number; that is, they provide the information that could be obtained directly from Figure 12-2. This information is useful when it is necessary to compute the engineering quantities of interest (heat transfer coefficient and pressure drop) using the definition of the dimensionless parameters together with knowledge of the properties of the fluid and the core geometry.

Geometry Functions

Geometry functions provide the core-level geometric details of the core. These geometry factors are summarized in Table 12-2.

Table 12-2: Geometric details of the core returned by the Geometry functions in the Compact Heat Exchanger library.

Parameter	SI Unit	English Unit	Description
D_o	m	ft	outer diameter of tube[1]
fin_pitch	1/m	1/ft	number of fins per unit length
D_h	m	ft	hydraulic diameter of passages
fin_thk	m	ft	thickness of fins[2]
sigma	-	-	ratio of the minimum free flow area to the frontal area
alpha	m^2/m^3	ft^2/ft^3	ratio of the heat transfer surface area to the total volume
A_fin\A	-	-	ratio of the finned surface area to the total surface area
b_1	m	ft	thickness of passage through which fluid 1 flows

1. only applicable to circular tube and pin-fin plate-fin geometries; a value of -9999 will be returned for other geometries
2. not applicable to pin-fin plate-fin geometry

For example, consider a heat exchanger composed of the finned tube core shown in Figure 12-3 which is used in the following EES code:

```
$UnitSystem SI Mass J K Pa Rad
Call CHX_geom_finned_tube('fc_tubes_s80-38T': D_o, fin_pitch, D_h, fin_thk, sigma, alpha, A_fin\A)
```

The CHX_geom_finned_tube procedure call provides the core-level geometry for the finned tube core fc_tubes_s80_38T: D_o = 10.2 mm, fin_{pitch} = 315 1/m, D_h = 3.6 mm, fin_{thk} = 0.33 mm, σ = 0.534, α = 587 m^2/m^3, and A_{fin}/A = 0.913.

Coefficient of Heat Transfer Functions

Coefficient of Heat Transfer functions require the operating conditions including the fluid, mass flow rate, temperature and pressure, as well as the frontal area of the heat exchanger. These functions return the air-side heat transfer coefficient for the core of interest. For example, the following EES code:

```
$UnitSystem SI Mass J K Pa Rad
m_dot=0.2 [kg/s]                                   "mass flow rate"
A_fr=0.1 [m^2]                                     "frontal area"
Fluid$='Air'                                       "fluid"
T=300 [K]                                          "temperature"
P=100000 [Pa]                                      "pressure"
Call CHX_h_finned_tube('fc_tubes_s80-38T', m_dot, A_fr, Fluid$,T, P:h_bar)
```

calculates the heat transfer coefficient for air flowing through a heat exchanger core composed of finned tube core fc_tubes_s80_38T. The mass flow rate of air is $\dot{m}$ = 0.2 kg/s, the temperature is T= 300 K, the pressure is P = 100,000 Pa, and the frontal area is A_{fr} = 0.1 m^2. The resulting heat transfer coefficient is $\overline{h}$ = 55.6 W/m^2-K.

Pressure Drop Functions

Pressure Drop functions require the operating conditions including the fluid, mass flow rate, inlet and exit temperatures, and pressure, as well as the frontal area and length of the heat exchanger. These functions return the air-side pressure drop for the core of interest. For example, the following EES code:

```
$UnitSystem SI Mass J K Pa Rad
m_dot=0.2 [kg/s]                                   "mass flow rate"
A_fr=0.1 [m^2]                                     "frontal area"
Fluid$='Air'                                       "fluid"
T_i=300 [K]                                        "inlet temperature"
T_o=300 [K]                                        "outlet temperature"
P=100000 [Pa]                                      "pressure"
L=0.2 [m]                                          "length"
Call CHX_DELTAp_finned_tube('fc_tubes_s80-38T', m_dot, A_fr,L, Fluid$, T_i, T_o, P: DELTAp)
```

calculates the pressure drop for air flowing through a heat exchanger core composed of finned tube core fc_tubes_s80_38T. The mass flow rate of air is $\dot{m}$ = 0.2 kg/s, the inlet and outlet temperatures are T_i = 300 K and T_o = 300 K, the pressure is P = 100,000 Pa, the frontal area is A_{fr} = 0.1 m^2, and the length is L = 0.2 m. The resulting pressure drop is ΔP = 41.8 Pa.

12.3 Conduction Shape Factors

Conduction shape factors are used to encapsulate the solution to various two- and three-dimensional steady state conduction problems. The shape factor solution characterizes the rate of heat transfer ($\dot{q}$) between two surfaces (at temperatures T_1 and T_2) through a stationary medium with conductivity (k). The definition of the shape factor is:

$$S = \frac{\dot{q}}{k(T_1 - T_2)} \tag{12-1}$$

The shape factor is a function only of the geometry of the surfaces.

The index to the Conduction Shape Factors library in EES can be most easily viewed by selecting Conduction Shape Factors from the Heat Transfer Library drop down menu and then clicking the Index button. The graphical display of the various geometries that appears, shown in Figure 12-4, can be scrolled through; selecting a particular case leads to a dialog that provides additional information.

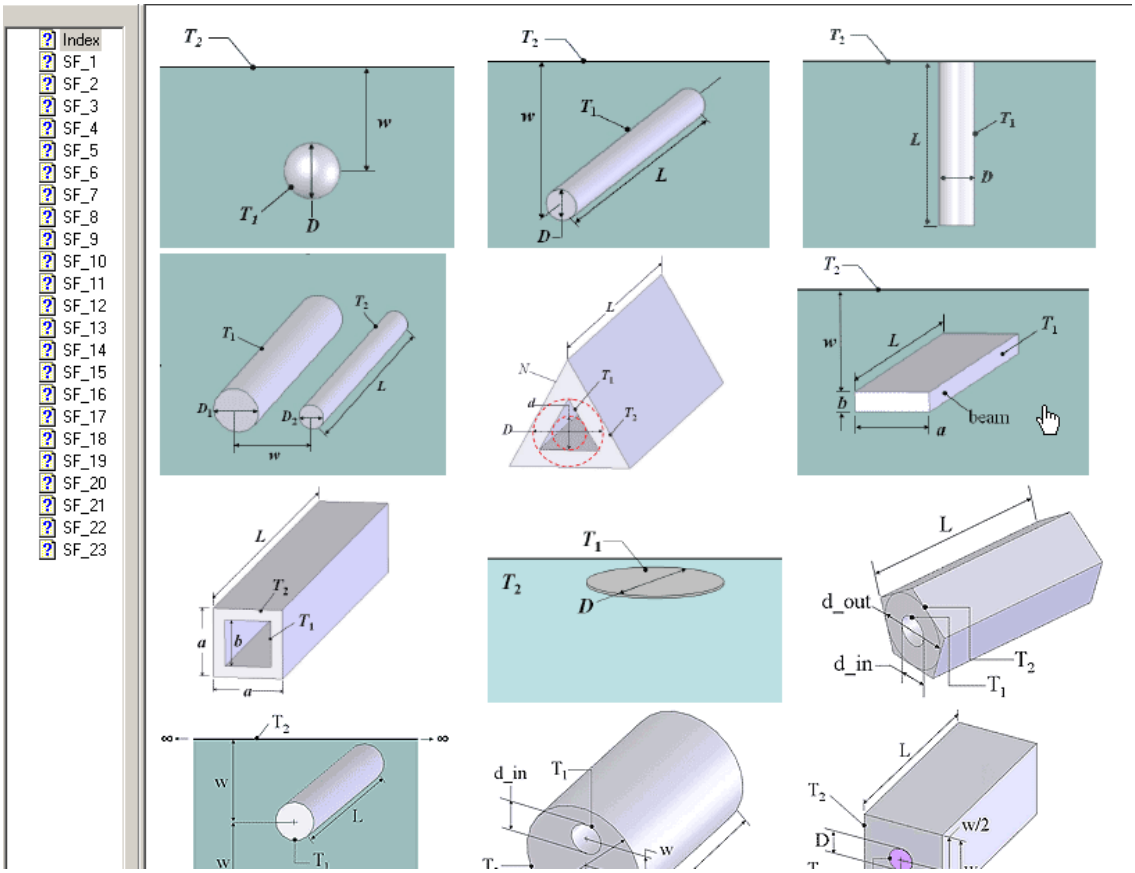


Figure 12-4: Index of conduction shape factors available in EES.

12.4 Convection

The Convection library contains functions that are used to compute heat transfer and pressure drop for single phase heat transfer processes. The correlations programmed in EES include internal and external forced convection, free convection, and flow through some regenerator packings.

Internal Forced Convection

There are four menus that pertain to internal forced convection (i.e., forced flow through a passage where the boundary layers become confined), the organization of these menus is shown in Table 12-3. These menus include Internal Flow - Dimensional which provides average (along the length of the passage) dimensional quantities, Internal Flow - Non-dimensional which provides average dimensionless quantities, Internal Flow - Dim (local) which provides local dimensional quantities, and Internal Flow - Non-dim (local) which provides local non-dimensional quantities.

Note that both dimensional and non-dimensional versions of most internal flow procedures are available. The non-dimensional versions provide the non-dimensional quantities of interest (e.g., Nusselt number and friction factor) given the non-dimensional conditions (e.g., Reynolds number and Prandtl number). The dimensional versions provide the dimensional quantities of interest (e.g., heat transfer coefficient and pressure drop) given the dimensional conditions (e.g., fluid, temperature, pressure, velocity, etc.). These two versions of the function are related. The dimensional version of the function calculates the non-dimensional input parameters to the non-dimensional version of the function. The non-dimensional version is called to obtain the non-dimensional quantity of interest which is finally used to obtain the dimensional quantities of interest.

Table 12-3: Organization of the forced internal convection portion of the heat transfer library.

Menu	Function	Description
Internal Flow - Dimensional	PipeFlow	returns the average heat transfer coefficient[1] and total pressure drop for flow through a round tube given the dimensional conditions
	DuctFlow	returns the average heat transfer coefficient[1] and total pressure drop for flow through a rectangular duct given the dimensional conditions
	Annular Flow	returns the average heat transfer coefficient[1] and total pressure drop for flow through an annular space given the dimensional conditions
	PipeDimensions	provides dimensional information for standard pipe sizes
Internal Flow – Non-dimensional	PipeFlow_N	returns the average Nusselt number[1] and average friction factor for flow through a round tube given the non-dimensional conditions
	DuctFlow_N	returns the average Nusselt number[1] and average friction factor for flow through a rectangular duct given the non-dimensional conditions
	AnnularFlow_N	returns the average Nusselt number[1] and average friction factor for flow through an annular space given the non-dimensional conditions
Internal Flow – Dim (local)	PipeFlow_local	returns the local heat transfer coefficient[1] and local pressure gradient for flow through a round tube given the dimensional conditions
	DuctFlow_local	returns the local heat transfer coefficient[1] and local pressure gradient for flow through a rectangular duct given the dimensional conditions
	AnnularFlow_local	returns the local heat transfer coefficient[1] and local pressure gradient for flow through an annular space given the dimensional conditions
Internal Flow – Non-dim (local)	PipeFlow_N_local	returns the local Nusselt number[1] and local friction factor for flow through a round tube given the non-dimensional conditions
	DuctFlow_N_local	returns the local Nusselt number[1] and local friction factor for flow through a rectangular duct given the non-dimensional conditions
	AnnularFlow_N_local	returns the local Nusselt number[1] and local friction factor for flow through an annular space given the non-dimensional conditions

1. both the constant wall temperature and constant heat flux values are returned

External Forced Convection

There are two menus that pertain to external forced convection (i.e., forced flow over a body where the boundary layers are not confined); the organization is shown in Table 12-4. Notice that both dimensional and non-dimensional versions are provided. Detailed information concerning these functions can be obtained from the Function Information dialog.

Table 12-4: Organization of the forced external convection portion of the heat transfer library.

Menu	Function	Description
External Flow – Dimensional	External_Flow_Sphere	returns the drag force and average heat transfer coefficient for flow over a sphere given the dimensional conditions
	External_Flow_Cylinder	returns the drag force per unit length and average heat transfer coefficient for flow over a cylinder given the dimensional conditions
	External_Flow_Plate	returns the average shear stress and average heat transfer coefficient for flow over a flat plate given the dimensional conditions
	External_Flow_Inline_Bank	returns the pressure drop and average heat transfer coefficient for flow across an inline bank of tubes given the dimensional conditions
	External_Flow_Staggered_Bank	returns the pressure drop and average heat transfer coefficient for flow across a staggered bank of tubes given the dimensional conditions
	External_Flow_Diamond	returns the drag force per unit length and average heat transfer coefficient for flow over an extruded diamond shape given the dimensional conditions
	External_Flow_Square	returns the drag force per unit length and average heat transfer coefficient for flow over an extruded square shape given the dimensional conditions
	External_Flow_Hexagon1	returns the drag force per unit length and average heat transfer coefficient for flow over an extruded hexagonal shape with a flat perpendicular to the flow given the dimensional conditions
	External_Flow_Hexagon2	returns the drag force per unit length and average heat transfer coefficient for flow over an extruded hexagonal shape with a flat parallel to the flow given the dimensional conditions
	External_Flow_VerticalPlate	returns the drag force per unit length and average heat transfer coefficient for flow perpendicular to a flat plate given the dimensional conditions
External Flow – Non-dimensional	External_Flow_Sphere_ND	returns the drag coefficient and average Nusselt number for flow over a sphere given the non-dimensional conditions
	External_Flow_Cylinder_ND	returns the drag coefficient and average Nusselt number for flow over a cylinder given the non-dimensional conditions
	External_Flow_Plate_ND	returns the friction coefficient and average Nusselt number for flow over a flat plate given the non-dimensional conditions
	External_Flow_Diamond_ND	returns the drag coefficient and average Nusselt number for flow over an extruded diamond shape given the non-dimensional conditions
	External_Flow_Square_ND	returns the drag coefficient and average Nusselt number for flow over an extruded square shape given the non-dimensional conditions
	External_Flow_Hexagon1_ND	returns the drag coefficient and average Nusselt number for flow over an extruded hexagonal shape with a flat perpendicular to the flow given the non-dimensional conditions
	External_Flow_Hexagon2_ND	returns the drag coefficient and average Nusselt number for flow over an extruded hexagonal shape with a flat parallel to the flow given the non-dimensional conditions
	External_Flow_VerticalPlate_ND	returns the drag coefficient and average Nusselt number for flow perpendicular to a flat plate given the non-dimensional conditions

Free Convection

There are two menus that pertain to free convection (i.e., natural convection where fluid motion is induced by buoyancy force); the organization of these menus is shown in Table 12-5. Notice that there are again both dimensional and non-dimensional versions for most of the procedures.

Table 12-5: Organization of the free convection portion of the heat transfer library.

Menu	Function	Description
Free Convection – Dimensional	Tilted_Rect_Enclosure	returns the heat transfer coefficient for a tilted rectangular enclosure given the dimensional conditions
	FC_plate_vertical	returns the heat transfer coefficient for a heated (or cooled) vertical plate given the dimensional conditions
	FC_plate_horizontal1	returns the heat transfer coefficient for a heated upward facing (or cooled downward facing) horizontal plate given the dimensional conditions
	FC_plate_horizontal2	returns the heat transfer coefficient for a heated downward facing (or cooled upward facing) horizontal plate given the dimensional conditions
	FC_plate_tilted	returns the heat transfer coefficient for a tilted plate that is either heated or cooled given the dimensional conditions
	FC_sphere	returns the heat transfer coefficient for a heated (or cooled) sphere given the dimensional conditions
	FC_horizontal_cylinder	returns the heat transfer coefficient for a heated (or cooled) horizontal cylinder given the dimensional conditions
	FC_vertical_cylinder	returns the heat transfer coefficient for a heated (or cooled) vertical cylinder given the dimensional conditions
	FC_vertical_channel	returns the heat transfer coefficient for a heated (or cooled) vertical channel formed by two infinite plates given the dimensional conditions
Free Convection – Non-dimensional	Tilted_Rect_Enclosure_ND	returns the Nusselt number for a tilted rectangular enclosure given the non-dimensional conditions
	FC_plate_vertical_ND	returns the Nusselt number for a heated (or cooled) vertical plate given the non-dimensional conditions
	FC_plate_horizontal1_ND	returns the Nusselt number for a heated upward facing (or cooled downward facing) horizontal plate given the non-dimensional conditions
	FC_plate_horizontal2_ND	returns the Nusselt number for a heated downward facing (or cooled upward facing) horizontal plate given the non-dimensional conditions
	FC_sphere_ND	returns the Nusselt number for a heated (or cooled) sphere given the non-dimensional conditions
	FC_horizontal_cylinder_ND	returns the Nusselt number for a heated (or cooled) horizontal cylinder given the non-dimensional conditions
	FC_vertical_cylinder_ND	returns the Nusselt number for a heated (or cooled) vertical cylinder given the non-dimensional conditions
	FC_vertical_channel_ND	returns the Nusselt number for a heated (or cooled) vertical channel formed by two infinite plates given the non-dimensional conditions

Regenerator Packings

Regenerator packings are matrices of solid material with a geometry that provides a high surface area per volume so that fluid flowing through the packing is in intimate contact with the solid. Typical regenerator packings are beds of spheres or stacked screens. There are two menus that pertain to flow through regenerator packings. The organization of these menus is shown in Table 12-6. Notice that again both dimensional and non-dimensional versions of these functions are available.

Table 12-6: Organization of the regenerator packing portion of the heat transfer library.

Menu	Function	Description
Regenerator Packing – Dimensional	PackedSpheres	returns the friction factor, heat transfer coefficient and number of transfer units for a packed bed of spheres given the dimensional conditions
	Screens	returns the friction factor, heat transfer coefficient and number of transfer units for a packed bed of screens given the dimensional conditions
	Triangular_channels	returns the friction factor, heat transfer coefficient and number of transfer units for a packed bed of triangular channels given the dimensional conditions
Regenerator Packing – Non-dimensional	PackedSpheres_ND	returns the friction factor and Colburn j-factor for a packed bed of spheres given the non-dimensional conditions
	Screens_ND	returns the friction factor and Colburn j-factor for a packed bed of screens given the non-dimensional conditions
	Triangular_channels_ND	returns the friction factor and Colburn j-factor for a packed bed of triangular channels given the non-dimensional conditions

12.5 Fin Efficiency

Fins are surfaces that extend into a fluid from a base material in order to increase the surface area that is exposed to convection. The fin efficiency is used to encapsulate the solution for various types of fin geometries. Fin efficiencies are derived with the assumption that the extended surface approximation applies; that is, the temperature distribution is 1-D. The fin efficiency is defined as:

$$\eta_{fin} = \frac{\dot{q}}{\overline{h}\, A_s \left(T_b - T_\infty\right)} \tag{12-2}$$

where $\overline{h}$ is the heat transfer coefficient between the fluid and the surface, A_s is the area of the fin that is exposed to fluid, T_b is the temperature of the base material, T_∞ is the temperature of the fluid, and $\dot{q}$ is the rate of heat transfer from the fin to the fluid. Fin efficiency solutions in EES are provided both in dimensional and non-dimensional terms, like the convection solutions. The organization of the Fin Efficiency library is shown in Table 12-7.

Table 12-7: Organization of the Fin Efficiency heat transfer library.

Menu	Function	Description
Dimensional Efficiency	eta_fin_straight_rect	returns the efficiency of a constant rectangular cross-sectional area fin given the dimensional conditions
	eta_fin_straight_triangular	returns the efficiency of a straight fin with a triangular profile given the dimensional conditions
	eta_fin_straight_parabolic	returns the efficiency of a straight fin with a parabolic, concave profile given the dimensional conditions
	eta_fin_straight_parabolic2	returns the efficiency of a straight fin with a parabolic, convex profile given the dimensional conditions
	eta_fin_spine_rect	returns the efficiency of a constant circular cross-sectional area fin given the dimensional conditions
	eta_fin_spine_triangular	returns the efficiency of a fin with a circular cross-section and triangular profile given the dimensional conditions
	eta_fin_spine_parabolic	returns the efficiency of a fin with a circular cross-section and parabolic, concave profile given the dimensional conditions
	eta_fin_spine_parabolic2	returns the efficiency of a fin with a circular cross-section and parabolic, convex profile given the dimensional conditions
	eta_fin_annular_rect	returns the efficiency of an annular fin with a rectangular profile given the dimensional conditions
Non-dimensional efficiency	each of the dimensional efficiency functions are also available in the non-dimensional library by adding _ND to the function name	

Note that in order for the fin efficiency solutions to be useful, it is necessary to know the surface area that is exposed to fluid, A_s in Eq. (12-2). The equations for the surface area are included in the Help information for each fin type; this information can be accessed by selecting Info from the Function Information dialog.

12.6 Fouling Factors

Heat exchangers that operate for a prolonged period of time will become "fouled"; that is, contamination will build up on the surfaces. This contamination will lead to an additional resistance to heat transfer and therefore degrade the heat exchanger performance. The resistance to heat transfer that results from fouling (R_f) is typically calculated using an area-specific fouling factor (R_f''):

$$R_f = \frac{R_f''}{A_s} \tag{12-3}$$

where A_s is the surface area that is exposed to fluid. Area-specific fouling factors are provided empirically for specific types of fluids and treatments (e.g., city water, river water, treated water, etc.). The Fouling Factor library provides some values of the area-specific fouling factors, as shown in Figure 12-5. These are organized by fluid type and then by treatment.

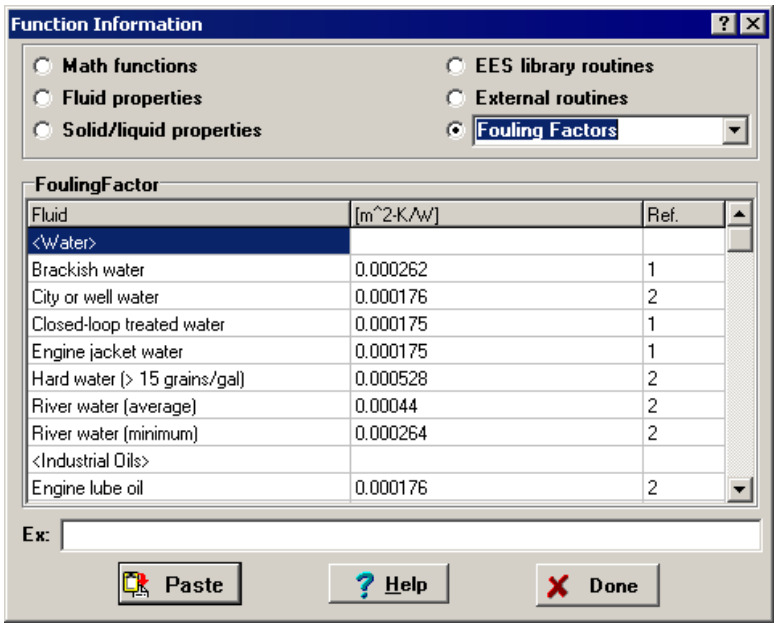


Figure 12-5: Fouling Factors library.

12.7 Heat Exchangers

Heat exchanger solutions for various configurations have been derived under the assumption of constant property fluids, no axial conduction or jacket heat loss, and uniform conductance. These solutions are typically represented in either log mean temperature difference (LMTD) or effectiveness-NTU (ε - *NTU*) form.

Log Mean Temperature Difference Solutions

The log mean temperature difference form of a heat exchanger solution is given by:

$$\dot{q} = \Delta T_{lm} UA \tag{12-4}$$

where $\dot{q}$ is the rate of heat transfer, UA is the conductance of the heat exchanger, and ΔT_{lm} is the log mean temperature difference. The log mean temperature difference is computed according to:

$$\Delta T_{lm} = F\, \Delta T_{lm,cf} \tag{12-5}$$

The parameter $\Delta T_{lm,cf}$ in Eq. (12-5) is the log mean temperature difference for a counterflow heat exchanger:

$$\Delta T_{lm,cf} = \frac{\left(T_{H,out} - T_{C,in}\right) - \left(T_{H,in} - T_{C,out}\right)}{\ln\left[\dfrac{\left(T_{H,out} - T_{C,in}\right)}{\left(T_{H,in} - T_{C,out}\right)}\right]} \tag{12-6}$$

where $T_{H,in}$ and $T_{H,out}$ are the hot fluid inlet and outlet temperatures, respectively, and $T_{C,in}$ and $T_{C,out}$ are the cold fluid inlet and outlet temperatures, respectively. The parameter F in Eq. (12-5) is a configuration-specific correction factor that must be less than unity. The correction factor is typically obtained as a function of two dimensionless parameters, P and R, using charts of the type shown in Figure 12-6 which is for a crossflow heat exchanger with both fluids unmixed.

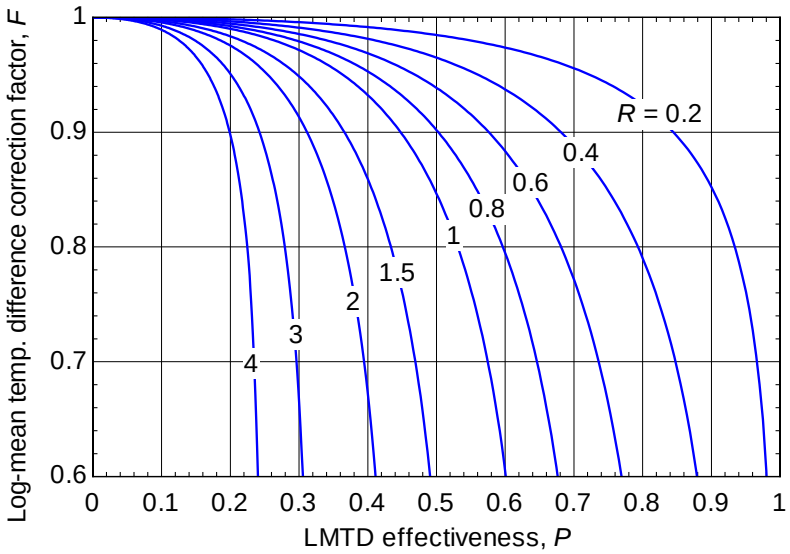


Figure 12-6: Correction factor for a crossflow heat exchanger with both fluids unmixed as a function of the parameters *P* and *R*.

The dimensionless parameters P and R are defined according to:

$$P = \frac{\left(T_{C,out} - T_{C,in}\right)}{\left(T_{H,in} - T_{C,in}\right)} \tag{12-7}$$

$$R = \frac{\left(T_{H,in} - T_{H,out}\right)}{\left(T_{C,out} - T_{C,in}\right)} \tag{12-8}$$

The first menu under the Heat Exchanger library is labeled F for LMTD, as shown in Figure 12-7. The user can scroll through the various configurations, as shown.

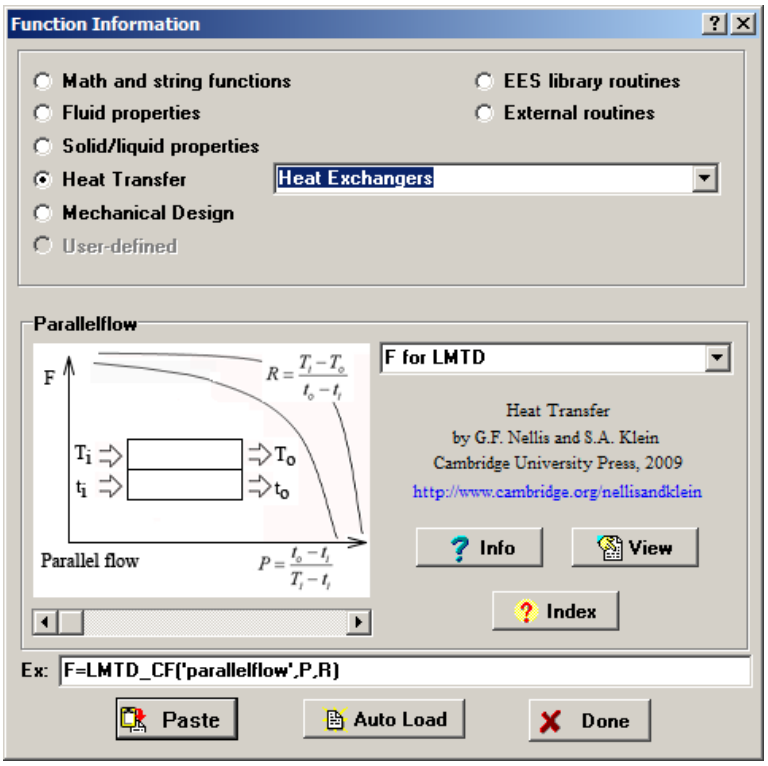


Figure 12-7: The F for LMTD menu under the Heat Exchanger library.

The calling format for the function LMTD_CF is:

```
F=LMTD_CF(Configuration$, P, R)
```

where Configuration$ is a string that contains the configuration (e.g., 'parallelflow' or 'shell&tube_N'), and P and R are the dimensionless parameters defined in Eqs. (12-7) and (12-8). The function LMTD_CF returns the correction factor F defined in Eq. (12-5).

Effectiveness-NTU Solutions

The effectiveness-NTU (ε - *NTU*) form of the solution is typically considered to be more convenient than the *LMTD* form. The two forms are algebraically identical; however, the ε - *NTU* form is easier to use for almost any problem. The ε - *NTU* form of a heat exchanger solution is essentially a relationship between the effectiveness of a heat exchanger (ε), the number of transfer units (*NTU*) and the capacitance rates of the fluids ($\dot{C}_C$ and $\dot{C}_H$) that has been derived for a specific heat exchanger configuration. The effectiveness is defined as:

$$\varepsilon = \frac{\dot{q}}{\dot{C}_{min}\left(T_{H,in} - T_{C,in}\right)} \tag{12-9}$$

where $\dot{C}_{min}$ is the minimum of $\dot{C}_C$ and $\dot{C}_H$. The number of transfer units is defined as:

$$NTU = \frac{UA}{\dot{C}_{min}} \tag{12-10}$$

The effectiveness-NTU solutions are typically arranged to either provide the effectiveness given *NTU* (which is useful for simulating a particular heat exchanger) or the *NTU* given the effectiveness (which is useful for designing a heat exchanger). There are two menu items labeled NTU -> Effectiveness and Effectiveness -> NTU in the Heat Transfer library which provide the solutions in each of these formats for a variety of heat exchanger configurations. For example, Figure 12-8 illustrates the Function Information dialog for the function under the NTU -> Effectiveness menu that provides the value of ε given *NTU* for a crossflow heat exchanger in which both fluids are unmixed.

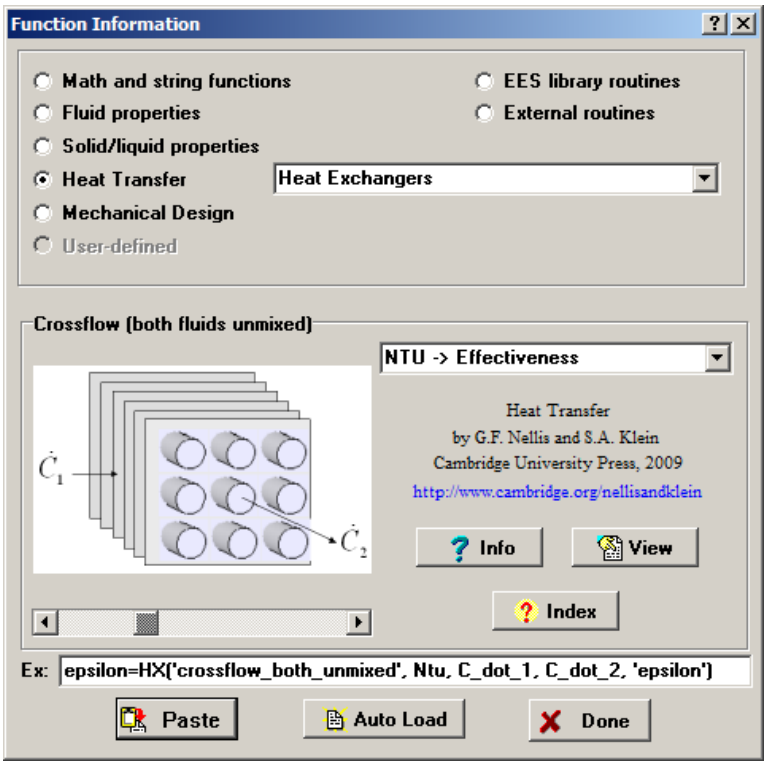


Figure 12-8: Function Information dialog showing the function that provides ε given *NTU* for a crossflow heat exchanger in which both fluids are unmixed.

Axial Conduction and Regenerator Models

The menu labeled Axial Conduction (numerical model) provides access to a numerical model of a counterflow heat exchanger that includes axial conduction. The menu labeled Regenerator (numerical model) provides access to a numerical model of a regenerator. These models are discussed in Nellis and Klein (2009).

12.8 Radiation View Factors

Radiation view factors are used to determine the degree to which two surfaces in a radiation problem are able to "see" one another which is important relative to how much radiation exchange occurs. The radiation view factor from surface *i* to surface *j* is defined as:

$$F_{i,j} = \frac{\text{radiation leaving surface } i \text{ that goes directly surface } j}{\text{total radiation leaving surface } i} \tag{12-11}$$

There are a number of different methods for computing the view factor between two surfaces including inspection, integration of the view factor equation, or Monte Carlos simulations. However, the view factor for many different surface combinations have been determined and are available from different sources in the form of equations, figures, and tables. Many of these view factor solutions have been programmed in EES and are available in the View Factor library. The view factor library is divided into three menus. The 2-Dimensional menu provides view factors for surfaces that extend infinitely; these view factors can be obtained using the Crossed and Uncrossed Strings method described by McAdams (1954). The 3-Dimensional menu provides view factors for a large number of three-dimensional surfaces. The Differential menu provides view factors in which one or both of the surfaces are differentially small; these are useful for problems with non-uniform temperature in which integration is required.

The best way to view and use the view factor library is to select the Index button and view the available functions graphically, as shown in Figure 12-9 for the three-dimensional view factor library.

Note that the functions in the view factor library return $F_{1,2}$; that is, the view factor from surface 1 to surface 2. Therefore, it is important to understand which of the two surfaces is identified as being surface 1 and which is surface 2. These surfaces are labeled on the diagram associated with the view factor function.

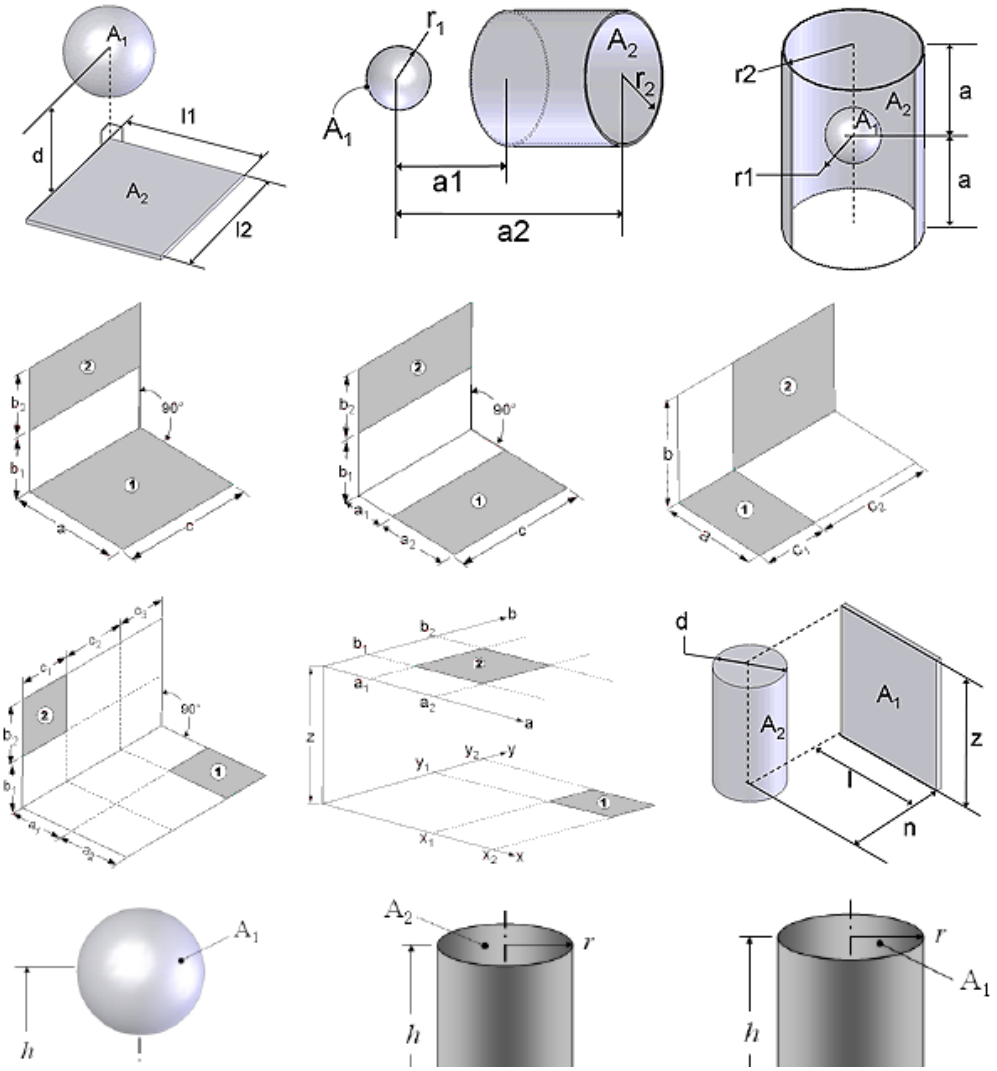


Figure 12-9: Graphical index of three-dimensional view factor library.

12.9 Transient Conduction

Analytical solutions to a number of transient conduction problems for typical shapes under common boundary conditions have been derived and several of these are available in the Transient Conduction library. These solutions are divided by whether the object of interest is semi-infinite or bounded. Semi-infinite objects have an infinite extent in the direction away from the surface at which the thermal disturbance occurs; for short times, all objects can be considered to be semi-infinite. The organization of the Transient Conduction library is shown in Table 12-8.

Table 12-8: Organization of the Transient conduction heat transfer library.

Menu	Function	Description
Semi-Infinite Body	SemiInf1	returns the temperature within a semi-infinite body subjected to a step change in the surface temperature given dimensional inputs
	SemiInf2	returns the temperature within a semi-infinite body subjected to a surface heat flux given dimensional inputs
	SemiInf3	returns the temperature within a semi-infinite body subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	SemiInf4	returns the temperature within a semi-infinite body subjected to a pulse of energy at time = 0 given dimensional inputs
	SemiInf5	returns the temperature within a semi-infinite body subjected to a sinusoidally varying surface temperature given dimensional inputs
Bounded 1-D - Dimensional	planewall_T	returns the temperature within a plane wall subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	planewall_Q	returns the total heat transfer (relative to time = 0) per area into a plane wall subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	cylinder_T	returns the temperature within a cylinder subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	cylinder_Q	returns the total heat transfer (relative to time = 0) per length into a cylinder subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	sphere_T	returns the temperature within a sphere subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
	sphere_Q	returns the total heat transfer (relative to time = 0) into a cylinder subjected to a step change in the temperature of the fluid adjacent to the surface (w/ a convective boundary condition) given dimensional inputs
Bounded 1-D - Non-dimensional	each of the functions in the Bounded 1-D - Dimensional library are also available in the non-dimensional library by adding _ND to the function name	

References

Kayes, W.M. and A.L. London, *Compact Heat Exchangers*, 3rd Edition, McGraw Hill, New York, (1984).

Klein, S.A. and Nellis, G.F., *Thermodynamics*, Cambridge University Press, New York, (2011).

McAdams, W.H., *Heat Transmission*, 3rd Edition, McGraw-Hill, New York, (1954).

Nellis, G.F. and S.A. Klein, *Heat Transfer*, Cambridge University Press, New York, (2009).

13 COMPLEX ALGEBRA

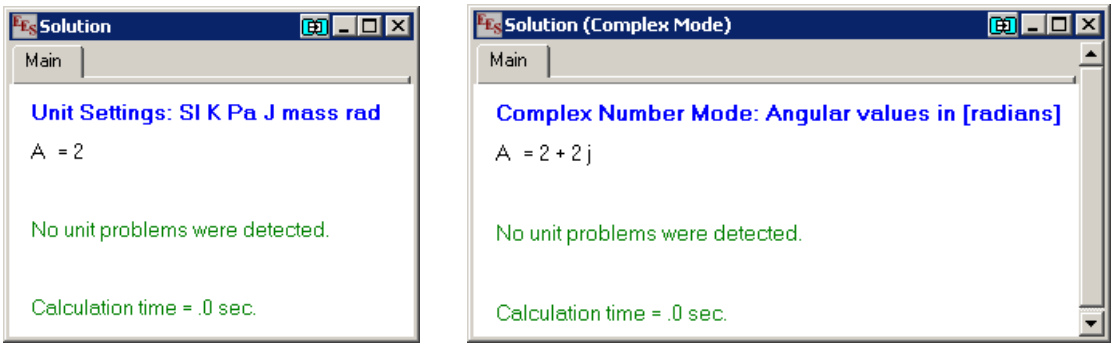
Variables in EES are associated with several types of information in addition to their numerical values. For example, variables can include units, limits, and guess values. If the Complex Algebra option is selected, then variables in EES can also have both real and imaginary parts. In this case, EES will carry out complex algebra automatically when performing the calculations required by the program. Complex numbers are often used in engineering analysis.

13.1 Introduction to Complex Algebra in EES

Up to this point, every EES variable has had only a real part. For example, the EES code:

```
A=2
```

leads to the solution shown in Figure 13-1(a).



(a) **(b)**

Figure 13-1: Solution Window with (a) complex algebra de-activated, and (b) complex algebra activated.

Activating Complex Algebra

A complex number can have both a real and imaginary part:

$$A = A_r + A_i\, j \tag{13-1}$$

where the subscript *r* indicates the real part and the subscript *i* indicates the imaginary part. The symbol *j* in Eq. (13-1) indicates the square root of negative one, $\sqrt{-1}$; often the symbol *i* is used for this purpose. To activate complex algebra in EES, select Preferences from the Options menu and then scroll to the Complex tab, as shown in Figure 13-2. Select Do complex algebra and then indicate whether the symbol i or j is to be used to indicate $\sqrt{-1}$.

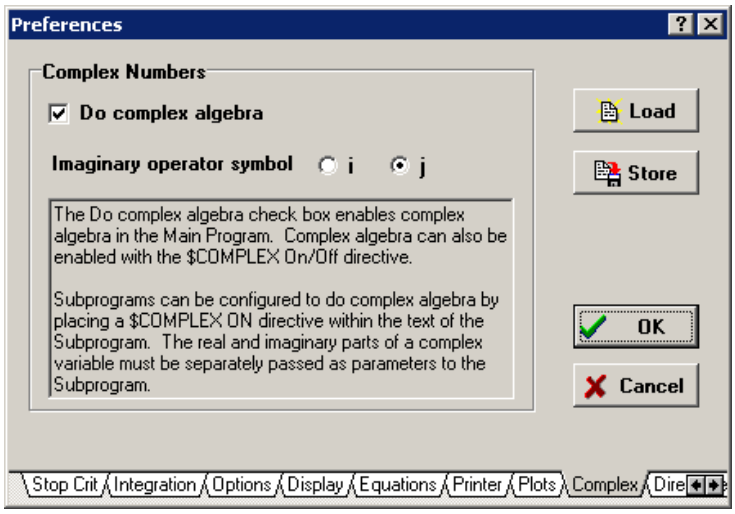


Figure 13-2: Complex tab in the Preferences dialog.

An alternative (and more convenient) method for activating complex algebra is to utilize the $Complex On directive. For example, the directive below:

```
$Complex On j
```

activates complex algebra and specifies that the symbol j will be used to indicate $\sqrt{-1}$.

Another way to active complex algebra is to click in the complex section of the status bar at the bottom of the Equations Window, as shown in Figure 13-3. There are three positions for this tab. Complex: Off, Complex: On (i) and Complex: On (j). Each click in this section of the status bar will toggle the complex algebra setting to the next option.

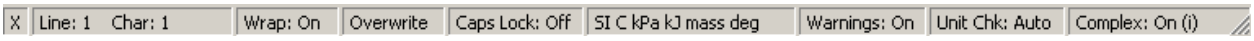


Figure 13-3: Status bar showing Complex section at the right.

Assigning Complex Variables in Rectangular Form

With complex algebra activated, each variable in EES has both a real and imaginary part. These parts can be assigned independently by using the subscript _r to indicate the real part and the subscript _i to indicate the imaginary part. For example, the EES code below sets the real and imaginary parts of the variable A to be 2:

```
$Complex On j
A_r=2                                   "real part of A"
A_i=2                                   "imaginary part of A"
```

Solving results in the Solution Window shown in Figure 13-1(b). Both parts of the complex variable can also be assigned at the same time:

```
$Complex On j
A=2+2*j                                 "assigning both parts at once"
```

Note that the statement:

```
$Complex On j
A=2
```

automatically sets both the real and imaginary parts of the variable A; the imaginary part is set to zero and the real part to two.

Assigning Complex Variables in Polar Form

Complex variables can be written in either rectangular or polar form. The specification of a complex variable in terms of its real and imaginary parts, as in Eq. (13-1), is its rectangular form. A complex number can alternatively be specified in terms of its magnitude and phase:

$$A = A_m \angle A_\theta \tag{13-2}$$

where A_m is the magnitude of the complex variable and A_θ is its phase. Equation (13-2) is the polar form of a complex number. The magnitude and phase are related to the real and imaginary components according to:

$$A_m = \sqrt{A_r^2 + A_i^2} \tag{13-3}$$

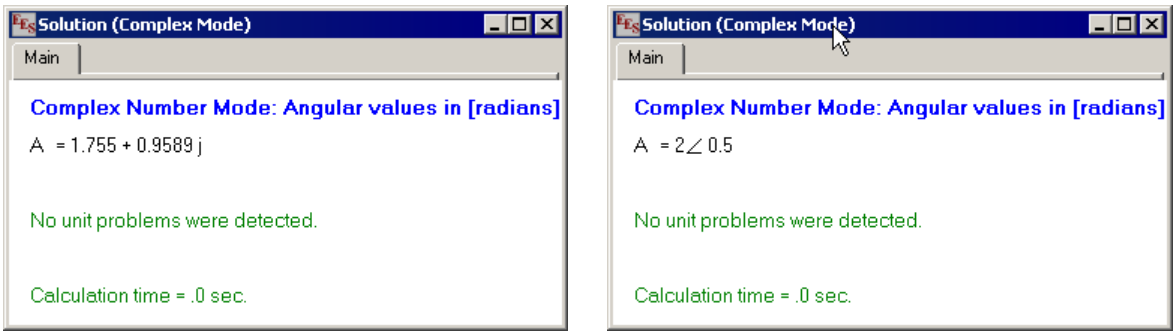
$$A_\theta = \tan^{-1}\left(\frac{A_i}{A_r}\right) \tag{13-4}$$

Complex variables can be defined in EES in their polar form according to Eq. (13-2). The magnitude is entered first followed by the symbol < and the angle:

```
$Complex On j
$UnitSystem SI Mass J K Pa Rad
A=2<0.5
```

Solving results in the solution shown in Figure 13-4(a).

Complex numbers can be output in either rectangular or polar form and the polar form can be expressed in either radians or degrees. Right-click on the value of A in the Solution Window to select the output format, as shown in Figure 13-5. If the output format is set to polar(rad) then the Solution Window will appear as shown in Figure 13-4(b).



(a) **(b)**

Figure 13-4: Solution (a) displayed in rectangular form, and (b) displayed in polar form.

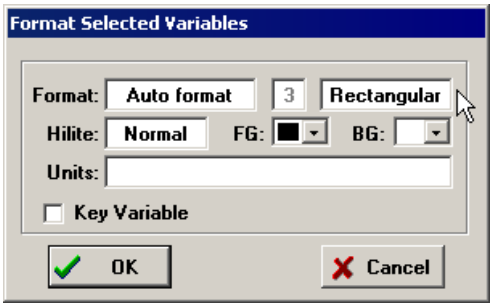


Figure 13-5: Selecting the output format for a complex variable in the Solution Window.

Notice that there was no designation of the unit for the angle in the above EES code; therefore, it is assumed that the angle (0.5) has units that are consistent with the unit system settings in EES (radians, according to the $UnitSystem directive). It is better to directly specify the unit for the angle in a polar specification of the angle. This can be done by entering either rad or deg immediately after the angle (without square brackets or a space):

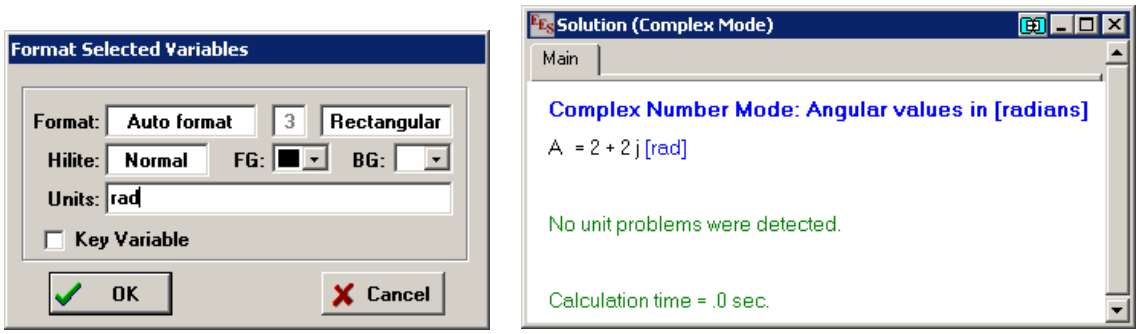
```
$Complex On j
A=2<0.5rad
```

The CIS function can be employed in place of the < symbol to establish the angle:

```
$Complex On j
A=2*CIS(0.5rad)
```

Units of Complex Variables

The units of both parts of a complex variable in EES must be the same. The units can be specified for a complex variable using the same techniques as for a real variable. In the Solution Window, right click on the variable to access the Format Selected Variables dialog shown Figure 13-6(a) and specify the units. The units will be indicated in the Solution Window, as shown in Figure 13-6(b).



(a) **(b)**

Figure 13-6: (a) Specify units using the Format Selected Variables dialog and (b) the resulting Solution Window.

The units of complex variables used in calculations will be checked in the usual manner.

The Variable Information Window

The characteristics of a complex variable in EES can be examined and modified using the Variable Information window (select Variable Info from the Options menu), as shown in Figure 13-7.

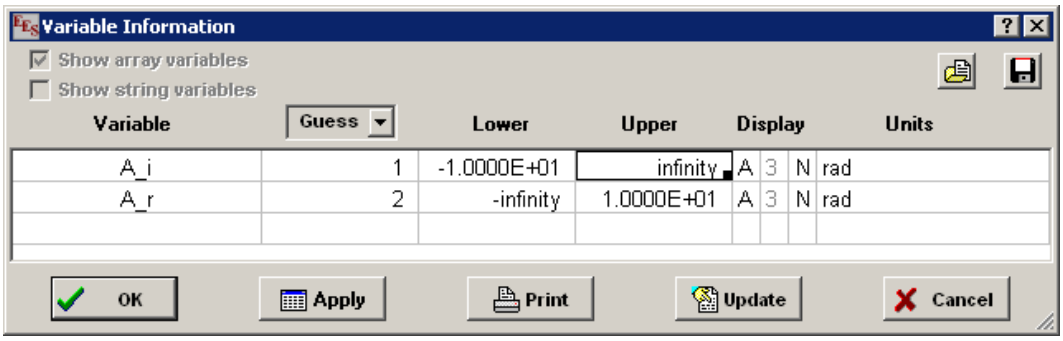


Figure 13-7: Variable Information Window showing a complex variable.

The guess value and limits of the complex and real parts of the variable can be set independently; however, the units of both parts will automatically be adjusted to be the same.

Complex Algebra

Calculations accomplished using complex variables will automatically carry out complex algebra. For example, the variables *A* and *B* can be defined as:

$$A = 2 + 2\,j \tag{13-5}$$

and

$$B = 3 + 1\,j \tag{13-6}$$

The product of *A* and *B* is also a complex number:

$$C = AB = (2+2\,j)(3+1\,j) = (6+2\,j+6\,j+2\,j^2) = 4+8\,j \qquad (13\text{-}7)$$

These operations are automatically carried out in EES:

```
$Complex On j
A=2+2*j
B=3+1*j
C=A*B
```

which leads to the solution shown in Figure 13-8.

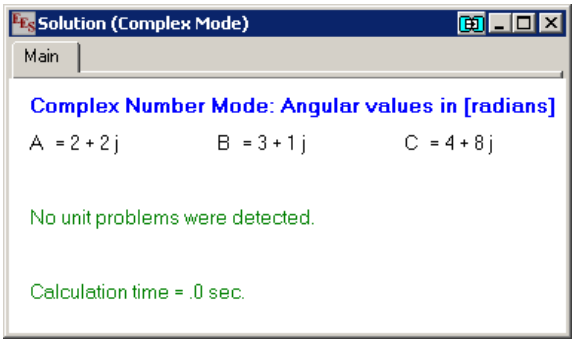


Figure 13-8: Solution window showing the result of complex algebra.

Similar operations are programmed in EES for division, addition, etc.

Functions Specific to Complex Variables

There are a few built-in functions that are specific to complex numbers; these are summarized in Table 13-1.

Table 13-1: Functions that are specific to complex variables.

Function	Description	Example	Returns
Real	returns the real part of a complex variable	Y=Real(2+3*j)	2
Imag	returns the imaginary part of a complex variable	Y=Imag(2+3*j)	3
Magnitude	returns the magnitude of a complex variable	Y=Magnitude(2+3*j)	3.606
Angle	returns the angle of a complex variable in the angle units associated with the EES Unit System	Y=Angle(2+3*j)	0.983 rad[1] 56.3 deg[2]
AngleDeg	returns the angle of a complex variable in degree	Y=AngleDeg(2+3*j)	56.3 deg
AngleRad	returns the angle of a complex variable in radian	Y=AngleRad(2+3*j)	0.983 rad
CIS	returns the complex number with magnitude 1 and angle specified by the argument[3]	Y=CIS(0.5rad)	0.8776+0.4794j
Conj	returns the complex conjugate of a complex number	Y=Comp(2+3*j)	2-3*j

1. if the EES Unit System is set to Rad for angles
2. if the EES Unit System is set to Deg for angles
3. the angle is assumed to be in the units set by the EES Unit System unless specified in the argument

Built-In Functions and Complex Variables

Many of the built-in mathematical functions in EES have been programmed to operate correctly with complex arguments; these are listed in Table 13-2. Many functions do not operate with complex arguments; examples of these functions are listed in Table 13-3.

Table 13-2: Built-in functions that operate with complex numbers.

Function	Description & Notes
Abs	returns the magnitude of a complex number
Cos, Sin, Tan	trigonometric functions
Cosh, Sinh, Tanh	hyperbolic trigonometric functions
Exp, Ln, Log10	exponential, natural log, and base 10 log
Sqrt	square root

Table 13-3: Built-in functions that do not operate with complex numbers.

Function(s)	Description
ArcX where X=Cos, Cosh, etc.	arc-cosine, arc-hyperbolic cosine, etc.
Average	average of arguments
BesselX where X = I, _I0, J, etc.	Bessel and modified Bessel functions
Ceil, Trunc, Round, Floor	rounding functions
Chi_Square, Gamma_, Probability	probability functions
Erf, Erfc	Gaussian error function and complementary error function
Max, Min	maximum and minimum of arguments
Mod	modulus of the arguments
Sign	sign of the argument

13.2 Complex Algebra Example

This section utilizes complex variables in EES in order to carry out a simple example related to AC circuits. Figure 13-9 illustrates an RCL circuit with inductance L = 22 mH, capacitance C = 1 μF, and resistance R = 1 kΩ. The input voltage to the circuit, V_{in}, varies sinusoidally with amplitude A = 1 V and frequency f = 1000 Hz:

$$V_{in} = A\sin\left(2\pi f t\right) \tag{13-8}$$

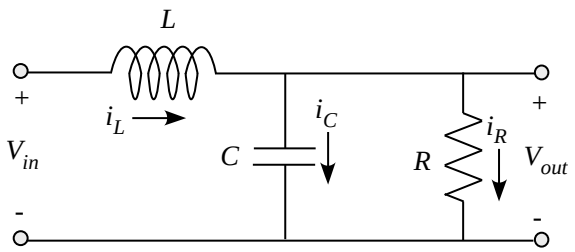


Figure 13-9: An RCL circuit.

The constitutive equations for the inductor, capacitor, and resistor are:

$$i_L = \frac{V_L}{L\,D} \tag{13-9}$$

$$i_C = C\,D V_C \tag{13-10}$$

$$i_R = \frac{V_R}{R} \tag{13-11}$$

where D represents the differential operator, i_L, i_C, and i_R are the current through the inductor, capacitor, and resistor, respectively, and V_L, V_C, and V_R are the voltage drop across each of these elements (in the direction of the current). A current balance leads to:

$$i_L = i_C + i_R \tag{13-12}$$

Substituting Eqs. (13-9) through (13-11) into Eq. (13-12) provides:

$$\frac{V_L}{L\,D} = C\,D V_C + \frac{V_R}{R} \tag{13-13}$$

The voltage across the resistor and capacitor are the same and equal to the output voltage:

$$V_C = V_R = V_{out} \tag{13-14}$$

The voltage across the inductor is equal to the difference between the input and output voltages:

$$V_L = V_{in} - V_{out} \tag{13-15}$$

Substituting Eqs. (13-14) and (13-15) into Eq. (13-13) leads to:

$$\frac{\left(V_{in} - V_{out}\right)}{L\,D} = C\,D V_{out} + \frac{V_{out}}{R} \tag{13-16}$$

Rearranging leads to the ordinary differential equation for the circuit:

$$V_{in} = L\,C\,D^2 V_{out} + \frac{L}{R} D V_{out} + V_{out} \tag{13-17}$$

which can be written as a transfer function:

$$TF\left(D\right) = \frac{V_{out}}{V_{in}} = \frac{1}{L\,C\,D^2 + \frac{L}{R} D + 1} \tag{13-18}$$

The frequency response of a linear system is given by:

$$V_{out} = TF_m(j\omega)A\sin\left[\omega t + TF_\theta(j\omega)\right] \tag{13-19}$$

where $\omega = 2\pi f$ is the angular frequency and $TF_m(j\omega)$ and $TF_\theta(j\omega)$ are the magnitude and angle associated with the complex number that is obtained by substituting $j\omega$ into the transfer function for the differential operator.

Complex algebra is activated using the $Complex On directive and the inputs are entered in EES:

```
$UnitSystem SI Mass J K Pa s Rad
$Complex On j
L=22e-3 [Henry]                    "inductance"
R=1000 [ohm]                       "resistance"
C=1e-6 [Farad]                     "capacitance"
A=1 [V]                            "amplitude of input voltage"
f=1000 [Hz]                        "frequency of input voltage"
```

The angular frequency is calculated and the value of the transfer function at $D = j\omega$ is determined.

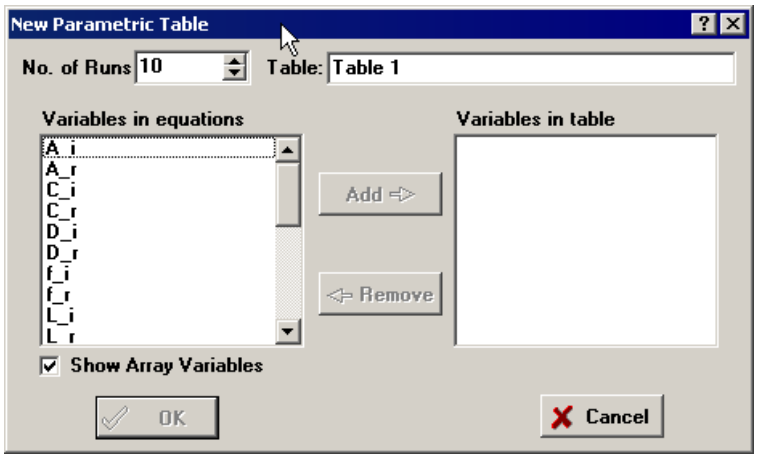
```
omega=2*pi*f                       "angular frequency"
D=j*omega                          "value of differential operator"
TF=1/(L*C*D^2+L*D/R+1)             "transfer function"
```

A value is specified for time ($t = 0$) and the input and output voltages are computed according to Eqs. (13-8) and (13-19), respectively. Note the use of the Magnitude and AngleRad functions to determine the magnitude and angle of the transfer function, respectively.

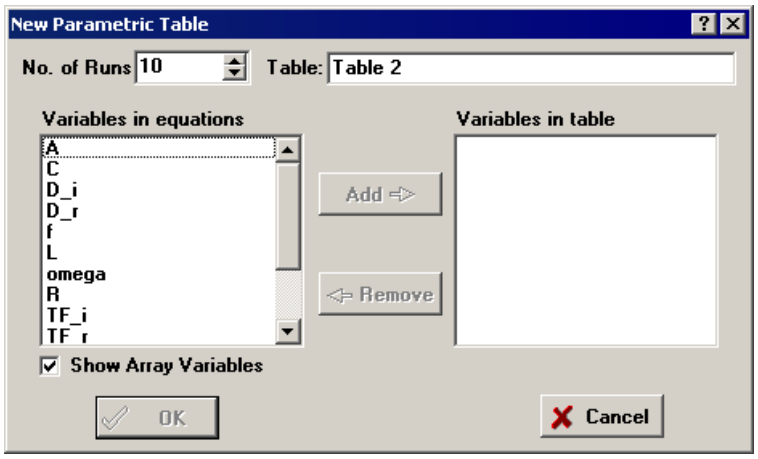
```
t=0 [s]                                          "time"
V_in=A*sin(omega*t)                              "input voltage"
V_out=A*Magnitude(TF)*sin(omega*t+AngleRad(TF))  "output voltage"
```

Parametric Table

We might want to prepare a plot showing the input and output voltages as a function of time during one cycle ($0 < t < 0.001$ s). Select New Parametric Table from the Tables menu in order to access the New Parametric Table dialog, shown in Figure 13-10(a).



(a)



(b)

Figure 13-10: New Parametric Table dialog (a) with all variables complex and (b) with some variables declared to be real using the $Real directive.

Notice that internally EES has created two variables for each of the variables in the problem; one for the real component (indicated by _r) and the other for the imaginary component (indicated by _i). In order to parametrically vary the value of the variable t, both t_r and t_i must be set. There are a number of ways to accomplish this. We could put both the variables t_r and t_i in the table as well as V_out_r and V_in_r. Set the value of t_i to 0 in every row and vary t_r from 0 to 0.001 s, as shown in Figure 13-11. Comment out the specified value of time in the Equations Window:

```
{t=0 [s]}                                    "time"
```

and run the Parametric Table.

Parametric Table

Table 1

1..10	1 t_i [s]	2 t_r [s]	3 $V_{out,r}$	4 $V_{in,r}$ [V]
Run 1	0	0	-3.798	0
Run 2	0	0.0001111	-0.5875	0.6428
Run 3	0	0.0002222	2.898	0.9848
Run 4	0	0.0003333	5.028	0.866
Run 5	0	0.0004444	4.805	0.342
Run 6	0	0.0005556	2.334	-0.342
Run 7	0	0.0006667	-1.23	-0.866
Run 8	0	0.0007778	-4.217	-0.9848
Run 9	0	0.0008889	-5.232	-0.6428

Figure 13-11: Parametric table with t_r, t_i, V_out_r and V_in_r.

Plot the values of V_out_r and V_in_r as a function of t_r in order to obtain Figure 13-12.

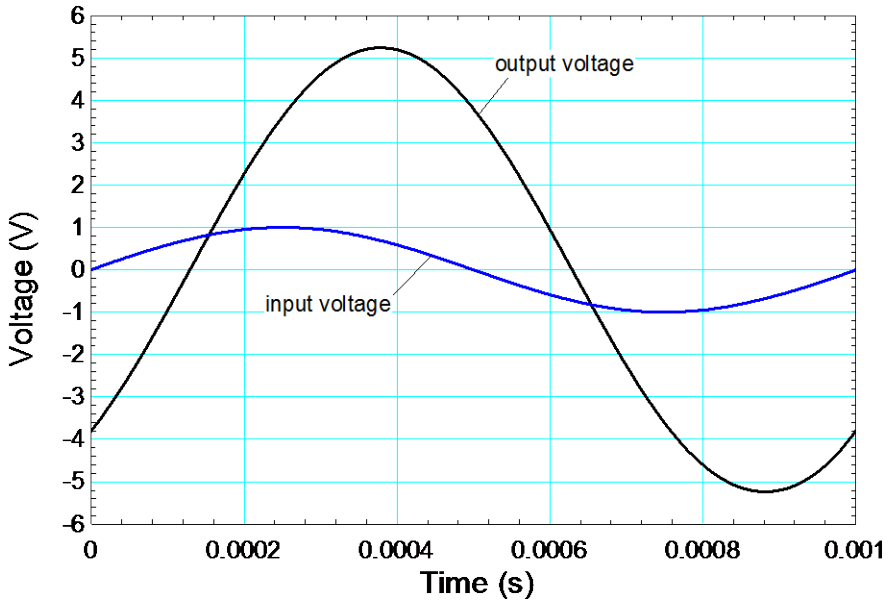


Figure 13-12: Output and input voltage as a function of time.

The $Real Directive

It is inconvenient to have to separately deal with the imaginary and real parts of the variable t since we know that time can only take on real values. The $Real directive allows you to declare variables that can only take on real values (i.e., their imaginary parts are always assumed to be zero). Any variable following the $Real directive will be considered to be real. For this problem, the variables L, R, C, A, f, omega, t, V_in, and V_out are all unambiguously real:

```
$UnitSystem SI Mass J K Pa s Rad
$Complex On j
$Real L R C A f omega t V_in V_out
```

```
L=22e-3 [Henry]                                  "inductance"
R=1000 [ohm]                                     "resistance"
C=1e-6 [Farad]                                   "capacitance"
A=1 [V]                                          "amplitude of input voltage"
f=1000 [Hz]                                      "frequency of input voltage"

omega=2*pi*f                                     "angular frequency"
D=j*omega                                        "value of differential operator"
TF=1/(L*C*D^2+L*D/R+1)                           "transfer function"

t=0 [s]                                          "time"
V_in=A*sin(omega*t)                              "input voltage"
V_out=A*Magnitude(TF)*sin(omega*t+AngleRad(TF))  "output voltage"
```

Select New Parametric Table from the Tables menu and you will see that only the variables D and TF are complex and therefore only these variables have a real and complex part in the variable list, as shown in Figure 13-10(b). It is only necessary to select the variables t, V_in, and V_out in order to carry out the parametric study.

Subprograms

Functions, modules, and procedures cannot operate in complex mode. However, subprograms do support complex algebra. Subprograms can be utilized in complex mode even if the Equations Window is not operating in complex mode, allowing complex calculations to be localized while the rest of the calculations are carried out in the normal way.

14 DIRECTIVES

EES compiles the information in the Equations Window into a stack-based form that allows for efficient computation before calculations are initiated. The information in the Equations Window consists mostly of equations, but it can also contain directives. Directives are instructions to the EES compiler that are acted on only during the compilation process. Directives are identified with a $ symbol as the first character. Directives can be used to accomplish tasks as simple as controlling the tab locations in the Equations Window or as complicated as controlling the order in which the equations are solved. We have referred to directives throughout this book. This chapter presents a comprehensive list of the directives that are available in EES. The directives are organized based on their purpose. A complete list of the directives implemented at the time of publication is provided in Appendix C. The following sections describe the directives and provide examples of their use.

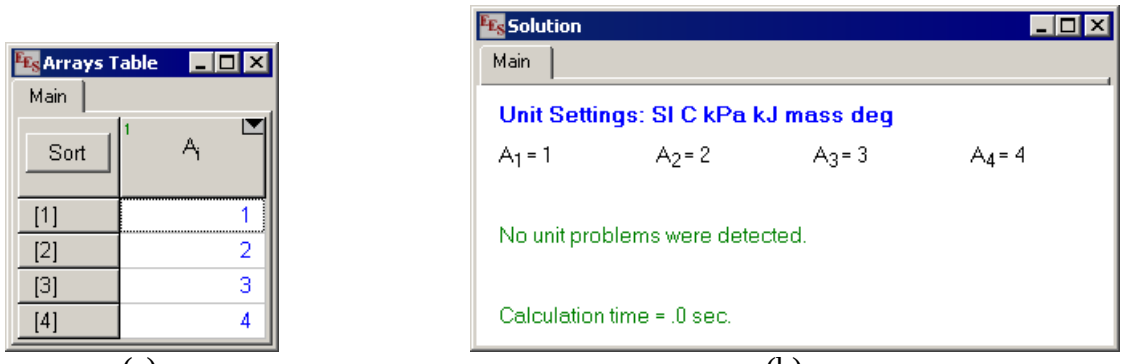
14.1 Directives related to Display & Formatting Options

$Arrays

Arrays are discussed in Section 1.7. Array variables can have multiple values and therefore behave as vectors or matrices. Array variables in the Main EES program are, by default, displayed in the Arrays Window. For example, consider the EES code:

```
duplicate i=1,4
    A[i]=i
end
```

which creates an array A[] that has four elements. Solve this problem and select Arrays from the Windows menu to access the Arrays Window, as shown in Figure 14-1(a).



(a) (b)

Figure 14-1: Array shown in (a) the Arrays Window and (b) the Solution Window.

Placing the $Arrays directive in the Main EES program controls whether the array is displayed in the Arrays Window ($Arrays ON) or the Solutions Window ($Arrays OFF). The EES code:

```
$Arrays OFF
```

```
duplicate i=1,4
    A[i]=i
end
```

will result in array variables shown in the Solution Window, as shown in Figure 14-1(b), rather than in the Arrays window.

Arrays can be used within functions and procedures, as discussed in Section 3.6. They can also be used in subprograms and modules, which are described in Chapter 10. It is not always convenient to pass arrays as input or output arguments from a function, procedure, subprogram, or a module. In many cases, the only reason for passing an array out is so that the values within the array can be used to create a plot or saved to a file. By default, array variables that are used within a function, procedure, subprogram, or module are placed in the associated Solution Window tab and therefore cannot be plotted. By placing the $Arrays On directive within function, procedure, subprogram, or module, the array variables in that code section are placed within the associated tab of the Arrays Window and they therefore can be plotted or saved to a file. This option is illustrated in Section 3.6.

$Bookmark

For large EES programs it is convenient to be able to quickly jump to a particular location in the program. You will find the $Bookmark directive to be a convenient method of navigating your program. The format of the $Bookmark directive is:

```
$Bookmark Name
```

where Name is a descriptive name given to that location in the code. For example, open the EES code titled "A heat pump cycle using R134a" which can be found by selecting EES Example Problems from the Examples menu and then selecting Properties, thermodynamic. This is not a particularly large program, but we can use it to illustrate the process of setting and using bookmarks. Set a bookmark (called Compressor) at the location in the code where the compressor is analyzed and another bookmark (called Condenser) at the location where the condenser is considered.

```
$Bookmark Compressor
"!Compressor"
x[1]=1                                  "saturated vapor at compressor inlet"
P[1]=pressure(R134a,T=T[1],x=x[1])
h[1]=ENTHALPY(R134a,T=T[1],x=x[1])
s[1]=entropy(R134a,T=T[1],x=x[1])
s_ID[2]=s[1]                            "ideal compressor is isentropic"
P[2]=pressure(R134a,T=T[3],x=1)
h_ID[2]=enthalpy(R134a,P=P[2],s=s_ID[2])
W_c_ID=-(h_ID[2]-h[1])*m_dot;           "power requirement for ideal compressor"
ComEff=0.60                             "Isentropic efficiency"
W_c=W_c_ID/ComEff                       "power requirement for actual compressor"
h[2]=h[1]-W_c/m_dot                     "energy balance on adiabatic compressor"
VolFlow=m_dot*volume(R134a,T=T[1],x=x[1])
VolFlow=4.3E-3 [m^3/s]                  "compressor volumetric flowrate"
```

```
"!Condenser"
$Bookmark Condenser
T_H=20 [°C]                                  "building air temperature"
Beta=1.75 [kW/C]                             "HX effectiveness-capacitance rate product"
Q_H=Beta*(T[3]-T_H)                          "heat exchanger relationship"
Q_H=(h[2]-h[3])*m_dot                        "energy balance"
h[3]=ENTHALPY(R134a,T=T[3],x=0)              "saturated liquid at condenser outlet"
P[3]=P[2]
```

Right click in the Equations Window (or the Formatted Equations Window) to bring up the pop-up menu, shown in Figure 14-2. All of the bookmarks that have been entered into the Equations Window will appear at the bottom of the menu. Selecting one of the bookmark menu items immediately moves the cursor and display of the window to the related portion of the equation set.

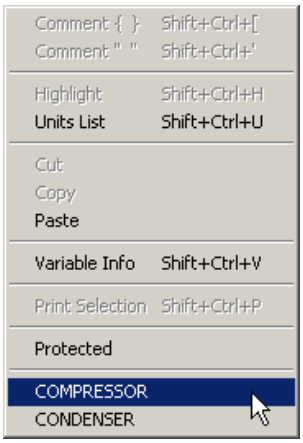


Figure 14-2: Pop-up menu with bookmarks.

$HideWindow

The $HideWindow directive causes a specified window to be hidden. The protocol for the $HideWindow directive is:

```
$HideWindow WindowName
```

where the parameter WindowName indicates the window to be hidden. The parameter WindowName can be set to any of the values listed in Table 14-1.

Table 14-1: Window names for use in the $HideWindow directive.

Window Name	Corresponding window to be hidden
Equations	Equations Window
Formatted	Formatted Equations Window
Plot	Plot Window
Diagram	Diagram Window
Solution	Solution Window
Arrays	Arrays Window
Parametric	Parametric Table Window
Lookup	Lookup Table Window
Residuals	Residuals Window
Integral	Integral Table Window

Report	Report Window

$Private

The development of library files that can contain internal functions, procedures and subprograms is discussed in Chapter 11. The EES code associated with a library file can normally be viewed from within the Function Information dialog by selecting the View Code button. For example, selecting the View Code button in the Function Information dialog shown in Figure 14-3 will allow the user to examine the EES code associated with the function Eb in the Blackbody library.

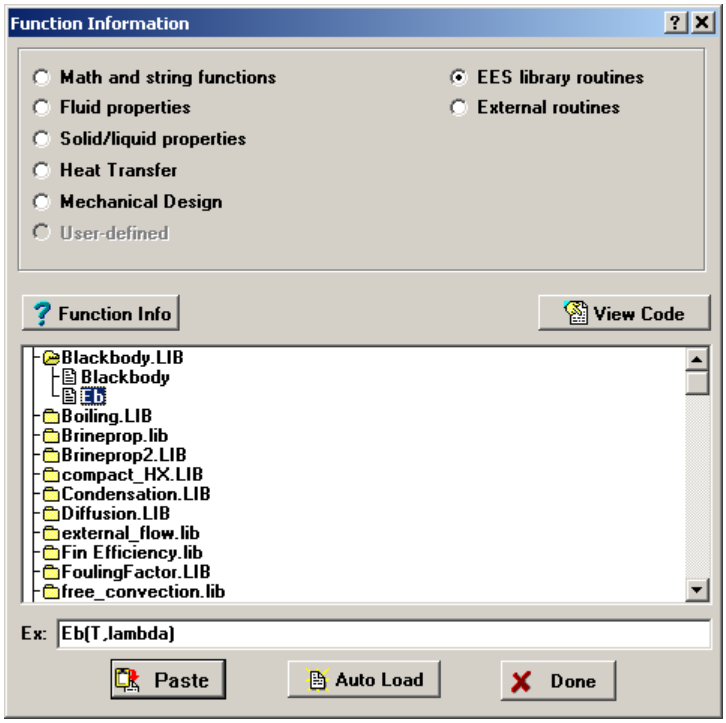


Figure 14-3: Function Information dialog for EES library routines showing the View Code button.

If the $Private directive is placed inside of a function, procedure or subprogram that is contained in a library file, then the EES code for the function will not be accessible using the View Code button. For example, if the Blackbody.lib file is opened from the Userlib\Heat Transfer\ folder and the $Private directed is inserted within the declaration for the function Eb:

```
function Eb(T,lambda)
    $Private
    C1=3.7420e8
    C2=1.4388e4
    sigma=5.670e-8 [W/m^2-K^4]
    Eb=C1/(lambda^5*(exp(C2/(lambda*T))-1))
end
```

then the function Eb will be removed from the list under Blackbody.lib, as shown in Figure 14-4. The help information for the private function can still be accessed by selecting Function Info for the library.

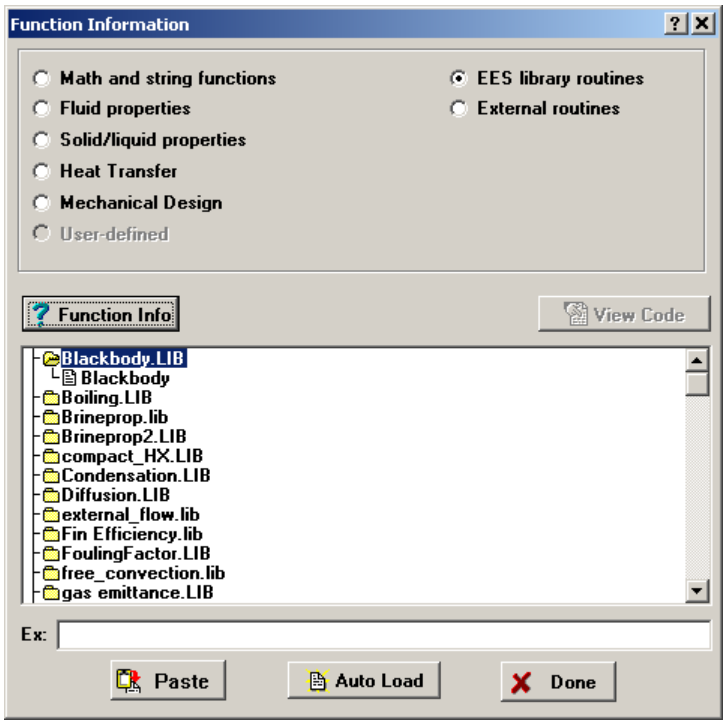


Figure 14-4: Function Information dialog for EES library routines showing that the function Eb is declared Private using the $Private directive.

$ShowWindow

By default, the Solution Window is shown after the calculations are completed (unless the calculations were initiated from the Diagram Window, as discussed in Chapter 15). The $ShowWindow directive allows a different window to be brought forward at the completion of the calculations. The format is:

```
$ShowWindow WindowName1 'WindowName2'
```

where WindowName1 is the name of the window to be shown. The types of window and their associated names are summarized in Table 14-2.

Table 14-2: Types of windows.

Window Name Indicator	Type of window
Equations	Equations Window
Formatted	Formatted Equations Window
Plot	Plot Window
Diagram	Diagram Window
Solution	Solution Window
Arrays	Arrays Window
Parametric	Parametric Table Window
Lookup	Lookup Table Window
Residuals	Residuals Window
Integral	Integral Table Window
Report	Report Window
Solution Key Variables	Key Variables tab of the Solutions Window
Diagram 'Child'	The Child Diagram window named 'Child'

The parameter 'WindowName2' is a string or string variable containing the specific name of the window, should there be more than one instance of the type of window specified by WindowName1. For example, you may have several Plot windows, but you want to show Plot2 after calculations are completed:

```
$ShowWindow Plot 'Plot2'
```

To have the Key Variables Solution window appear after solving, you could enter

```
$ShowWindow Solution Key Variables
```

The parameter 'WindowName2' can be replaced by an integer, in which case it specifies the position of the tab that will be displayed.

$SumRow

It is possible to include a row at the bottom of a Parametric Table that shows the sum of each of the columns. Consider the Parametric Table shown in Figure 14-5(a) which does not include a sum row. In order to add a sum row, check the Include a Sum row in the Parametric table in the Options tab of the Preferences dialog as shown in Figure 14-6. The resulting Parametric Table is shown in Figure 14-5(b). The sum row option can also be activated with the directive $SumRow On or deactivated with the directive $SumRow Off.

```
$SumRow On
```

Parametric Table

Table 1

1..8	1 a	2 b
Run 1	1	1
Run 2	2	4
Run 3	3	9
Run 4	4	16
Run 5	5	25
Run 6	6	36
Run 7	7	49
Run 8	8	64

(a)

Parametric Table

Table 1

1..8	1 a	2 b
Run 1	1	1
Run 2	2	4
Run 3	3	9
Run 4	4	16
Run 5	5	25
Run 6	6	36
Run 7	7	49
Run 8	8	64
Sum	36	204

(b)

Figure 14-5: Parametric Table (a) without sum row and (b) with sum row.

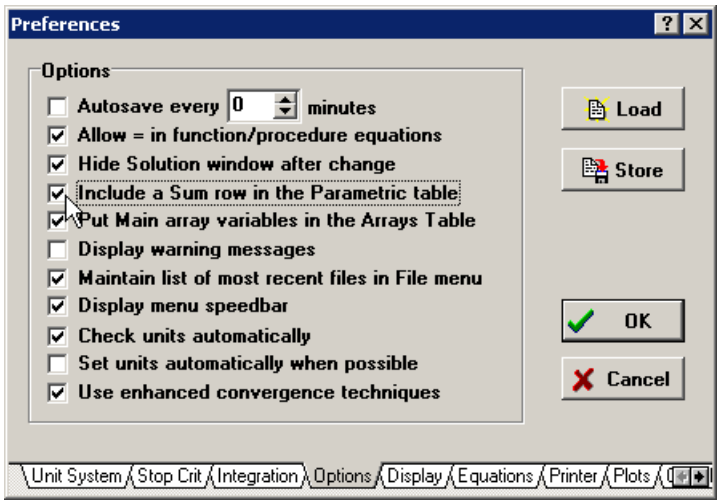


Figure 14-6: The option to Include a Sum row in the Parametric table is checked.

$TabStops

The $TabStops directive sets the tabs that are used in the Equations Window. Up to five tab stops may be set with this directive. The format for the $TabStops directive is:

```
$TabStops Stop1 Stop 2 ... Stop5 Unit
```

where Stop1 through Stop5 are numerical values in increasing order. The Unit must be either in (inch) or cm (centimeter). If no units are specified then inch is assumed if the Unit System is set to English and centimeter if the Unit System is set to SI.

$Warnings

Warnings in EES are generated for a variety of reasons. Warnings are generated internally by the EES program if the equations appear to be singular. Warnings can be generated within property correlations if they are used outside of their published range of applicability. Functions and procedures may also generate user-defined warnings, as discussed in Section 3.8. By default, warnings are displayed after the calculations have been completed. However, warnings can be disabled by de-selecting the Display warning messages in the Options tab of the Preferences dialog. Warnings can also be enabled or disabled using the $Warnings On and $Warnings Off directives, respectively.

14.2 Directives related to Units

$AutoSetUnits

The process of setting and checking the units of variables in EES is described in Section 1.5. The option of having EES automatically attempt to discern and set appropriate units for each variable based on the associated equations is also discussed in Section 1.5. This option can be activated or deactivated by placing the $AutoSetUnits On or $AutoSetUnits Off directive at the top of the Equations Window.

$CheckUnits

The process of checking units and some of the features of the $CheckUnits directive are discussed in Section 1.5. By default, EES will check the units of each equation each time that the program is run and report any unit warnings. This feature can be activated and deactivated by placing the $CheckUnits AutoOn or $CheckUnits AutoOff directive at the top of the Equations Window.

Unit checking can also be selectively deactivated for one or more sets of selected equations by enclosing them within a $CheckUnits Off directive and a $CheckUnits On directive, as shown.

```
Equation(s) which will have their units checked

$CheckUnits Off
Equation(s) which will not have their units checked
$CheckUnits On

Equation(s) which will have their units checked
```

This capability is useful if you would like to avoid entering the units of each of the constants in a curve-fit, as discussed in Section 2.2.

$Reference

The $Reference directive allows the reference state that is used for a built-in EES fluid that uses the fundamental equation of state to be adjusted from its default value. The format for the $Reference directive is:

```
$Reference Fluid  ReferenceID
```

where Fluid is the name of the fluid and ReferenceID indicates the reference state. The allowable values of ReferenceID are summarized in Table 14-3.

Table 14-3: Reference states that can be utilized in EES.

ReferenceID	Reference State	Notes
DFT	Default value associated with the fluid.	The default reference state is listed in the Fluid Property Information for each fluid
NBP	Specific enthalpy and specific entropy are set to 0 for saturated liquid at the Normal Boiling Point.	This option is not available for fluids with a critical pressure that is less than one atmosphere.[1]
ASH	Specific enthalpy and specific entropy are set to 0 for saturated liquid at -40°C (which is also -40°F). This is the ASHRAE standard reference state.	This option is not available for fluids with a critical temperature that is less than -40°C.[1]
IIR	Specific enthalpy is set to 200 kJ/kg and specific entropy is set to 1.0 kJ/kg-K for saturated liquid at 0°C. This is the International Institute of Refrigeration standard reference state.	This option is available for fluids with a critical temperature that is less than 0°C.[1]

1. In this case the reference state will be reset to the default value.

$UnitSystem

The $UnitSystem directive is discussed in Section 1.5 and provides a method for specifying the unit system settings in EES. All of the specifications that could be set in the Unit System tab of the Preferences dialog, shown in Figure 14-7, can be set using the $UnitSystem directive.

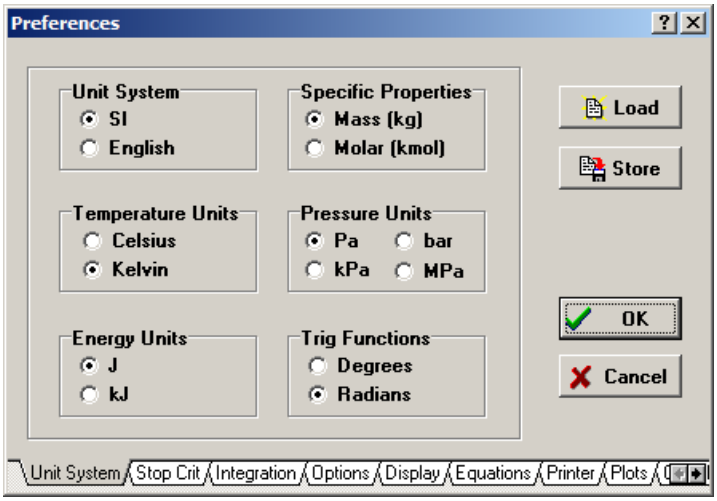


Figure 14-7: Unit System tab of the Preferences dialog.

The format for the $UnitSystem directive is:

$UnitSystem SI Mass (or Mole) Deg (or Rad) kPa (or Pa, bar, MPa) J (or kJ) C (or K)

or

$UnitSystem Eng Mass (or Mole) Deg (or Rad) psia (or atm) F (or R)

Note that it is not possible to specify a mixed unit system. For example, if the unit system is SI, then specifying F for the temperature units will result in temperature units of C, not F.

14.3 Directives for Code Segments

$Common

The $Common directive allows selected variables that are defined in the main EES program to be accessible from within functions, procedures, modules, or subprograms without providing them as input arguments. Note that common variables should never be used in library files. The $Common directive is discussed in Section 3.6 as a method for accessing elements of an array that exists in the main EES program from within a function or procedure.

The $Common directive is placed within the declaration of the function, procedure, module or subprogram of interest. Variables following the $Common directive and separated by commas (when using the decimal point as the decimal separator) or semicolons (when the comma is the decimal separator) are accessible within the code segment. For example, consider the function Test:

```
Function Test(A,B)
    $Common C,D
    Test=A+B+C+D
End

A=1
B=2
C=3
D=4
E=5
```

The function Test has access to variables A and B because they were passed as input arguments but it also has access to variables C and D through the $Common directive. Note that the value of a variable accessed using the $Common directive cannot be changed within the function (i.e., the information flow is only one-way, which is different from common variables in most formal programming languages). Modifying the function Test as shown below:

```
Function Test(A,B)
    $Common C,D
    Test=A+B+C+D
    C=2
End
```

will lead to the error message shown in Figure 14-8.

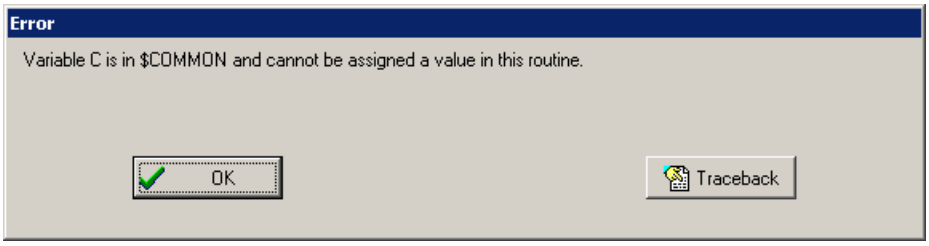


Figure 14-8: Error message associated with changing the value of a common variable within a function.

$Constant

The $Constant directive allows the specification of a user-defined constant that is available within a specific EES file, as discussed in Section 1.9. The calling protocol for the $Constant directive is:

```
$Constant Name# = ValueSI [UnitsSI] ValueEng [UnitsEng]
```

where Name# is the name of the constant, ValueSI and ValueEng are the values of the constant in the SI and English units specified by [UnitsSI] and [UnitsEng]. For example, a constant that provides the Bohr radius can be entered in EES according to:

```
$Constant a_0#=5.291772e-11 [m]  1.736146e-10 [ft]
```

Multiple constants can be defined using the $Constant directive; however, each constant must be defined on its own line and these directives should appear before the constant is used. Constants are global; that is, they are available in all functions, procedures, subprograms, and modules as well as the Main program.

Constants are useful to specify the size of an array that is passed to a function or procedure using array range notation, as discussed in Section 3.6. For example, the EES code below uses the constant N# to set the size of the array A[ ] that will be passed to the function SumSquares.

```
$Constant N# = 150                          "size of array"

Function SumSquares(A[1..N#])
    S:=0                                    "initialize sum"
    i:=1                                    "initialize index"
    Repeat
        S:=S+A[i]^2                         "sum the square of each element"
        i:=i+1                              "increment the element"
    Until (i>=N#)                           "stopping condition"
    SumSquares:=S                           "set the function name to the calculated value"
end
```

$RequiredOutputs

The $RequiredOutputs directive allows a procedure to be called with fewer outputs than are used in the declaration statement. The $RequiredOutputs directive must be placed within the procedure declaration. The format is:

```
$RequiredOutputs N
```

where N is the minimum number of allowable outputs that can be used when the procedure is called. If N is greater than the number of outputs used in the procedure declaration then the directive will be ignored.

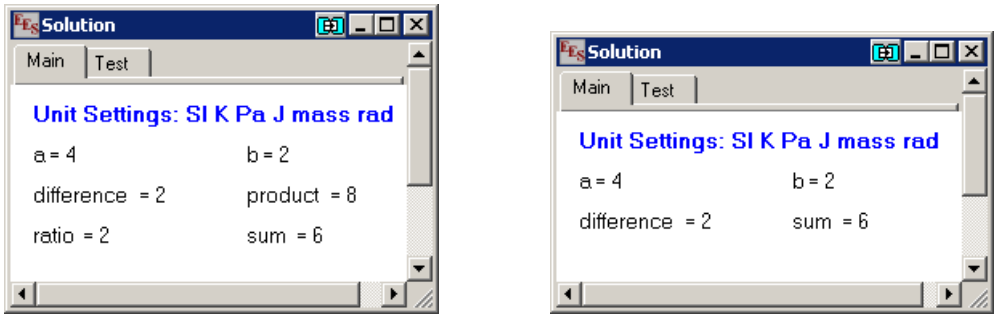
As an example, consider the procedure Test, which computes the sum, difference, product, and ratio of the two inputs:

```
Procedure Test(a,b: sum, difference, product, ratio)
    sum:=a+b
    difference:=a-b
    product:=a*b
    ratio=a/b
End
```

The procedure can be called as usual:

```
a=4
b=2
Call Test(a,b: sum, difference, product, ratio)
```

leading to the solution shown in Figure 14-9(a).



(a) **(b)**

Figure 14-9: Solution Window with (a) all outputs assigned and (b) only two outputs assigned.

If the procedure Test is called with fewer than the four declared outputs:

```
Call Test(a,b: sum, difference)
```

Then the error message shown in Figure 14-10 is obtained.

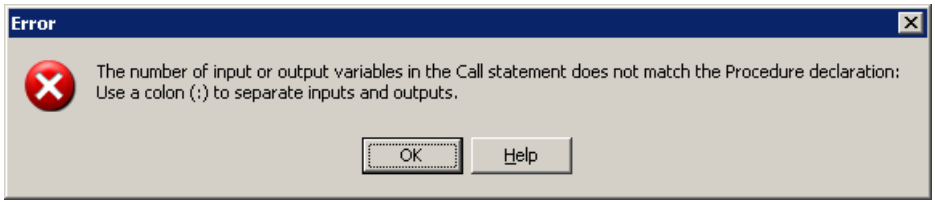


Figure 14-10: Error message associated with using too few output variables in the Call statement.

If the $RequiredOutputs directive is placed in the procedure:

```
Procedure Test(a,b: sum, difference, product, ratio)
    $RequiredOutputs 2
    sum:=a+b
    difference:=a-b
    product:=a*b
    ratio=a/b
End
```

Then there will not be an error message and the solution shown in Figure 14-9(b) is obtained.

14.4 Directives related to Complex Algebra

$Complex

EES can be configured to operate in complex mode so that each variable is assumed to have both a real and imaginary part. This capability is discussed in Chapter 13. The $Complex directive can be used to activate or deactivate this mode and also to specify whether the variable i or j is used to represent $\sqrt{-1}$. For example,

```
$Complex On j
```

activates complex algebra and indicates that j will be used as $\sqrt{-1}$.

$Real

The variables following the $Real directive are assumed to be real when EES is operating in complex mode. These variables cannot have a nonzero imaginary part and therefore only their real component can be manipulated. The $Real directive is discussed in Section 13.2.

14.5 Directives related to Saving & Copying Data

$CopyToLookup

The Lookup command was discussed in Section 1.8 as a means of accessing data contained in a Lookup Table. The Lookup command can also be used to place calculated results into a Lookup Table when it is called from within an internal function or procedure. The $CopyToLookup directive is available in the Professional version of EES and it provides a better way to move selected variables to a lookup table.

The calling protocol for the $CopyToLookup directive is:

```
$CopyToLookup /T /C /R 'TableName', 'ColumnName', StartRow, Var1, Var2, Var3[1..N]
```

The parameters 'TableName' and 'ColumnName' specify the table and the column where the variables will be written; these parameters can either be strings or string variables. The value of StartRow is the row in which the first data item will be written and it can either be an integer or a variable. Each additional data item will be written in a successive row. Data items to be written appear last and they are listed in the order that they are to be written to the table. Note that the array range notation, discussed in Section 1.7, can be used to specify a range of elements within an array.

There are several optional flags at the start of the directive. The /T flag indicates that a new table with the name 'TableName' should be created if it does not already exist. The /C flag indicates that a new column with the name 'ColumnName' should be created if it does not already exist. Finally, the flag /R indicates that additional rows should be added, if necessary, in order to write all of the data in the list. The $CopyToLookup directive can be used within the Main EES program and also within functions, procedures, and subprograms.

The EES code:

```
n=10
duplicate i=1,n
    X[i]=i^2
end
$CopyToLookup /T /C /R 'Data Table', 'Data 1', 1, X[1..n]
```

will result in the Lookup Table shown in Figure 14-11.

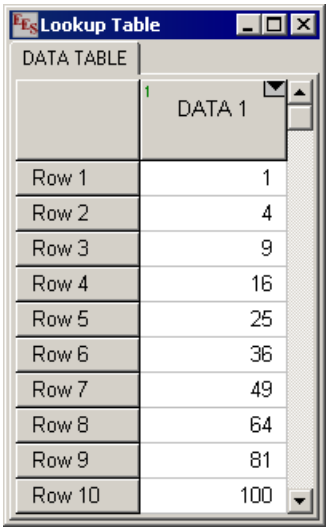


Figure 14-11: Lookup Table created by the $CopyToLookup directive.

$Export

The $Export directive allows variables within an EES program to be written to an ASCII file. This ASCII file can subsequently be read into EES by using the Open Lookup Table command from the Tables menu, as discussed in Section 1.8, or by using the $Import directive. The ASCII file can also be read into other applications, such as a spreadsheet program.

The format of the $Export directive is:

```
$Export /A /F /C /Q 'FileName', Variable 1:Format 1 Variable List 2:Format 2 Array[1..n]:Format 3
```

The parameter 'FileName' indicates the name of the file to which the variables are exported. The extension of the file name controls its format. Allowable extensions include .csv (comma-separated values, a format recognized by most other software), .dat (in which values are separated by a space), and .tab (in which values are separated by a tab). If the filename extension is .txt then EES will save the file in the Lookup File Format with header information. The header information will include the name of the variable, its units, and display format. The resulting file can be read directly using the Open Lookup Table command from the Tables menu. Note that if the file name is 'ClipBoard' then the variables will be placed as tab-delimited text onto the clipboard, from which they can be pasted to any other application.

These variables appearing in the $Export directive can be variable names, numbers, or constants. Array range notation, discussed in Section 1.7, can also be used. Each variable can optionally by followed with a colon and a format specification statement. The format statements indicate the type of formatting that should be used to write the variables in the variable list. The format selections are discussed in Section 1.2 and repeated here. If no format specification is present then the variables will be exported in floating point format with 8 decimal places (F8). The indicator Fn indicates floating point with the numerical parameter n indicating the number of decimal places. The format indicator En specifies exponential format where n is number of

significant figures. If the format specification is A then the format used with be the same as the format that is used to display the variable in the Solution Window of the EES program.

There are a number of optional flags at the beginning of the $Export statement. The /A flag indicates that the values should be appended to the end of the file, if one already exists. If the $Export command is used in an EES program that involves a Parametric Table, then by default the results associated with each run will be exported to the file. The /F flag indicates that only the results associated with the final run should be exported. The /C flag indicates that the file will be cleared before each export operation. The /C and /A options are mutually incompatible and should not be used together. If the /Q flag is used then the single quotes that normally surround strings will not be exported.

The $Export directive can be placed anywhere in your EES program including within functions, procedures, and subprograms. Multiple $Export directives will be evaluated in the order that they appear in the Equations Window.

$Import

The $Import directive will read a list of variables from an ASCII file. The file could have been written by another EES program, a different program, or by a previous execution of the current EES program using the $Export directive. The $Import and $Export directives provide a convenient means for communicating between EES and other software.

The format of the $Import directive is:

```
$Import 'FileName' /skiplines=n Var1  Var2  Array[1..N]  S$  …
```

The parameter 'FileName' indicates the name of the file from which the variables are imported; if the file name is 'ClipBoard' then the variables will be read from the clipboard in text format. The variable list includes variables that are used in the EES program. The values of these variables will be imported from the file. Array range notation and string variables are both supported. The /skiplines=n flag is optional; if used, the flag will cause the first n lines of the file to be skipped and the import process will begin on the next line.

The $Import directive can be placed anywhere within the Equations Window and will be executed before the calculations are initiated. The $Import directive will only execute once at the start of calculations even if the calculations are repeated within a Parametric Table. Multiple $Import directives will be evaluated in the order that they occur in the Equations Window.

One use for the $Import and $Export directives is to identify a periodic steady state solution to a problem that has a harmonic forcing function. For example, consider the ordinary differential equation:

$$\frac{dy}{dt} + 0.2\,y = 0.2\sin(t) \tag{14-1}$$

The forcing function, 0.2 sin(t), repeats every 2π seconds and therefore there must be a periodic

steady state solution to the differential equation that is independent of the initial condition. The solution will approach periodic steady state after a large number of cycles have been simulated. Periodic steady state has been achieved when the solution at $t = 2\pi$ is the same as $t = 0$ (i.e., the value of y is the same at the beginning and end of the cycle).

We can solve Eq. (14-1) numerically using the Integral command in EES, discussed in Section 7.2. An arbitrary initial condition, y_0, is assumed:

```
y_0=0.2                                  "initial condition"
```

and Eq. (14-1) is used to program the state equation:

```
dydt+0.2*y=0.2*sin(t)                    "ordinary differential equation"
```

The state equation is integrated from y_0 at $t = 0$ to y at $t = 2\pi$ numerically using the Integral command and the results are saved in an Integral Table and used to generate Figure 14-12.

$$y = y_0 + \int_{t=0}^{t=2\pi} \frac{dy}{dt} dt \tag{14-2}$$

```
y=y_0+Integral(dydt,t,0,2*pi)            "solution for one cycle"

$IntegralTable t, y                      "saved in an Integral table"
```

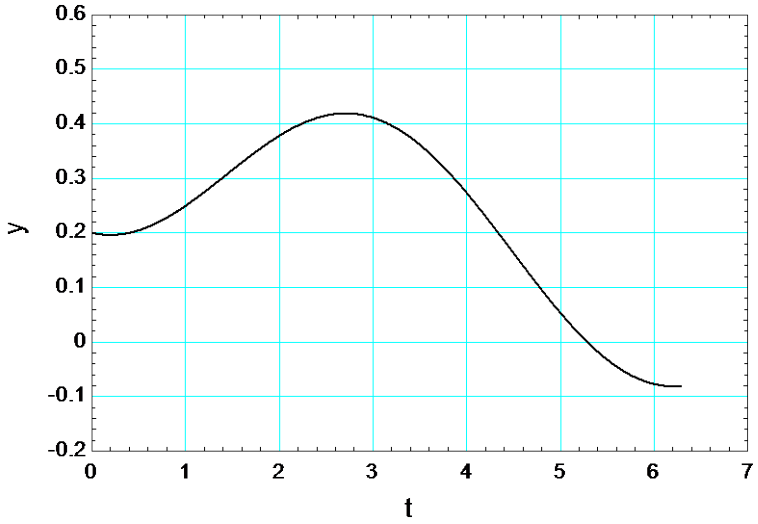


Figure 14-12: Solution for *y* as a function of *t* for one cycle starting from an arbitrary initial condition.

The solution shown in Figure 14-12 is not the periodic steady state solution because the final solution and initial condition are not the same. In order to obtain the periodic steady state solution, it is necessary to have the starting value of y equal to its final value. This solution can be found by using a $Import directive to obtain the initial condition, y_0, from the Clipboard and then using a $Export directive to export the final solution to the Clipboard. Copy the value 0.2 to

the Clipboard. Replace the specification of y_0 in the Equations Window with a $Import directive, as shown.

```
$Import 'ClipBoard' y_0

{y_0=0.2}                                  "initial condition"
dydt+0.2*y=0.2*sin(t)                      "ordinary differential equation"
y=y_0+Integral(dydt,t,0,2*pi)              "solution for one cycle"

$IntegralTable t, y                        "saved in an Integral table"
$Export 'ClipBoard' y
```

Repeated solutions of the EES program correspond to successive simulations of the cycle; each simulation begins where the last cycle ended. Figure 14-13 shows the solution for the first five cycles; notice that cycle five approaches the periodic steady state solution.

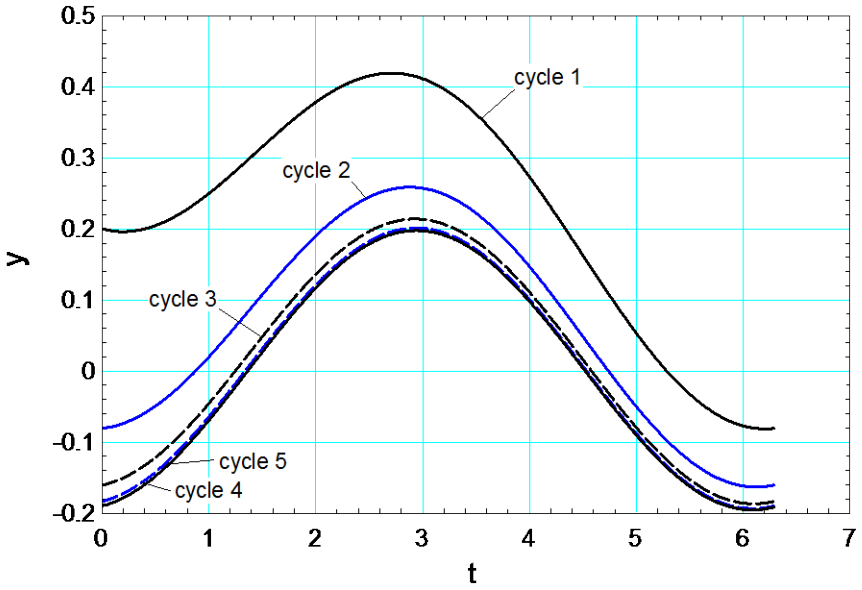


Figure 14-13: Solution for *y* as a function of *t* for several cycles, each starting with an initial condition that is equal to the final value of the previous cycle.

$OpenLookup

The $OpenLookup directive is available in the Professional version of EES. The $OpenLookup directive opens a file and reads the data contained in the file into a Lookup Table. The $OpenLookup directive allows the use of a format file (.fmt) to specify the structure and format of the data in the file. This capability allows data to be constructed from a wide variety of sources and subsequently read into a Lookup Table and conveniently used with EES programs.

The format of the $OpenLookup directive is:

```
$OpenLookup 'FileName' /Format = 'Format.fmt' /Rename = 'LookupTableName'
```

The string 'FileName' contains the name of the file that contains the data and it can either be a string or a string variable. The /Format flag is optional, but can be used to specify a format file (.fmt) that contains the format specification that should be used to read the file. The details of how to create a .fmt file can be found in the EES online help by searching for .fmt.

The /Rename flag is optional and can be used to rename the Lookup Table to the string 'LookupTableName'. If the /Rename flag is not present, then the Lookup Table will have the same name as the file containing the data, 'FileName'.

Note that when EES encounters the $OpenLookup directive it will immediately check to see if a Lookup Table with the specified name has already been opened. If the time/date information in the corresponding disk file has not changed then no action will be taken. Therefore, the $OpenLookup directive will not needlessly reconstruct the Lookup Table each time the equations are solved.

If a single question mark (?) is provided in place of the file name string 'FileName', then EES will allow the user to navigate to the file that is to be read using a standard Open File dialog. The resulting file name will be substituted for the ? so that the Open File dialog is not presented during every execution of the program. The use of a double question mark (??) will result in the Open File dialog being presented each time the equations are compiled and solved.

$SaveLookup

The $SaveLookup directive is available in the Professional version of EES. The $SaveLookup directive will save a specified Lookup Table to a disk file after the calculations are completed. The format for the directive is:

```
$SaveLookup 'LookupTableName' 'FileName'
```

where 'LookupTableName' is the name of the Lookup Table and 'FileName' is the name of the disk file. As with the $OpenLookup directive, the use of the ? or ?? symbols in place of the file name will open a dialog that allows the user to navigate to a selected file. A single question mark will use the file name that was first entered for each successive calculation, overwriting the contents each time the equations are solved. A double question mark will prompt the user to identify a file each time the equations are solved. Note that the $SaveLookup directive is not as generally useful as the $SaveTable directive and is only included for backward compatibility.

$SaveTable

The $SaveTable directive saves the contents of a selected table to a disk file upon completing calculations. The format is:

```
$SaveTable 'TableName' 'FileName' /A /E /F /N /T /Q
```

where 'TableName' is the name of the table (either a string or a string variable). The format for the table name string should be 'Type of Table:Name of Table'. Type of Table is either Parametric, Lookup, Arrays, or Integral depending on the type of table to be saved. Name of Table is the

name of the table to be saved. For example, to save the Parametric table named Table 1, 'TableName' would be 'Parametric:Table 1'. If no colon and name are provided then the table that is foremost when the $SaveTable directive is executed will be saved.

The parameter 'FileName' is the name of the disk file, which can be a string constant or string variable. The file name should include the extension, which will dictate the format that is used. Allowable formats are EES Lookup Table (.lkt), text file (.txt), and comma-separated values (.csv). The file name can also include directory information; if no directory information is provided then the file will be saved in the current directory which is typically the directory from which the EES file is read. Note that information about the current directory can be obtained using the EESFileDir$ function. As with the $SaveLookup directive, the use of the ? or ?? symbols in place of the file name will open a dialog that allows the user to navigate to a selected file. A single question mark will use that file name for each successive calculation, overwriting the contents each time the equations are solved. A double question mark will prompt the user to identify a file each time the equations are solved.

There are a number of optional flags that are included at the end of the directive; none of these can be used when you are saving to a disk file with Lookup Table (.lkt) format. The /A flag will cause the data to be appended to an existing file. The /E flag causes a .txt file to be saved in EES format so that it can be opened directly using the Open Lookup Table command from the Tables menu. The /F command causes results to only be exported from the last run of a Parametric Table. The /N command causes the header and unit information to be saved. The /T command transposes the rows and columns. The /Q command causes strings to be exported without single quotes.

14.5 Directives related to Program Execution & Debugging

$DoLast & $EndDoLast

Section 5.1 discusses the methodology used within EES to block and reorder equations sets in order to allow them to be solved most efficiently. This process occurs internally and is not under the control of the user. EES will compile all equations before beginning any calculations. It is sometimes desirable to perform a subset of calculations initially and then compile and solve the remaining calculations. For example, if one set of equations results in the computation of the limits of a duplicate loop or an integral, then these equations must be solved first. Another situation where this capability is useful is when you want to carry out calculations that involve all of the entries in a Parametric Table and therefore these calculations must be accomplished only after the table is solved.

Equations that are placed between a $DoLast and $EndDoLast directive will automatically be compiled and solved only after all of the other equations have been compiled and solved. Multiple pairs of $DoLast and $EndDoLast directives can be used, but they cannot be nested.

Consider a simple example. The size of an array (N) is to be determined by solving the following simultaneous set of equations:

$$a + N + b = 5 \tag{14-3}$$

$$2N - 3 + a = 10 \tag{14-4}$$

$$b = 2 \tag{14-5}$$

```
a+N+b=5
2*N-3+a=10
b=2
```

The array is defined according to:

$$A_i = i^2 \quad \text{for } i = 1..N \tag{14-6}$$

```
duplicate i=1,N
    A[i]=i^2
end
```

Solving this set of equations will lead to the error message shown in Figure 14-14.

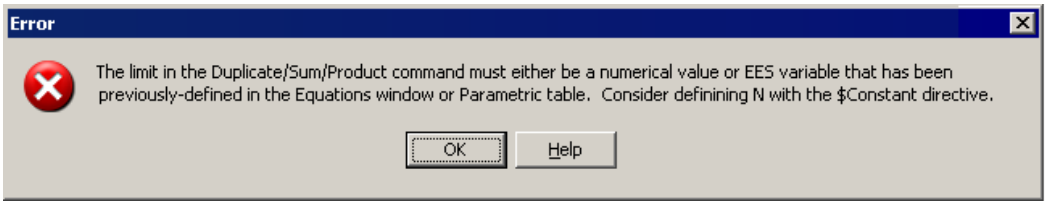


Figure 14-14: Error message.

The equations required to determine N must be compiled and solved before the Duplicate loop is compiled. The problem can be solved by placing the Duplicate statement between a $DoLast and a $EndDoLast directive.

```
a+N+b=5
2*N-3+a=10
b=2

$DoLast
duplicate i=1,N
    A[i]=i^2
end
$EndDoLast
```

$If, $IfNot, $Else & $EndIf

The $If, $IfNot, $Else, and $EndIf directives allow sets of equations in the Equations Window to be activated or deactivated depending on various conditional statements. These conditional statements can involve the OR keyword. The conditional statements that are recognized by EES are summarized in Table 14-4.

Table 14-4: Recognized conditional keywords.

Conditional statement	Description
DiagramWindow	This condition is true if the main Diagram Window (discussed in Chapter 15) is displayed on the screen. In this condition, any input variables that are defined on the Diagram Window will be used while compiling the equations. If the Diagram Window is hidden (DiagramWindow = False) then any input variables defined in the Diagram Window will be ignored.
IntegralTable	This condition is true if a $IntegralTable directive appears in the EES file; the $IntegralTable directive is discussed in Section 7.2.
Macro	This condition is true if calculations are initiated by running or playing a Macro command list. Macros are discussed in Chapter 19.
Min/Max (or MinMax)	This condition is true if calculations have been initiated with the Min/Max command. The Min/Max command will carry out optimization, as discussed in Chapter 6.
Min/MaxTable (or MinMaxTable)	This condition is true if calculations have been initiated with the Min/Max Table command. The Min/Max Table is discussed in Section 6.2.
Min/MaxTable (or MinMaxTable) = 'TableName'	This condition is true if calculations have been initiated with the Min/Max Table command from the Parametric Table named 'TableName'. String variables for 'TableName' are not accepted.
ParametricTable	This condition is true if calculations have been initiated with the Solve Table (see Section 1.3), Min/Max Table (Section 6.2), or Uncertainty Propagation Table (Section 8.3) command.
ParametricTable = 'TableName'	This condition is true if calculations have been initiated with the Solve Table, Min/Max Table, or Uncertainty Propagation Table command and the Parametric Table name is 'TableName'. String variables for 'TableName' are not accepted.
TableRun# = (or < or >) Value	This condition is true if calculations have been initiated with the Solve Table, Min/Max Table, or Uncertainty Propagation Table command and the row that is currently being solved is equal to (or less than or greater than) the value specified by Value. Value must be an integer number (not a variable). The use of this condition requires that EES recompile the equations after each row of the Parametric Table is solved which may reduce computation speed.
UnitSystem('XX')	This condition is true if the unit system setting in EES (discussed in Section 1.5) are consistent with the value 'XX'. The value 'XX' can correspond to any of the unit specifications that can be specified using the Unit System tab of the Preferences dialog or with the $UnitSystem directive. These include: 'SI', 'Eng', 'Mass', 'Molar', 'Deg', 'Rad', 'kPa', 'Pa', 'MPa', 'bar', 'psia', 'atm', 'C', 'K', 'F', 'R', 'kJ'
StringVariable$ = 'String Constant'	StringVariable$ corresponds to a string variable that is set before the $If directive either in the Equations Window or the Diagram Window. This condition allows you to activate or deactivate sets of variable by simply changing the value of a string variable.
Uncertainty	This condition is true if calculations have been initiated with the Uncertainty Propagation command, as discussed in Section 8.2.
UncertaintyTable	This condition is true if calculations have been initiated with Uncertainty Propagation Table (Section 8.3) command.
UncertaintyTable = 'TableName'	This condition is true if calculations have been initiated with the Uncertainty Propagation Table command and the name of the table is 'TableName'. String variables for 'TableName' are not accepted.

The $If directive is extremely useful in a variety of contexts. We used the command $If ParametricTable in Section 1.3 to set variables that are used with the Solve command but not with the Solve Table command. The $If directive removed the equations in the Equations Window that specify the values of variables that were being parametrically varied by solving a Parametric Table.

The $If directive is also useful for controlling the execution of a program so that you can gradually add sophistication and complexity while maintaining reasonable guess values and therefore assuring convergence. The process that EES uses to solve a nonlinear set of equations and the importance of guess values is discussed in Chapter 5. Often it is possible to solve a simple, or linearized, version of a problem in order to establish reasonable guess values for a more complicated, nonlinear version of the same problem. Rather than rewriting the EES code, the $If directive allows the program to switch between a linear and nonlinear version by changing the value of a string variable.

As a simple example, consider a cylindrical conductor that carries *current* = 70 amp along its axis. The conductor is D = 4 mm in diameter and L = 1 m long. The ends of the conductor are maintained at T_H = 293.2 K. The outer surface of the conductor is insulated from convection or radiation. The conductor has constant electrical resistivity ρ_e = 1x10^{-8} ohm-m and a temperature dependent conductivity:

$$k(T) = 150\left[\frac{\text{W}}{\text{m-K}}\right] - 0.1\left[\frac{\text{W}}{\text{m-K}^2}\right](T - 300[\text{K}]) \tag{14-7}$$

The inputs are entered in EES:

```
$UnitSystem SI Mass J K Pa Rad
D=4.0 [mm]*convert(mm,m)                "diameter"
L=1 [m]                                 "length"
T_H=293.2 [K]                           "end temperature"
current=70 [amp]                        "current"
rho_e=1e-8 [ohm-m]                      "electrical resistivity"
```

The cross-sectional area of the conductor is:

$$A_c = \pi\frac{D^2}{4} \tag{14-8}$$

```
A_c=pi*D^2/4                            "cross-sectional area"
```

The temperature distribution can be accomplished numerically using the method discussed in Nellis and Klein (2009). Nodes are positioned linearly along the axis of the conductor; the axial location of each node (x) is given by:

$$x_i = i\frac{L}{(N-1)} \quad \text{for } i = 1..N \tag{14-9}$$

where N is the number of nodes. The distance between adjacent nodes is:

$$\Delta x = \frac{L}{(N-1)} \tag{14-10}$$

```
N=10 [-]                                    "number of temperature nodes"
Duplicate i=1,N
      x[i]=L*(i-1)/(N-1)                    "axial position"
end
Dx=L/(N-1)                                  "differential length"
```

The temperature of the end nodes are set to the specified end temperature:

```
T[1]=T_H
T[N]=T_H
```

The temperatures of the inner nodes, $i = 2$ to $i = N - 1$, are obtained using energy balances. The rate of ohmic dissipation within each node must be balanced by the rate of conduction out of each node:

$$\frac{\rho_e \Delta x\, current^2}{A_c} = \frac{k_{LHS,i} A_c (T_i - T_{i-1})}{\Delta x} + \frac{k_{RHS,i} A_c (T_i - T_{i-1})}{\Delta x} \quad \text{for } i = 2..(N-1) \tag{14-11}$$

where k_{LHS} is the conductivity evaluated at the average of the nodal temperature (T_i) and the temperature of the node to the left (T_{i-1}) and k_{RHS} is the conductivity evaluated at the average of T_i and T_{i+1}:

$$k_{LHS,i} = \frac{(T_i + T_{i-1})}{2} \quad \text{for } i = 2..(N-1) \tag{14-12}$$

$$k_{RHS,i} = \frac{(T_i + T_{i+1})}{2} \quad \text{for } i = 2..(N-1) \tag{14-13}$$

These equations are implemented using a Duplicate loop:

```
Duplicate i=2,(N-1)
      k_LHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i-1]+T[i])/2-300 [K])
      k_RHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i+1]+T[i])/2-300 [K])
      rho_e*Dx*current^2/A_c=k_LHS[i]*A_c*(T[i]-T[i-1])/Dx+k_RHS[i]*A_c*(T[i]-T[i+1])/Dx
end
```

Solving leads to the Arrays Table shown in Figure 14-15(a). Notice that the solution is nonphysical and involves negative conductivities and very large temperatures; this solution occurs because the guess value for each of the temperatures in the array are set to their default value (1 K).

One method of approaching this problem is to solve the problem using a constant value of conductivity, which results in a linear set of equations. The solution to the linear set of equations

can be used as reasonable guess values from which to start the solution to the nonlinear set of equations. Define a string variable Mode$ that will be set to 'nonlinear' or 'linear' depending on the equation set to be solved.

```
Mode$='linear'
```

If the value of Mode$ is set to 'linear' then the conductivities are set to constant values using the $If directive. Otherwise the temperature dependent conductivities are used.

```
$If Mode$='Linear'
    k_LHS[i]=50 [W/m-K]
    k_RHS[i]=50 [W/m-K]
$Else
    k_LHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i-1]+T[i])/2-300 [K])
    k_RHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i+1]+T[i])/2-300 [K])
$EndIf
```

Arrays Table

Main

Sort	1 x_i [m]	2 T_i [K]	3 $k_{LHS,i}$ [W/m-K]	4 $k_{RHS,i}$ [W/m-K]
[1]	0	293.2		
[2]	0.1111	3201	5.269	-135.9
[3]	0.2222	3117	-135.9	-128.7
[4]	0.3333	3057	-128.7	-124.2
[5]	0.4444	3026	-124.2	-122.6
[6]	0.5556	3026	-122.6	-124.2
[7]	0.6667	3057	-124.2	-128.7
[8]	0.7778	3117	-128.7	-135.9
[9]	0.8889	3201	-135.9	5.269
[10]	1	293.2		

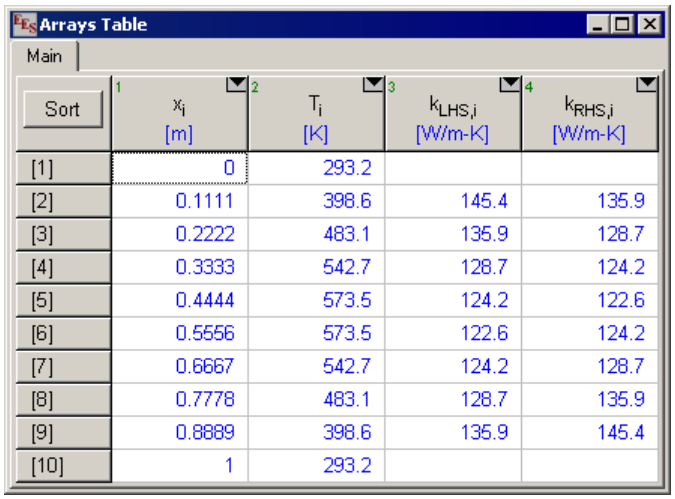
(a)

Arrays Table

Main

Sort	1 x_i [m]	2 T_i [K]	3 $k_{LHS,i}$ [W/m-K]	4 $k_{RHS,i}$ [W/m-K]
[1]	0	293.2		
[2]	0.1111	599.7	50	50
[3]	0.2222	829.5	50	50
[4]	0.3333	982.7	50	50
[5]	0.4444	1059	50	50
[6]	0.5556	1059	50	50
[7]	0.6667	982.7	50	50
[8]	0.7778	829.5	50	50
[9]	0.8889	599.7	50	50
[10]	1	293.2		

(b)



(c)

Figure 14-15: Arrays Table showing the solution of (a) the nonlinear equations solve with the guess values set to their default values, (b) the solution to the linear equations, and (c) the solution to the nonlinear equations with guess values set to the linear equations solution.

Note that the equation set that is not used is grayed out and not compiled. The solution to the linear set of equations is shown in Figure 14-15(b); the solution set is reasonable regardless of guess values since the equations are linear. Select Update Guess Values from the Calculate menu in order to set the guess values for each nodal temperature to the linear solution. Then switch the value of Mode$ to 'nonlinear'.

```
Mode$='nonlinear'
```

Note that the equations used to calculate conductivity are switched:

```
$If Mode$='Linear'
    k_LHS[i]=50 [W/m-K]
    k_RHS[i]=50 [W/m-K]
$Else
    k_LHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i-1]+T[i])/2-300 [K])
    k_RHS[i]=150 [W/m-K]-0.1 [W/m-K^2]*((T[i+1]+T[i])/2-300 [K])
$EndIf
```

and the solution shown in Figure 14-15(c) is obtained. The more reasonable guess values have resulted in the correct solution to the nonlinear equations.

$Include

The $Include directive automatically loads various types of external information into an EES file. The $Include directive should be placed at the top of the EES Equations Window in order to ensure that it is processed before the equations are compiled. The format is:

```
$Include FileName
```

where FileName indicates the file that contains the information to be loaded. The file name must include the extension and the complete path name.

There are various types of information that can be included using the $Include directive. A .txt file can be loaded that includes EES equations that will be compiled together with those in the Equations Window. Note that the included equations must not themselves utilize a $Include directive. The $Include directive can also be used to automatically load a library file that contains a set of internal functions and procedures (.lib), as discussed in Chapter 11, or a library file that contains a set of external functions (.dlf) and procedures (.fdl or .dlp), as discussed in Chapter 19.

The process of defining new units in a .unt file is discussed in Section 1.5. The .unt file can be included with a $Include directive. The process of defining and storing your preferences indicating how EES is configured is discussed in Section 1.9. The preferences can be stored as a .prf file and subsequently automatically loaded using a $Include directive. This option is particularly useful if you are running EES on a network installation in which you may not have control of or authorization to change the default .prf file.

In the Professional version of EES, the $Include directive can be used to load a variable information file (.var) that is either written using the Save Inputs button in the Diagram Window or the Save option in the Variable Information dialog. The $Include directive can also be used to automatically load a macro file (.emf); macros are discussed in Chapter 18.

$IntegralAutoStep

The $IntegralAutoStep directive allows the user to specify the parameters that are used to automatically control the size of the integration steps in a numerical integration accomplished with the Integral command. This directive is discussed in Section 7.2 and allows the user to specify the same parameters that would otherwise be controlled in the Integration tab of the Preferences dialog.

$IntegralTable

The $IntegralTable directive controls the creation of an Integral Table which contains the integration variable and any other variables of interest that are computed during the numerical evaluation of an integral with the Integral command. The $IntegralTable directive is discussed in Section 7.2.

$MaxCalls

The $MaxCalls directive can be placed within a function or procedure in order to limit the number of times that it is called. The format is:

```
$MaxCalls = N
```

where N is the maximum allowable number of times that the function can be called (N must be greater than 2). The $MaxCalls directive should be placed immediately after the declaration of the function or procedure.

The purpose of this directive is to eliminate convergence problems that can otherwise occur due to control decisions made using the logic statements that can be employed within a function or procedure. In some problems, these logic statements will lead to repeated on/off decisions that never terminate. The $MaxCalls directive eliminates this issue; if the number of calls to the function or procedure exceeds the value set by the $MaxCalls directive then the values returned will be the same as were returned for the last valid call. If a Parametric Table is being solved, then the parameter N establishes the maximum number of times a function or procedure can be called for each row.

$StopCriteria

The $StopCriteria allows the user to specify the stop criteria that control the iterative calculation process used by EES to solve the equations. The parameters that can be set using this directive are the same as those that can be specified from the Stop Crit tab of the Preferences dialog, shown in Figure 14-16.

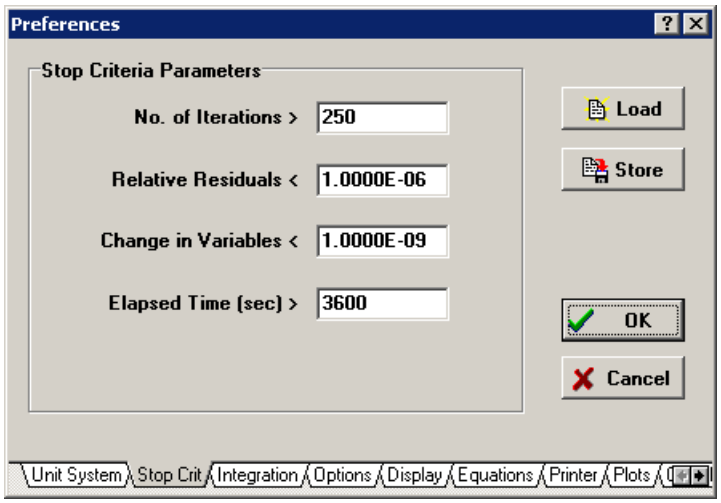


Figure 14-16: Stop Crit tab of the Preferences dialog.

The $StopCriteria directive corresponding to the stop criteria settings shown in Figure 14-16 is:

```
$StopCriteria Iterations=250 Residuals=1e-6 Variables=1e-9 Time=3600
```

Note that it is not necessary to provide all of the parameters; any parameter that is not specified in the $StopCriteria directive will be set according to the default values or the values set in the Stop Crit tab of the Preferences dialog. Settings made using the $StopCriteria directive will override those made in the Stop Crit tab of the Preferences dialog.

$Trace

The $Trace directive produces a table that contains the values of specified variables that occur during the iterative solution process to the equation set. The $Trace directive allows the iterative solution process used by EES to be examined. The $Trace directive is discussed in detail in Section 5.4 and is primarily used for debugging. The format of the directive is:

```
$Trace /N_i Variable1, Variable2, ..., Variable N
```

where Variable1, Variable2, etc. are the variables in the Main EES program that are to be traced. The parameter N_i is the maximum number of iterations that will be recorded. The variables are placed in a Lookup Table that has the name Trace.

$UpdateGuesses

The $UpdateGuesses directive causes the guess values to be automatically updated after the calculations are completed. The $UpdateGuesses directive operates only within the scope of the program unit in which it is located in (either the main program or a subprogram).

References

Nellis, G.F. and S.A. Klein, *Heat Transfer*, Cambridge, New York, (2009).

15 THE DIAGRAM WINDOW

The Diagram Window provides the capability to easily develop a user-friendly, graphical interface to an EES program. The Diagram Window provides a screen to place a schematic or graphic. The graphic can be prepared using the drawing tools provided in the Diagram Window tool bar or it can consist of one or more items copied from any drawing program. With little effort, text can be placed at any location on the Diagram Window to display calculated information from the EES program. Inputs to the calculations can also be entered directly in the Diagram Window, instead of using equations in the Equations Window. Calculations and other operations can be initiated and controlled from the Diagram Window using buttons.

The Professional version of EES greatly extends the capabilities of the Diagram Window by allowing drop-down lists, radio button groups, check boxes and slider controls as alternative input methods. Alternative output methods include live plots that update when the calculations are completed. The Professional version allows Child Diagram Windows that are activated by "hot areas" or using buttons. Additional types of buttons can be placed on the Diagram windows to load or save inputs/outputs, print, start a macro (see Chapter 19) or link to other programs. The Diagram Window is the key to the animation capabilities in EES that are described in Chapter 16 and the Executable/Distributable programs that are described in Chapter 17.

15.1 Placing Graphic Objects in the Diagram Window

Start EES and bring the Diagram Window to the front either by selecting Diagram Window from the Windows menu or by entering Ctrl-D.

Development and Application Modes

The Diagram Window and Child Diagram Windows (discussed in Section 15.5) operate in two modes that are referred to as the *development mode* and the *application mode*. When the tool bar shown in Figure 15-1 is visible, the Diagram Window is in development mode. If the tool bar is not visible then the Diagram Window is in application mode. The tool bar can be made visible or hidden using the Show/Hide Diagram Tool Bar command from the Options menu, by clicking the Diagram Window speed button  that appears on the speed bar, or by entering Ctrl-D. Clicking the small X at the upper right corner of the Diagram Window tool bar hides the toolbar and places the Diagram Window into application mode. The function of each of the buttons in the tool bar is indicated in Figure 15-1 and described in detail in this section.

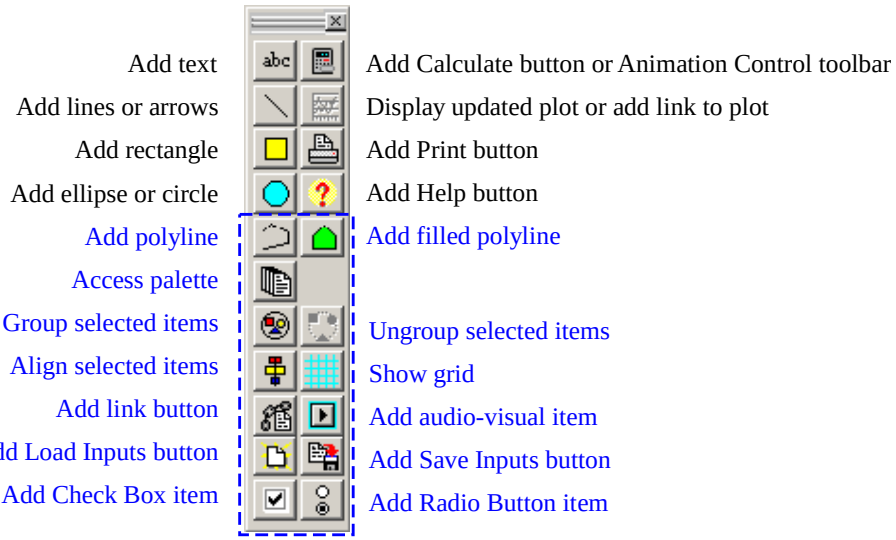


Figure 15-1: Diagram Window tool bar (items in dotted rectangle are only available in the Professional version of EES).

The Diagram Window must be in development mode in order to make any changes to it. Text and graphic items can be placed in the Diagram Window. Existing text and graphic items can be moved, modified, or deleted as discussed in this section. The Diagram tool bar shown in Figure 15-1 can be moved to any location by pressing the left mouse button down while the cursor is positioned at the top of the tool bar and then moving the mouse to drag the tool bar. A status bar is visible at the bottom of the Diagram Window when it is in development mode. The status bar shows the cursor position, the Diagram Window identity and information about the object on which the cursor is positioned on.

Graphical Objects from a Drawing Program

There are two ways to place graphical objects on the Diagram Window. One method is to create an object using any drawing program (e.g., PowerPoint or Canvas) and then use the commands in the drawing program to select and copy the item(s) to the clipboard. Return to EES and use the Paste command in EES to place the object(s) into the Diagram Window. The Commercial version will expect to find the graphic object on the clipboard as an enhanced metafile.

In the Professional version, you will see the Paste format dialog shown in Figure 15-2 if the graphic object is available in more than one format. In most cases, the device independent bitmap or the enhanced metafile are the best choices.

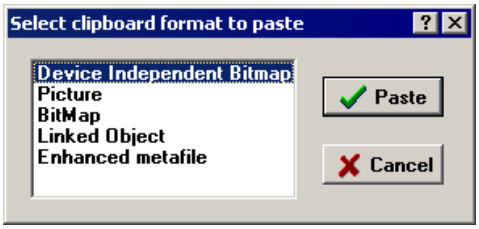


Figure 15-2: Paste format dialog.

Selecting the Paste button will place the graphic object on the Diagram Window. As an example, the throttle schematic shown in Figure 15-3 was created in a drawing program and pasted into the Diagram Window. The object can then be dragged to any location on the Diagram Window by pressing and holding the left mouse button down within a rectangle that encloses the object while sliding the mouse to a new location.

The object can be resized after it has been pasted into the Diagram Window. Clicking on the schematic will result in the display of 8 small boxes or handles; these are visible in Figure 15-3. Move the cursor to any of the boxes and then press and hold the left mouse button down. The cursor will change to indicate that you are resizing the object. Now move the mouse to a new location while holding the button down. The schematic will be resized. If the Shift key is depressed during this process then the aspect ratio of the object will remain unchanged. The graphic object can be deleted by selecting it and then pressing the delete key.

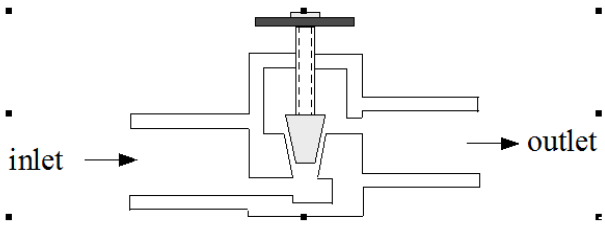


Figure 15-3: Throttle schematic created in a drawing program and copied into the Diagram Window.

Graphical Objects Drawn in the Diagram Window

A second way to create a drawing is to use the drawing tools provided on the Diagram Window toolbar. The toolbar provides graphic primitives such as lines/arrows ( ), ellipses ( ), and rectangles ( ).

The Professional version also provides tools for polylines ( ) and polygons ( ).

Drawing and Modifying Lines and Arrows

To place a line on the Diagram Window, click on the line button in the Diagram Window toolbar ( ). When you do this, the button will appear depressed and the cursor will change from the pointer to a plus sign. Move the cursor to the location that you want the line to start. Press and hold the left mouse button down as you move the mouse so that the cursor is at the position where the line should end, as shown in Figure 15-4. Then release the mouse button and the line will be drawn. If you hold the Shift key down while drawing the line, the line is forced to be drawn at an angle that is divisible by 45°.

After the line has been drawn, it can be moved to any location on the Diagram Window by pressing and holding the left mouse button down while the cursor is positioned somewhere in the middle of the line. Moving the mouse to a new location while holding the left button down will move the line. The location of the starting or ending point of the line can also be changed. Once drawn, the line can be selected by clicking the left mouse button in the middle of the line. The selected line will have small boxes (handles) at the ends, as shown in Figure 15-5. Move the

cursor to the box at the end of the line that you wish to relocate. Now press and hold the left mouse button down on the line handle while moving it to its new location.

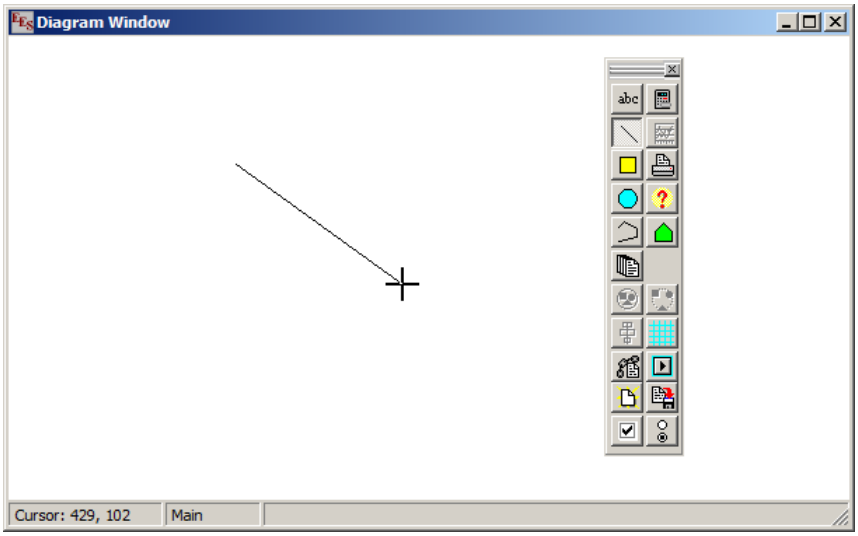


Figure 15-4: Placing a line on the Diagram Window.

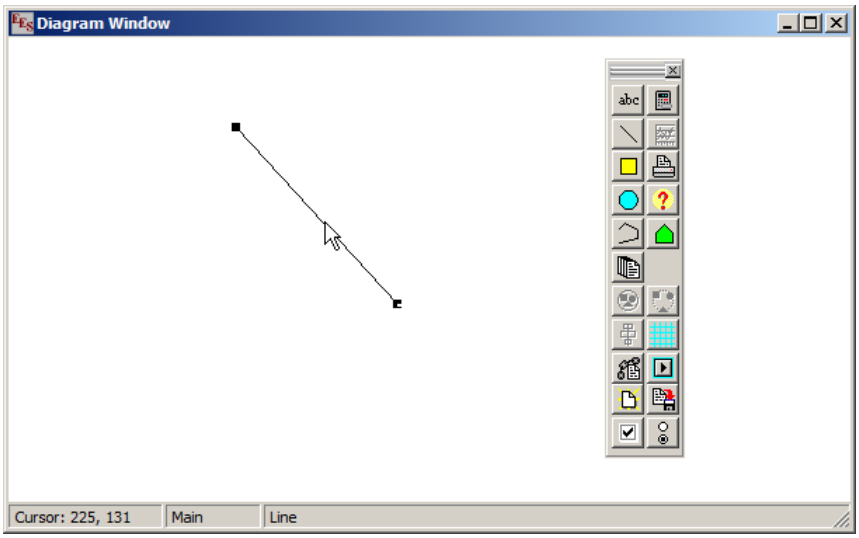


Figure 15-5: Selected line showing handles at its ends.

Each new line that is drawn will have the characteristics of the previous line. The first line that is drawn will be a solid thin black line. To change the characteristics of the line, either double-click the left mouse button or click the right mouse button while the cursor is located somewhere near the line center in order to activate a pop-up menu that contains various options, as shown in Figure 15-6. Note that the line can be cut or copied and its position relative to other graphic objects (i.e., backwards or forwards, corresponding to behind or in front of, respectively) can be controlled with options in the pop-up menu. Select the Properties menu item to bring up the Diagram Line Characteristics dialog shown in Figure 15-7.

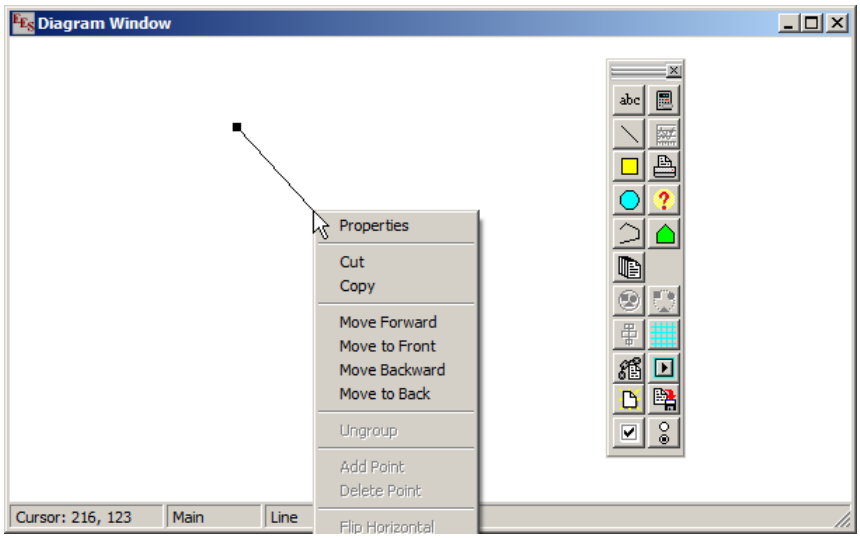


Figure 15-6: Pop-up menu resulting from double-clicking or right-clicking on a line.

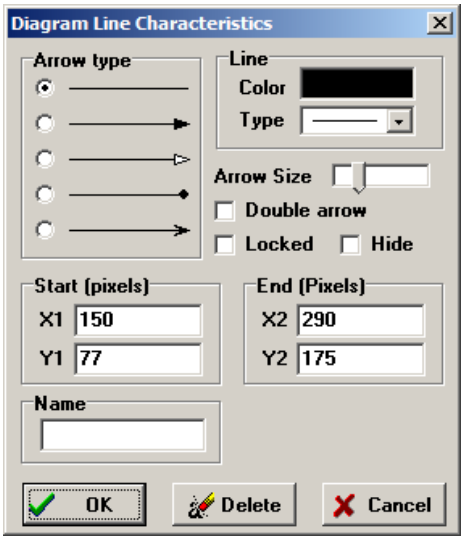


Figure 15-7: Diagram Line Characteristics dialog.

The color, line type (thick, dotted, dashed, etc.) and other characteristics of the selected line can be modified by changing the appropriate controls. The line can be made to appear as a single or double-headed arrow. The precise location of the start and end of the line can be entered in pixel coordinates. If the Locked button is selected then the position of the line cannot be altered with the cursor.

The Name field is also visible in the Diagram Line Characteristics dialog in the Professional version of EES. The Name field is used to provide animation effects, as described in Chapter 16. Animation may simply be the ability to hide or show the line.

Drawing and Modifying Rectangles

A rectangle with sharp or rounded corners can be placed on the Diagram Window by clicking on the Add Rectangle button in Diagram Window toolbar ( ) shown in Figure 15-1. After clicking this button, the cursor will change from a pointer to a plus sign. Move the mouse to position the cursor at the upper left corner of the rectangle and press and hold the left mouse button down. Next, move the cursor to the bottom right corner of the rectangle and release the mouse button. The rectangle will appear with its handles visible, indicating that it has been selected. The rectangle can be moved to another location by pressing and holding the left mouse button down at a location within the rectangle and moving the mouse while the button remains depressed. The length and/or width of the rectangle can be changed by pressing and holding the left mouse button down on one of the eight handles. This action will cause the cursor to change from the pointer to a double-arrow, as shown in Figure 15-8. While holding the left mouse button down, move the mouse to relocate the position of the handle. Note that the aspect ratio of the rectangle will remain constant if the Shift key is held down during this process.

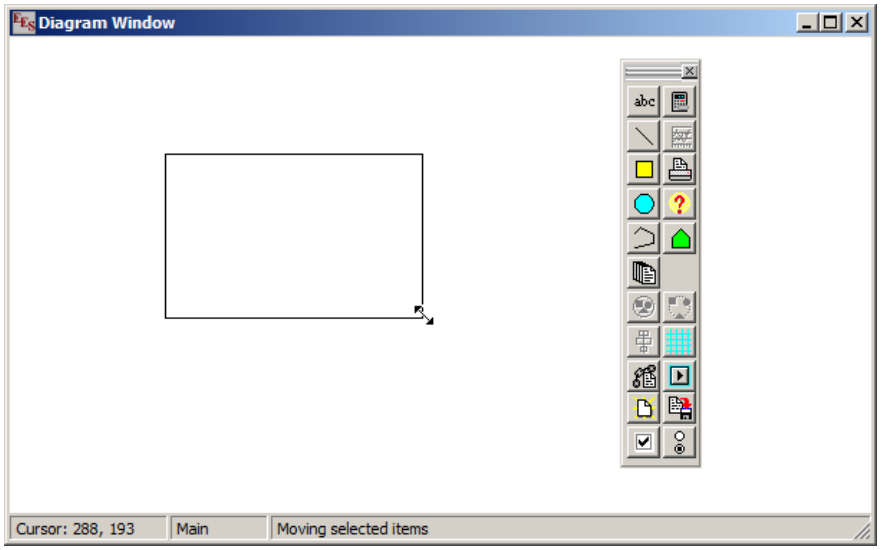


Figure 15-8: Changing the length or width of a rectangle.

The rectangle that is created will have the characteristics of the most recently created rectangle. The first rectangle that is placed on the Diagram Window will be drawn with a thin black border having square corners and no fill, as shown in Figure 15-8. The characteristics of the rectangle can be changed by double-clicking the left mouse button or clicking the right mouse button anywhere within the rectangle. Clicking the right mouse button displays a pop-up menu, shown in Figure 15-9. The rectangle can be cut or copied and then pasted into the Diagram Window of another instance of EES or into a Child Diagram Window (in the Professional version). The position relative to other graphic items (in front of or behind) can be adjusted. Selecting Properties from the pop-up menu will bring up the Diagram Object Characteristics dialog shown in Figure 15-10.

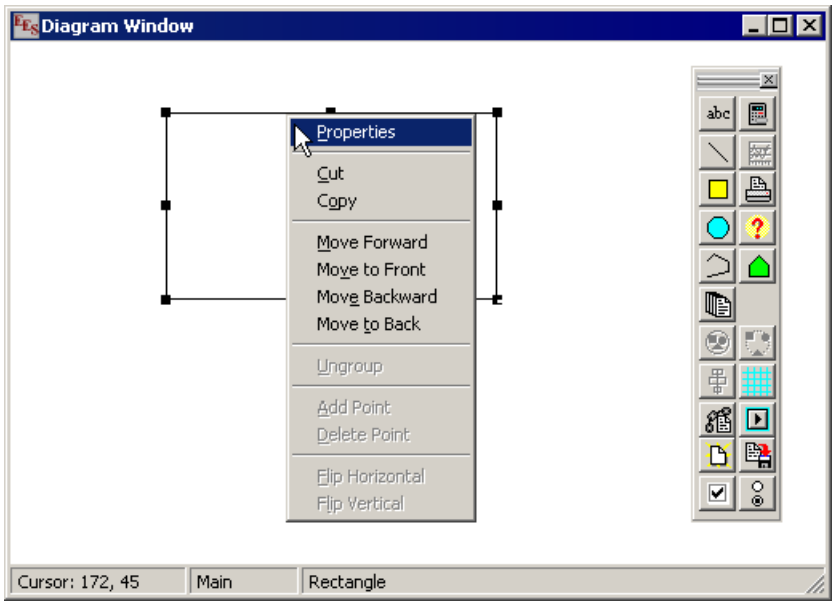


Figure 15-9: Pop-up menu resulting from right-clicking on the rectangle border.

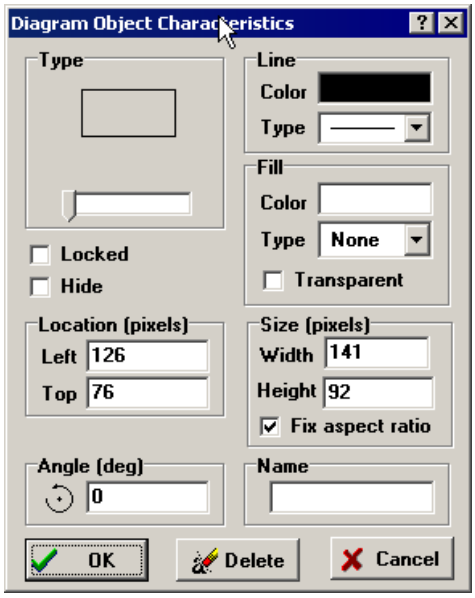


Figure 15-10: Diagram Object Characteristics dialog.

The slider control at the upper left in the Type box controls the sharpness of the corners; sliding the control to the right increases the rounding of the corners. The line color and line type can be selected from the Line box at the upper right. The Fill box provides options for the manner in which the rectangle object is filled. Select a color for the fill by clicking in the color box. Then select the fill type. If the Transparent check box is selected then the fill will be drawn in semi-transparent manner so that objects that are behind the rectangle can be seen. The exact location of the top-left corner and the width and height of the rectangle can be entered. Note that if the Fix aspect ratio check box is selected, as shown in Figure 15-10, then the width will change when the height is changed and vice-versa. To enter the width and height independently, uncheck the Fix aspect ratio check box.

The Angle and Name inputs are provided in the Professional version. These inputs are used for animation and for control of the object with EES variables, as explained in Chapter 16.

Drawing and Modifying Ellipses or Circles

An ellipse or circle can be placed on the Diagram Window using the Add Ellipse or Circle button ( ) on the Diagram Window toolbar shown in Figure 15-1. After clicking the button, move the mouse cursor to the desired upper left location of the object. Press and hold the left mouse button down as you move the cursor to the lower right location. If the Shift key is depressed, the object that is drawn will be a circle; otherwise it will be an ellipse. The ellipse/circle can be moved to a different location or resized in the same manner as described for a rectangle object. Double-clicking the left mouse button or clicking the right mouse button within the ellipse/circle will display the pop-up menu shown in Figure 15-9. Selecting the Properties menu item will access the Diagram Object Characteristics dialog, similar to the one shown in Figure 15-10, where the characteristics of the ellipse/circle object can be viewed or changed.

Drawing and Modifying Polylines and Polygons

The Professional version of EES allows polyline and polygon objects to be placed on the Diagram Window. Polyline objects consist of one or more connected line segments that can optionally be smoothed to form a Bezier curve. A polygon object consists of three or more line segments that enclose a space that can be filled with a color or pattern.

Click the Add Polyline button shown in Figure 15-1 ( ) to place a polyline or Bezier curve on the Diagram Window. The cursor will change from a pointer to a plus sign. Move the cursor to the desired location of the first point. The polyline object can be moved later, if desired, so selecting the exact position of the point is not necessary. After clicking on the starting point, move the cursor to the second point. A line will be displayed from the last point to the current cursor location. Click to position the second point. Continue this process until all of the segments of the polyline are drawn. Double-click (or press the Enter key) to end the drawing process. If you wish to abort the drawing process, press the Esc key.

Once the drawing process is completed, the polyline object can be selected by clicking the left mouse button while the cursor is positioned near the center of any line segment. The selected polyline will include handles at each of the vertices of the polyline. The entire polyline can be moved to a new location by pressing and holding the left mouse button near the center of any line segment and moving the cursor. The location of individual points composing the polyline can also be moved by selecting any one of the vertex handles of the polyline until the cursor changes to up-down-left-right arrows, as shown in Figure 15-11. While holding the left mouse button down, move the cursor to a new location. The position of the selected vertex will move with the mouse.

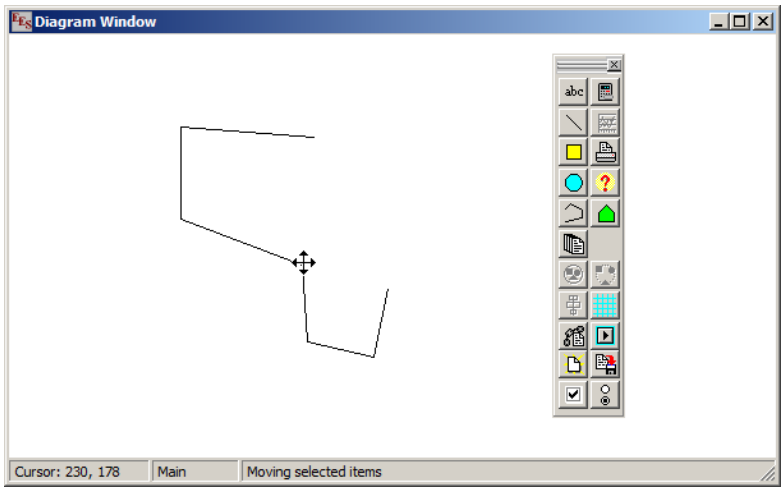


Figure 15-11: Relocating a vertex of a polyline.

Vertices can be added or deleted from an existing polyline by right-clicking on any vertex. The pop-up menu shown in Figure 15-9 will appear with the Add Point and Delete Point menu items enabled. Selecting the Add Point menu item will locate a new vertex between the vertex that was selected and its neighbor. The new vertex can then be moved to a new location. Selecting the Delete Point menu item will delete the vertex that the cursor was positioned on.

The properties of the polyline can be changed when the toolbar is visible by positioning the cursor on a line or vertex and either double-clicking the left mouse button or clicking the right mouse button and selecting Properties from the pop-up menu. The Polyline Characteristics dialog shown in Figure 15-12 will appear, allowing the line type and color can be changed. If the Smooth check box is selected then the polyline will be displayed with cubic Bezier curves using the endpoints and control points specified by the vertices. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points; the ending point of the previous curve is used as the starting point, the next two points in the sequence are control points, and the third is the ending point. Note that if the number of points in the polyline is not evenly divisible by three then one or two points will not be represented in the Bezier curve. When the Diagram Window toolbar is visible, dotted lines will be drawn between the vertices of the original polyline in order to clearly indicate their positions in case you wish to move them, as shown in Figure 15-13. These dotted lines will not be visible when the toolbar is hidden.

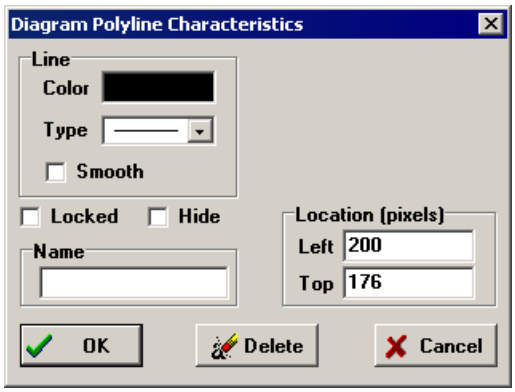


Figure 15-12: Diagram Polyline Characteristics dialog.

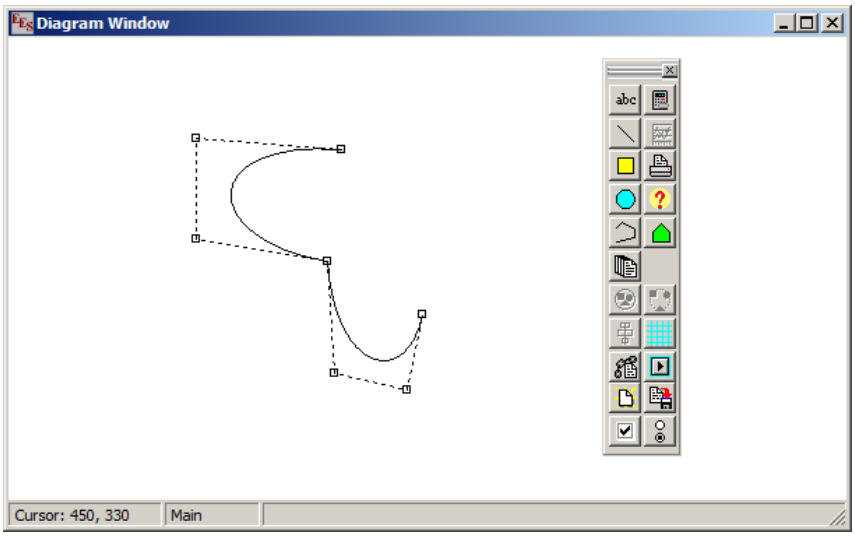


Figure 15-13: Polyline converted into Bezier curve by selecting the smooth option. The dotted lines will not be displayed in application mode (when the tool bar is hidden).

Polygon objects items can be drawn on the Diagram Window by clicking the Add filled polygon button on the Diagram Window toolbar ( ) shown in Figure 15-1. The process is identical to the process for drawing a polyline, except that the drawing process is concluded by clicking on the first point, which is marked with a circle. Alternatively, double-clicking the left mouse button or pressing the Enter key will cause the cursor to move from the last entered point to the first point in order to form an enclosed area and conclude the drawing process. The enclosed area will, by default, be shaded in lime color.

The polygon object can be moved to another location by pressing and holding the left mouse button while the cursor is anywhere within the polygon and then moving the mouse. The positions of any of the vertices of the polygon can be moved by pressing and holding the left mouse button while the cursor is positioned on a vertex. This action will cause the cursor to change from a pointer to left-right-up-down arrows. Additional vertices can be added and existing vertices can be deleted using the Add point and Delete point menu items in the pop-up menu that appears when you click the right mouse button on a vertex.

The characteristics of the polygon can be changed by double-clicking the left mouse button anywhere within the enclosed area or by clicking the right mouse button and selecting the Properties menu item from the pop-up menu. Either action will bring up the Polygon Characteristics dialog shown in Figure 15-14.

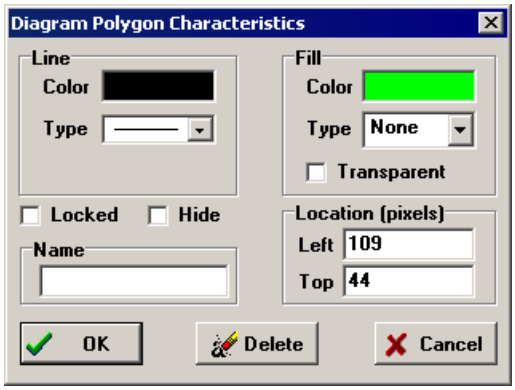


Figure 15-14: Diagram Polygon characteristics dialog.

The line color, line type, fill color, and fill type can all be specified. These variables can be controlled or animated with EES variables if a name is provided for the polygon, as described in Chapter 16. If the Transparent checkbox is selected then the graphic object will be display in a way that allows objects underneath it to be seen.

Accessing the Diagram Window Palette

The Diagram Window Palette provides access to graphic objects that can be placed on the Diagram (or Child Diagram) Window. Many useful graphic objects are provided with EES. In addition, users can provide their own graphic items.

The graphic files that are included in the Diagram Window Palette are located in the EES Userlib\Palette directory or within a subfolder in this directory. Graphic items can be provided in files with any of the following formats: .bmp, .emf, .jpg, or .wmf. Files can be added by the user in any of these formats to the Userlib\Palette folder or to a sub-folder within the Userlib\Palette folder.

When the Access Palette button on the Diagram Window Toolbar is selected, EES will bring up a dialog window that displays the files in the Userlib\Palette directory, as shown in Figure 15-15.

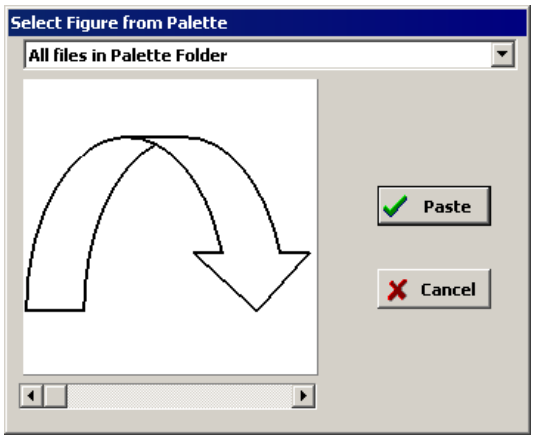


Figure 15-15: Diagram Window Palette dialog.

The drop-down list at the top of the dialog controls which graphic files are displayed. By default, all files in the Userlib\Palette folder and all subfolders within this folder will be displayed. Clicking on the drop-down list will display the items in the associated subfolders in the Userlib\Palette folder. If a subfolder is selected, then only the files in the selected subfolder will be displayed.

The contents of the graphic files in the selected directory are shown, one at a time, in the white rectangle. Click the scroll bar to display other graphic files in the selected folder. Note that the graphic is scaled to fit in the rectangle. The actual size may be larger or smaller than the displayed size.

Click the Paste button to Paste the graphic item into the Diagram Window. Once it is placed on the Diagram Window, the graphic item can be moved to a different location, stretched, or scaled. To move the item to different location, press and hold the left mouse button down on the item while sliding the item to a new location. To change the size of the item, click on one of the eight handles (small boxes) that surround the item when it is selected, as shown in Figure 15-16. If you hold the Shift key down while changing the size of the item, the aspect ratio will be maintained.

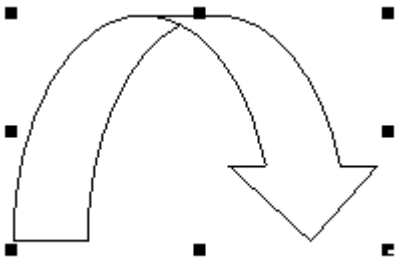


Figure 15-16: Item from the Palette dialog pasted into the Diagram Window.

Right-clicking on the graphic item will display the pop-up menu shown in Figure 15-17.

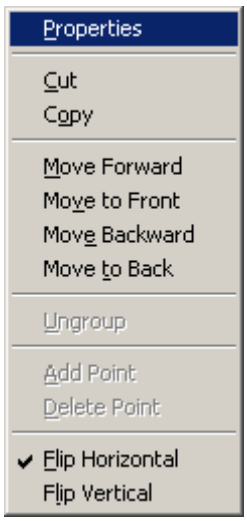


Figure 15-17: Pop-up menu that appears after clicking the right mouse button on a palette graphic item.

Select the Properties menu item to bring up a dialog that allows the attributes of the item to be changed. The dialog allows a name to be assigned for use in animation, as described in Chapter 16. The pop-up menu also provides menu items to flip the selected graphic item horizontally or vertically.

15.2 Placing Text Items in the Diagram Window

It is necessary for the Diagram Window to be in development mode (i.e., the Diagram Window toolbar must be visible) in order to add, delete or modify text items. Four types of text items can be placed on a Diagram Window: plain text, formatted text, EES input variables, and EES output variables.

The Professional version provides enhancements and object-linking ability for formatted text as well as additional capabilities for entering input variables, such as slider controls, check boxes, and radio groups.

Information on how to enter and modify these text items are presented in this section.

Entering Plain Text

Simple text items, such as labels, can be placed in the Diagram Window using the Add text button in the Diagram Window tool bar (abc) shown in Figure 15-1. Clicking this button will bring up the Diagram Text Item dialog, shown in Figure 15-18. The type of text that is entered is specified using the radio buttons in the Type group at the upper left of the dialog. Click the Text button to enter plain text, as shown in Figure 15-18.

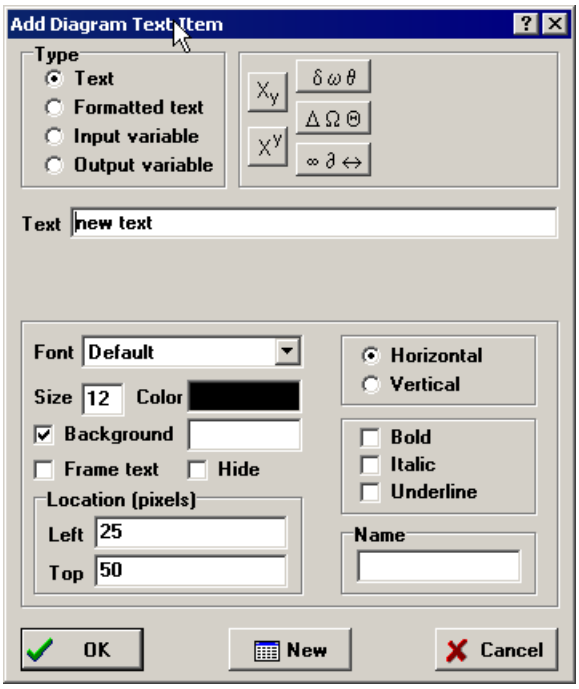
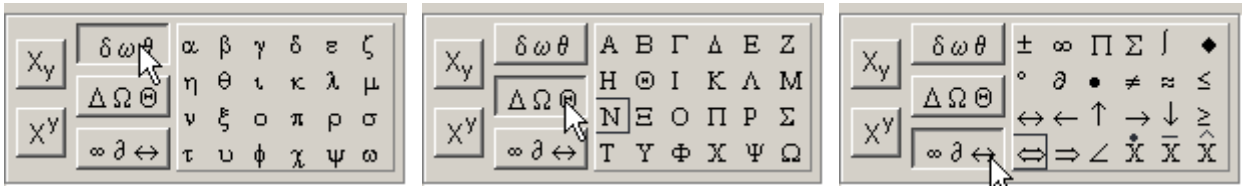


Figure 15-18: Diagram Text Item dialog.

The text is entered in the field to the right of the word Text, below the Type group. The field will display "new text" when a new text item is being created. Enter the text you wish to appear on the Diagram Window. The font, size, style, orientation, and the foreground and background colors can be specified.

In addition, the appearance of the text can be enhanced with the controls in the box at the upper right of the dialog. These controls merit further explanation. Clicking the X_y or X^y button will change the selected text in the text field into a subscript or superscript, respectively. For example, to enter $T_{initial}$, type Tinitial into the text field. Then select initial and click the X_y button. The text field will change to T\dinitial\. The \d instructs EES to display the following text up to the next \ as a subscript. \u in place of \d will result in a superscript. You can enter these codes directly, without using the buttons if you prefer.

Pressing the $\delta\,\omega\,\theta$ button will display a palette of lower case Greek letters, as shown in Figure 15-19(a). Sliding the cursor from the button to the palette while holding the left button down will place the associated letter into the text field at the cursor position. The text display will use the indicator \s to indicate symbol font followed by the character. A \ character terminates the symbol font. The $\Delta\,\Omega\,\Theta$ button provides the same capability for the upper case Greek letters, shown in Figure 15-19(b). The $\infty\,\partial\,\leftrightarrow$ button allows the symbols shown in Figure 15-19(c) to be entered. However, the last three buttons on this palette differ from the others. These buttons first require that a text be selected in the text field. Then, clicking the button will place a dot, bar, or hat symbol above the selected text.



(a) **(b)** **(c)**

Figure 15-19: Palettes for (a) lower case Greek letters, (b) upper case Greek letters, and (c) symbols.

Clicking the OK button will place the text item in the Diagram Window and close the dialog. Clicking the New button will also place the text item in the Diagram Window, but the dialog window will remain open so that additional text items can be entered. The Cancel button will close the Diagram Window without placing the text item.

When the Diagram Window is in development mode, any text item can be moved. Click on the text item to select it. Selected text items will be displayed within a red dotted line rectangle. Press and hold the left mouse button down while the cursor is positioned within the rectangle and slide the mouse to move the text item to the desired location.

The characteristics of existing text items can be viewed or changed by moving the cursor over the text item and double-clicking the left mouse button or clicking the right mouse button and then selecting the Properties menu item from the pop-up menu that will appear. Either action will bring up the Diagram Text Item dialog shown in Figure 15-18 where changes can be made.

Entering Formatted Text

Formatted text is defined here as a text item consisting of one or more lines of text or paragraphs with format attributes such as subscripts, style, font variations, and possibly embedded graphics. There is only one way to enter this information in the Commercial version and that is to compose and copy the text item in a word processor and paste it into the Diagram Window. The text item will then appear exactly as it did when it was copied, but it will be pasted as a graphic object and cannot be edited within EES.

The Professional version of EES provides much more capability for formatted text items. A paragraph can be composed directly in the Add Diagram Text item dialog when the Formatted Text button is selected, as shown in Figure 15-20. Selecting the OK button pastes the text in this dialog into the Diagram Window as an editable rich text item.

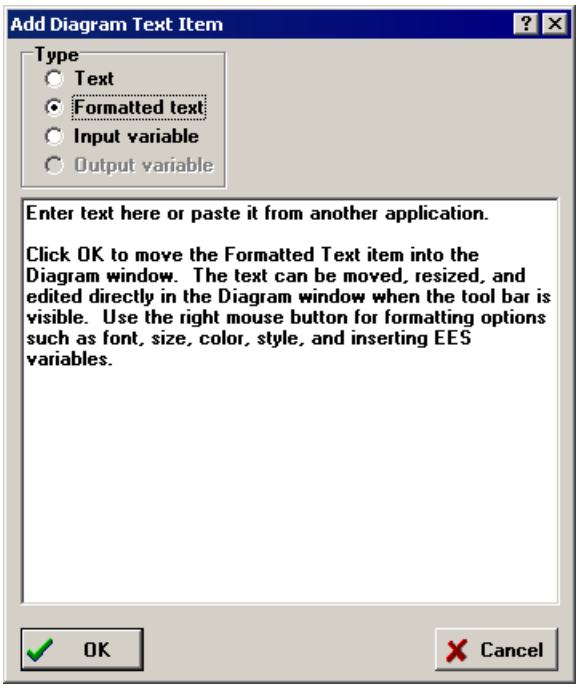


Figure 15-20: Formatted text entry option in the Add Diagram Text Item dialog.

Alternatively, a rich text item that has been composed in another application, such as a word processor, can be pasted directly into the Diagram Window using the Paste command from the Options menu. When EES detects a rich text item on the clipboard, it will display the Select clipboard format dialog shown in Figure 15-21 after the Paste command is issued.

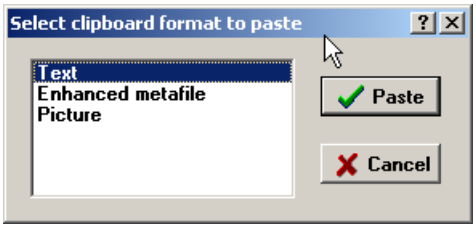


Figure 15-21: Paste options for text in the Professional version.

Selecting the Text option will paste the contents of the clipboard into the Diagram Window in rich text format, allowing it to be edited. Any graphic objects included with the text will be pasted as well. As an example, the thermodynamics problem statement shown in Figure 15-22 was copied from Microsoft Word and pasted into the Diagram Window with the Text option.

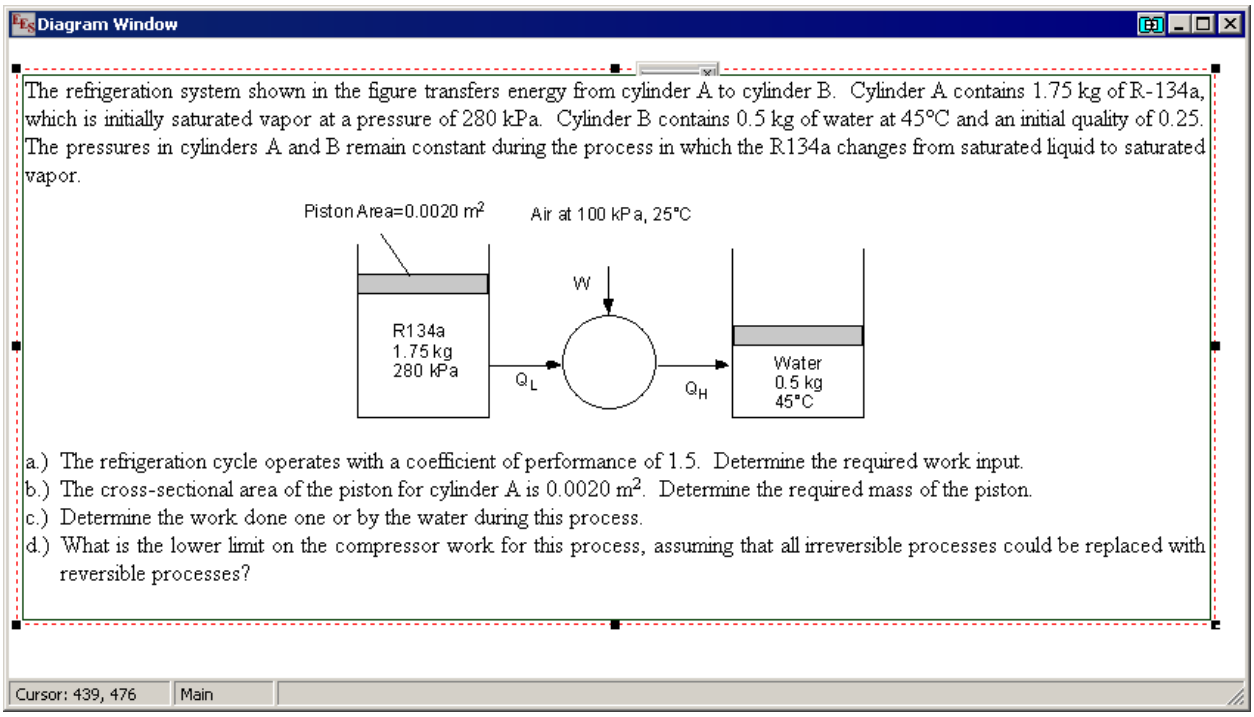


Figure 15-22: Text and graphics copied from a word processor and pasted with the text option.

The text width will initially be set to the width of the Diagram Window. Clicking the left mouse button with the cursor positioned on the text item will select it. A selected text item is displayed with a red dotted line border with 8 handles, as shown in Figure 15-22. The text can be resized by positioning the cursor on any one of the handles and then pressing and holding the left mouse button down while moving the mouse. The text can be moved when it is selected by pressing and holding the left mouse button down while the cursor is within the text rectangle. The text can be edited in the usual manner. Clicking the right mouse button in the text rectangle when the text is selected will access the pop-up menu shown in Figure 15-23. Using this menu, selected text can be cut or copied and the font, style, size and color can be changed.

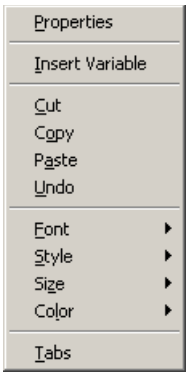


Figure 15-23: Pop-up menu for rich text item in the Diagram Window.

Selecting the Insert Variable menu item will bring up the dialog shown in Figure 15-24 which provides a way to insert an EES variable in a 'live' manner so that the value displayed in the text changes as the value of the variable changes. The dialog provides a list of all current EES

variables in the main section of the Equations Window. The check boxes allow the name, value, and/or units of the selected variable to appear at the cursor location. For example, clicking the OK button when the dialog is set as shown in Figure 15-24 will result in #3COP appearing at the cursor position. When the Diagram Window is viewed in application mode (i.e., with the tool bar hidden) the #3COP will be replaced with COP = 1.5 [-], where 1.5 is the current value of the variable COP. If the value of the variable COP is changed then the updated value will automatically appear in the text item.

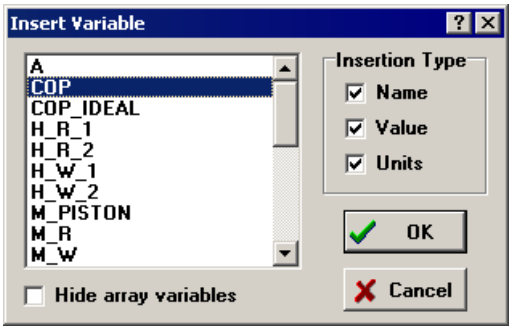


Figure 15-24: Insert Variable dialog.

The text item can also be pasted as an enhanced metafile or picture. Both of these options paste the rich text selection from the clipboard into the Diagram Window as a non-editable graphic object that appears exactly as it did when it was copied.

15.3 Entering Input and Output Variables

Normally, the values of EES variables are entered using equations in the Equations Window. However, the Diagram Window offers an alternative way to enter EES variables and provides the beginning of a graphical user interface to an EES program.

An Example of a Graphical User Interface

The input and output capabilities will be illustrated with a simple thermodynamics problem that calculates the outlet state of a refrigerant throttle given the fluid, throttle area (A), inlet temperature, pressure, and velocity (T_1, P_1, and $\tilde{V}_1$), and the outlet pressure (P_2). Open EES and enter the inputs in the Equations Window.

```
$UnitSystem SI C kPa J Mass
$TabStops 0.2 3.5 in

R$='R134a'                          "refrigerant"
A=0.005 [m^2]                       "cross-sectional area"
T[1]=50 [C]                         "inlet temperature"
P[1]=700 [kPa]                      "inlet pressure"
Vel[1]=15 [m/s]                     "inlet velocity"
P[2]=300 [kPa]                      "outlet pressure"
```

The specific volume and specific enthalpy at the inlet (v_1 and h_1) are fixed by the temperature and pressure. The mass flow at the inlet can be calculated according to:

$$\dot{m}_1 = \frac{\tilde{V}_1 A}{v_1} \tag{15-1}$$

A mass balance on the throttle provides:

$$\dot{m}_2 = \dot{m}_1 \tag{15-2}$$

```
v[1]=volume(R$,T=T[1],P=P[1])          "specific volume at inlet state"
h[1]=enthalpy(R$,T=T[1],P=P[1])        "specific enthalpy at inlet state"
m_dot[1]=A*Vel[1]/v[1]                 "mass flow rate at inlet"
m_dot[2]=m_dot[1]                      "steady-state mass balance"
```

The mass flow rate at the outlet is given by:

$$\dot{m}_2 = \frac{\tilde{V}_2 A}{v_2} \tag{15-3}$$

The specific volume and specific enthalpy at the exit (v_2 and h_2) are fixed by the exit temperature and pressure (T_2 and P_2). An energy balance on the throttle provides:

$$h_1 + \frac{\tilde{V}_1^2}{2} = h_2 + \frac{\tilde{V}_2^2}{2} \tag{15-4}$$

Equations (15-3) and (15-4) together with the property relations for $v_2(T_2, P_2)$ and $h_2(T_2, P_2)$ represent four equations in the four unknowns $\tilde{V}_2$, v_2, h_2, and T_2. They are entered in EES:

```
m_dot[2]=A*Vel[2]/v[2]                 "fixes velocity at outlet state"
v[2]=volume(R$,T=T[2],P=P[2])          "specific volume at outlet state"
h[2]=enthalpy(R$,T=T[2],P=P[2])        "specific enthalpy at outlet state"
h[1]+Vel[1]^2/2=h[2]+Vel[2]^2/2        "steady-state energy balance"
```

and solved, providing T_2 = 42.1°C. We will now enhance this problem by providing it with a graphical user interface that allows the inlet state to be specified in the Diagram Window.

Entering Input Numerical Variables from the Diagram Window

The input and outlet variables for this problem will be placed in the Diagram Window that contains the throttle schematic that was previously inserted, as shown in Figure 15-3. Note that the throttle schematic is also available from the Palette, discussed in Section 15.1. Select the Diagram Window menu item from the Windows menu or enter Ctrl-D to access the Diagram Window. If necessary, put the Diagram Window into development mode by selecting the Show Diagram Tool Bar menu item from the Options menu or by entering Ctrl-D again. The outlet pressure is currently set to 300 kPa with an equation in the Equations Window. We will instead

enter its value directly in the Diagram Window. Click on the Add Text button in the Diagram Window tool bar (in Figure 15-1) in order to access the Add Diagram Text Item dialog shown in Figure 15-25. Click the Input variable radio button and all of the variables in the Main section of the EES program will be displayed in the list on the right. Note that variables in functions, procedures, subprograms and modules are not included in this list and they cannot be entered in the Diagram Window. Select the variable P[2] and enter display specifications as you wish (Figure 15-25). Click the OK button to place the input field for the variable P[2] on the Diagram Window.

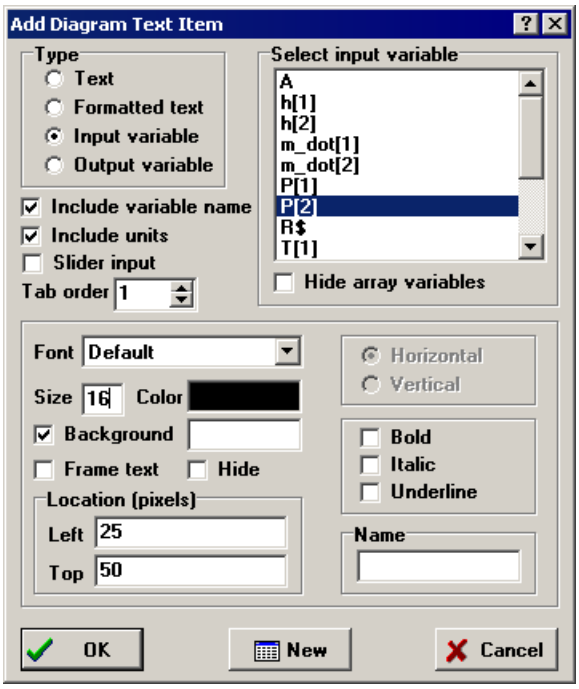


Figure 15-25: Specifying EES variable P[2] as an input variable.

An input variable will be displayed with the value enclosed in a rectangle. This value can be edited and it acts to set the value of the associated variable to the value entered in the box. The input variable box provides the same functionality as entering the equation P[2] = 300 [kPa] directly in the Equations Window.

The input variable is initially placed at a default location. Select the input variable by moving the cursor on it and pressing the left mouse button. A red dotted border will appear around a selected text item. Press and hold the left mouse button down and then slide the input variable to the desired location in the Diagram Window, as shown in Figure 15-27.

Displaying Output Variables on the Diagram Window

Next we will place an output variable on the Diagram Window. Click the Add text tool button from the Diagram Window tool bar ( ) and then select the Output variable radio button. Select the variable T[2] from the list that is populated at the right. Change its color to blue and the font size to 16 pt, as shown in Figure 15-26. Click OK to place the output variable T[2] onto the Diagram Window. Move the T[2] output variable to the desired location. The Diagram Window should now appear as shown in Figure 15-27.

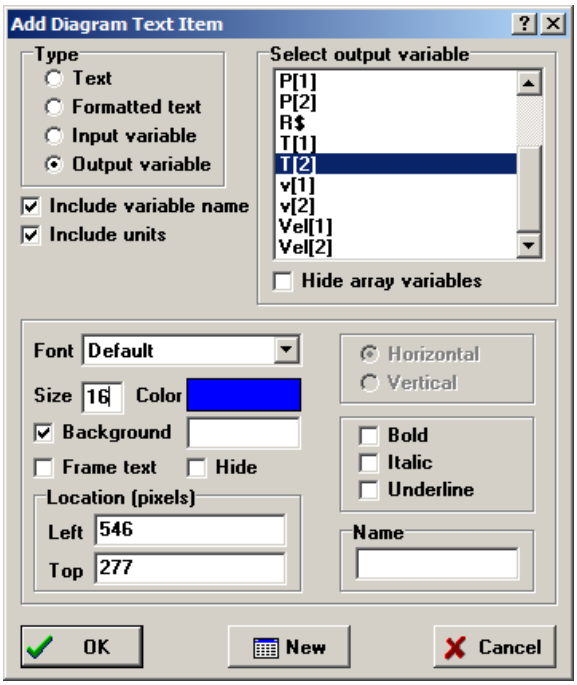


Figure 15-26: Displaying EES variable T[2] as an output variable.

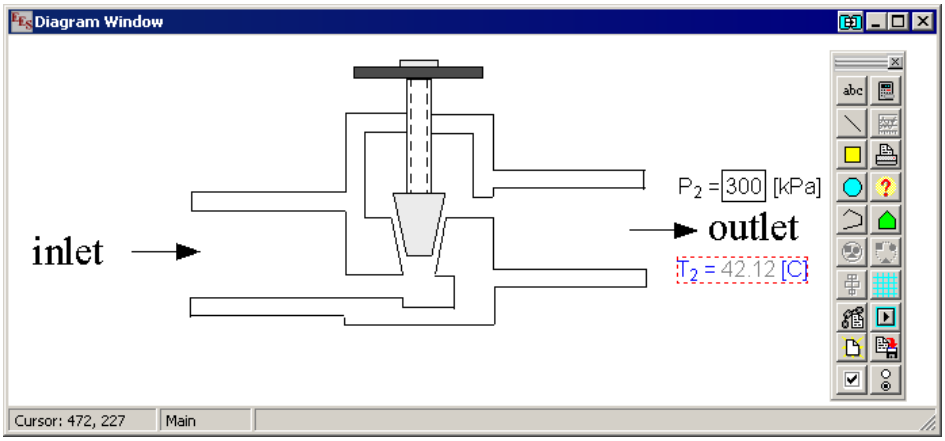


Figure 15-27: Diagram Window with one input and one output variable.

Place the Diagram Window into application mode and try to solve the problem by entering F2. You should see the error message shown in Figure 15-28 indicating that variable P[2] is now defined in both the Diagram Window and the Equations Window.

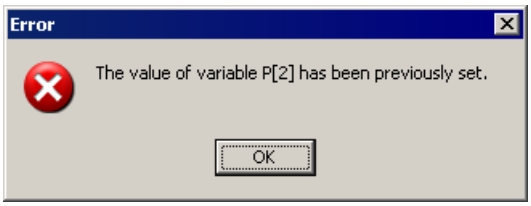


Figure 15-28: Error message indicating the P[2] is defined in both the Diagram and Equation Windows.

Comment out the equation that sets P[2] in the Equations Window or, alternatively, use the $IfNot directive with the conditional DiagramWindow so that the variable P[2] is only set if the calculations are not initiated from the Diagram Window. The $IfNot directive is discussed in Section 14.7.

```
$IfNot DiagramWindow
    P[2]=300 [kPa]                    "outlet pressure"
$EndIf
```

When P[2] is set with the $IfNot directive as shown above, the program will operate correctly whether or not the Diagram Window is used.

Bring the Diagram Window to the front and press F2 to solve the problem. The Diagram Window will be updated with the current value of variable T[2] when the calculations are completed. Change the value of P[2] to 100 kPa using the input variable in the Diagram Window and repeat the calculations. The Diagram Window should now indicate that T[2] is 32.23°C.

Limits on Input Variables

You can change the output format of the variable in the Variable Information dialog should you wish to display fewer significant figures. Also, you can place lower and upper limits on input variable using the lower and upper bounds entered in the Variable Information dialog. For example, perhaps we wish to restrict the lower bound for the downstream pressure that is entered by the user to be 100 kPa. Select the Variable Info from the Options menu and enter a lower bound of 100 for variable P[2], as shown in Figure 15-29.

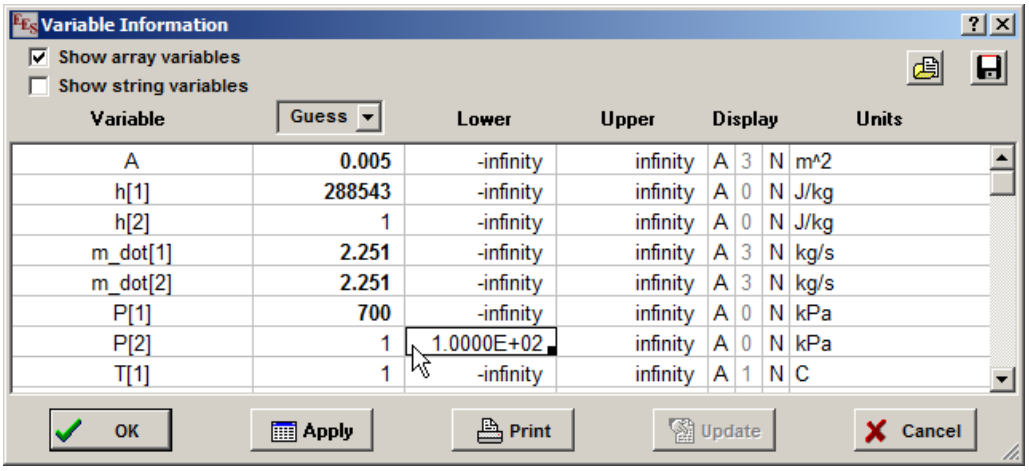


Figure 15-29: Variable Information Window showing the lower bound set for the variable P[2].

If you now attempt to enter a value for P[2] in the Diagram Window that is lower than 100 kPa you will receive the error message shown in Figure 15-30.

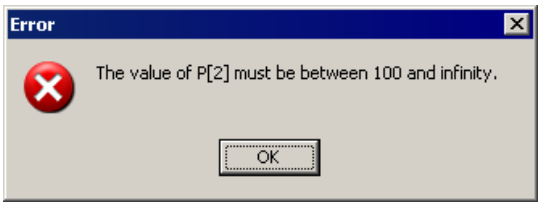


Figure 15-30: Error message that appears when an entered value is not within its lower and upper bounds.

Include the mass flow rate of refrigerant and the outlet velocity as outlet variables. The inlet pressure, inlet temperature, and inlet velocity can be placed on the Diagram Window as input variables. Be sure that all input variables are either commented out in the Equations Window or placed within the $IfNot DiagramWindow and $EndIf directive clause. The Diagram Window now appears as shown in Figure 15-31.

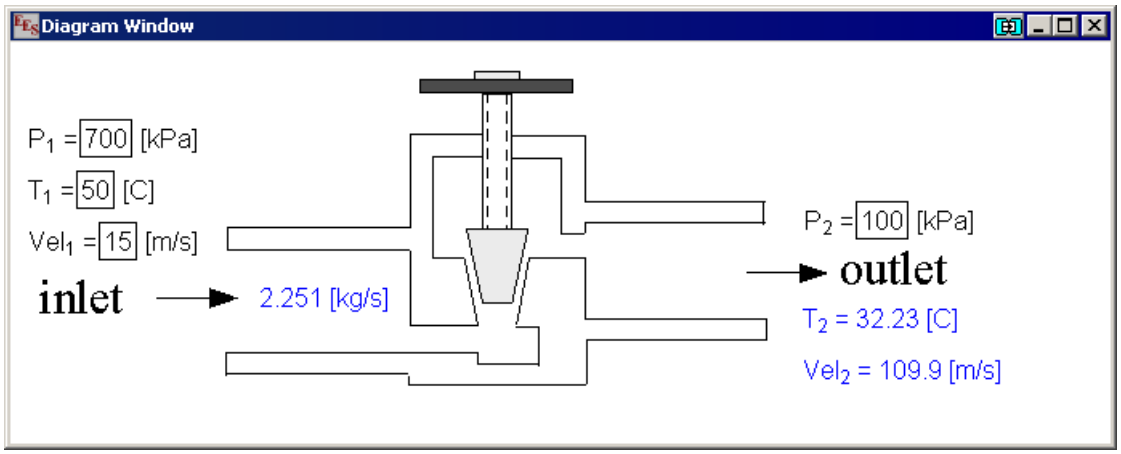


Figure 15-31: Diagram window with all input and output variables shown.

Entering Input String Variables from the Diagram Window

String variables can also be input and output variables. The string variable R$ was used in throttle example to set the name of the fluid flowing through the valve. By making R$ an input variable, we can select the fluid flowing through the throttle from the Diagram Window. Set the Diagram Window to Development mode and click the Add Text Button on the Diagram Window tool bar. Click the Input variable radio button and then select variable R$ from the list, as shown in Figure 15-32. We do not need to see the variable name on the Diagram Window; therefore, uncheck the Include variable name check box.

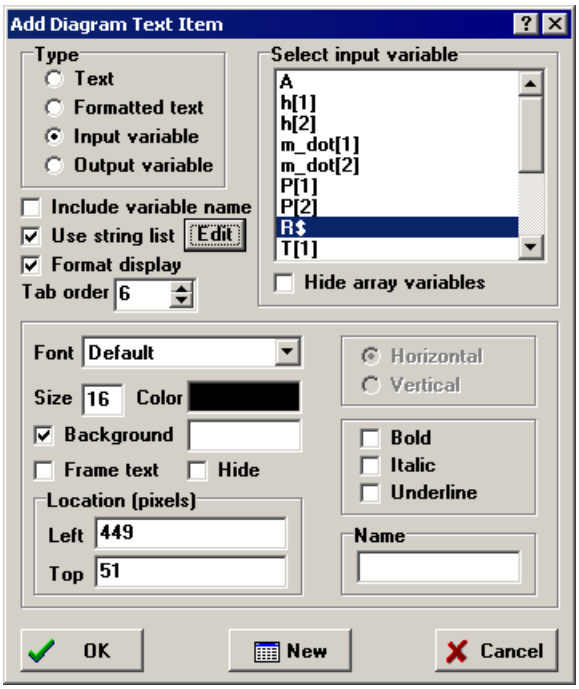


Figure 15-32: Add Diagram Text Item showing options for string variables.

The remaining two check boxes in Figure 15-32 require additional explanation. If you uncheck the Use string list check box and press OK, an input field for variable R$ will be placed on the Diagram Window. However, the user will need to type in the name of the fluid that is to be used, which does not provide any control over the name that is entered and therefore is not particularly convenient for the user. A better choice is to provide a list of allowable fluid names. This option is accomplished by checking the Use string list check box and then selecting the Edit button. A list box will appear, in which you can enter allowable values for the string variable R$, as shown in Figure 15-33.

The Professional version allows the list of strings to be supplied from a text file that is specified in the Read strings from file input box.

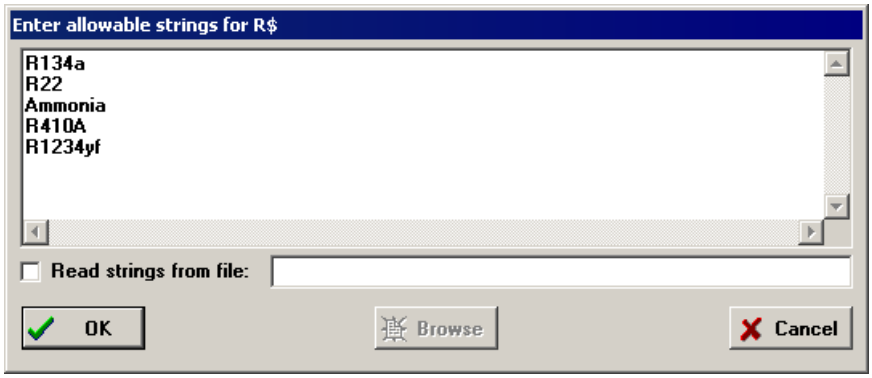


Figure 15-33: List of allows inputs for string variable R$.

Click the OK button to place the input list for variable R$ onto the Diagram Window. Move the input list to the desired location. Hide the tool bar to place the Diagram Window in application

mode. Comment out the equation that sets the string variable R$ in the Equations Window or move it into the $IfNot DiagramWindow / $EndIf clause. The Diagram Window will now provide a drop-down list of choices for the fluid, as shown in Figure 15-34. Note that when any input variable is changed, the color of all of the output variables is changed to gray in order to indicate that they are no longer correct, as shown in Figure 15-34.

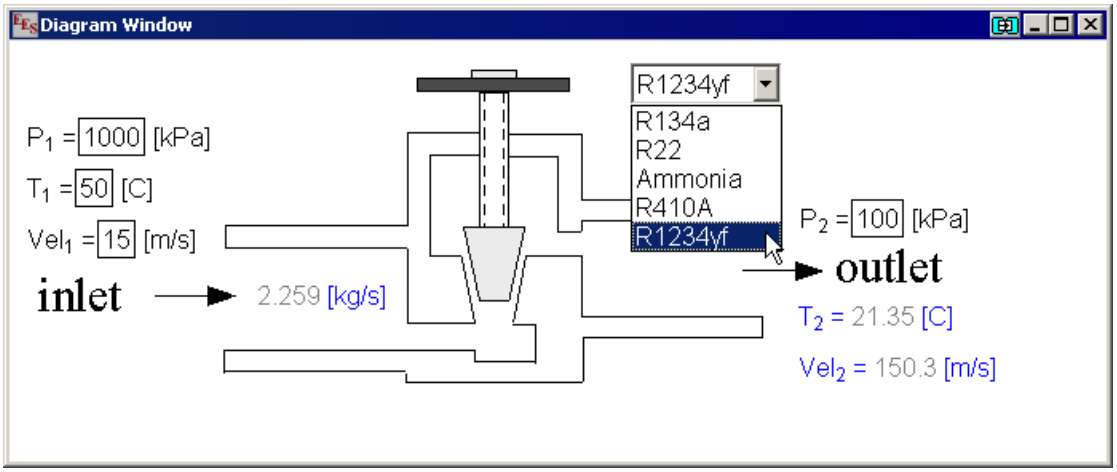


Figure 15-34: Diagram window showing drop-down list of choices for variable R$.

Mapping String Variables in a Drop-Down List to Equations

It is possible to map the strings that are listed in the drop-down list to additional equations. This is accomplished by appending to the string that is to be displayed the characters // and then providing one or more EES equations all on one line, separated by a semicolon (US) or a colon (European). The equation(s) that are provided can also be a $Include directive in order to specify a file that adds additional lines to the Equations Window from a text file.

The capability of mapping string variables can be used in the throttle example to allow the user to select from a drop-down list that is populated by industry-standard names for the refrigerants, rather than the corresponding EES fluid names. To proceed, enter Ctrl-D to place the Diagram Window in development mode with the tool bar visible. Select the refrigerant input list by clicking on it and delete it by pressing the Delete key. Click the Add text button on the tool bar and select Input variable. However, this time, scroll to the bottom of the list and click on *** New String Variable ***. Enter a variable name for the string variable, e.g., RInput$, as shown in Figure 15-35. Note that the variable name must end with $ to indicate that it is a string variable. EES will append $ if it is not provided.

Next, select the new variable, RInput$, from the list. Uncheck the Include variable name check box and check the Use String list and Format display check boxes, if they are not already checked. Click the Edit button and enter the information shown in Figure 15-36.

Click OK to close the list of strings and OK in the Add Diagram text dialog to place the string input list. Hide the Diagram Window toolbar to go into application mode. The list of industry-standard refrigerant names will now appear in the Diagram Window, as shown in Figure 15-37.

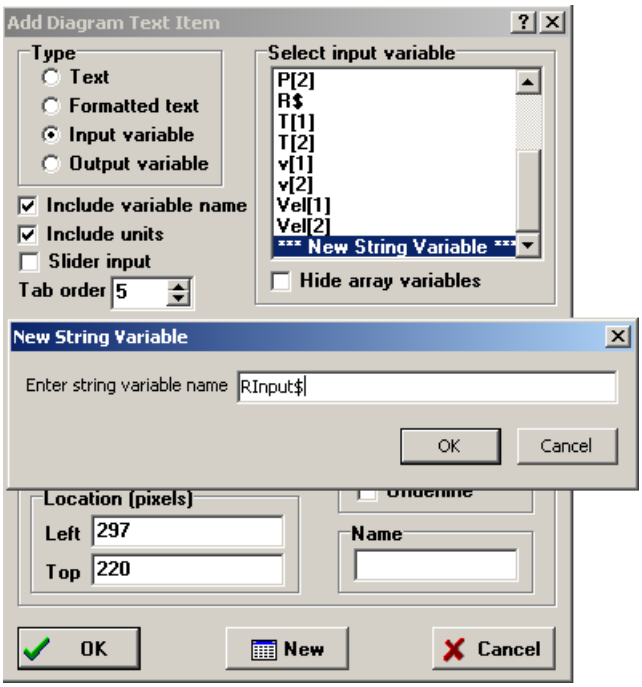


Figure 15-35: Specifying a new string variable.

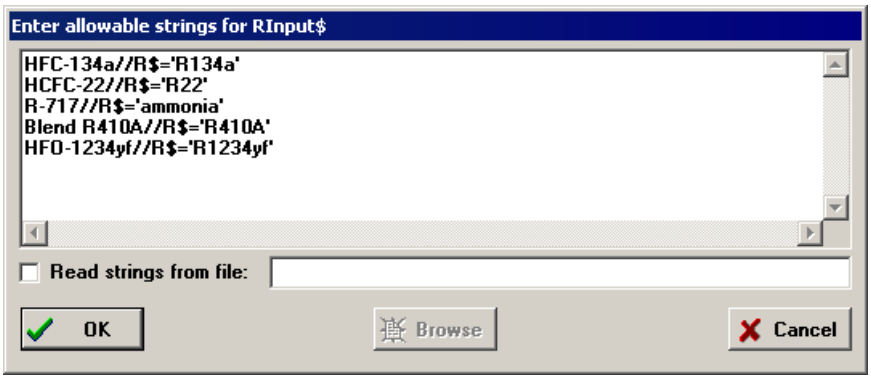


Figure 15-36: List of allowable strings for variable RInput$ with appended specification for variable R$.

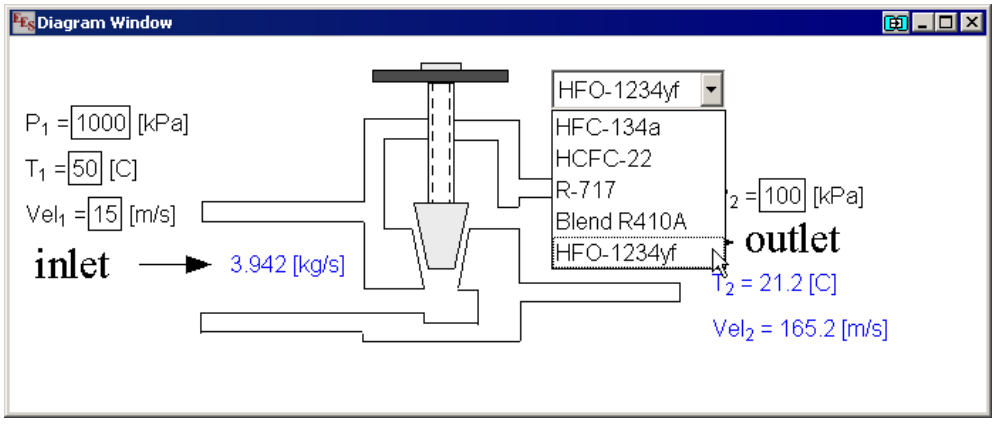


Figure 15-37: Diagram Window showing drop-down list for string variable RInput$.

Because the Format Display check box was checked, the text beginning with // for each entry will not be displayed. In addition, when a selection is made for variable RInput$ from the list, the information to the right of the // will be executed as EES equations. Note the multiple equations can be placed on one line provided a line separator, semi-colon (US) or colon (European), is provided between the equations.

The Professional version provides additional input methods as described in Section 15.6.

The Find Command

In very large Diagram Windows that include several Child Diagram Windows it may become difficult to locate the handle for a specific variable. In this situation, it is useful to utilize the Find command. For example, put the Diagram Window shown in Figure 15-37 into development mode and select Find from the Search menu. The Find Variables dialog shown in Figure 15-38 will be shown. Enter the variable of interest, e.g. P[1], and select OK. EES will open the Diagram or Child Diagram Window containing the input or output terminal corresponding to the variable. The terminal will be highlighted with a red dashed line, as shown in Figure 15-39.

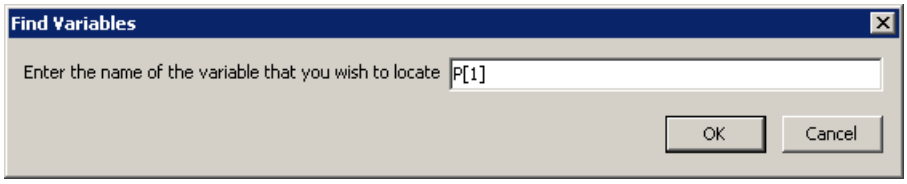


Figure 15-38: Find Variables dialog.

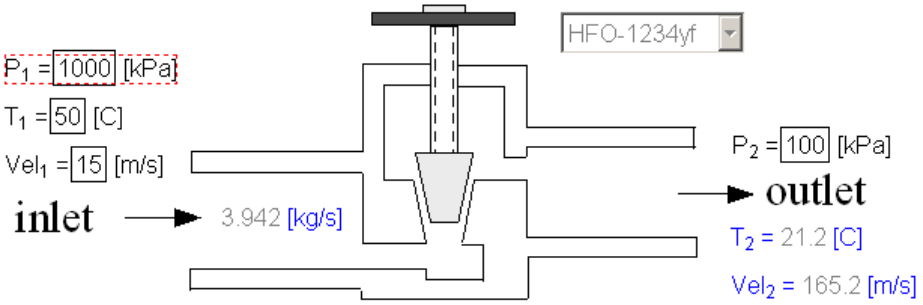


Figure 15-39: Diagram Window with the input variable P[1] highlighted by the Find command.

15.4 Copying and Resizing the Diagram Window

The Copy command from the Edit menu can be used to copy either the entire contents of the Diagram Window or selected items to the Clipboard.

The Copy Command in Application Mode

If the Diagram Window is in application mode, the Copy command will be displayed in the Edit menu as Copy Diagram. Selecting this menu command will create a picture of the Diagram Window as an enhanced metafile and place it on the Clipboard. This picture can then be pasted into a word processor or drawing application.

The Copy Command in Development Mode

The Copy command acts differently when the Diagram Window is in development mode. In this case, one or more selected graphic and text objects are copied to the clipboard. An object in the Diagram Window can be selected by clicking the left mouse button when the cursor is positioned on it. A selected object is displayed with its handles showing or with a red dotted border. If no keys are pressed, selecting an object unselects all other objects. However, multiple graphic or text objects can be selected by pressing and holding the Ctrl key while clicking on the objects. Clicking a selected object causes it to become unselected. Multiple objects can also be selected by dragging a selection rectangle around the objects, as demonstrated in Figure 15-40. The Select All command in the Edit menu will select all of the objects on the Diagram Window.

When one or more objects are selected, the Copy menu command will appear as Copy Items. Selecting this command will copy the objects in an internal form that maintains the information needed for their display in the Diagram Window. The objects can then be pasted into the Diagram Window to create duplicate instances or into a Child Diagram Window (see Section 15.6). The objects can also be pasted into a Diagram or Child Diagram Window in another instance of EES. However, in general Input/Output text items should not be copied into another instance of EES since they contain specific pointers to EES variables that may not be appropriate in a different EES program.

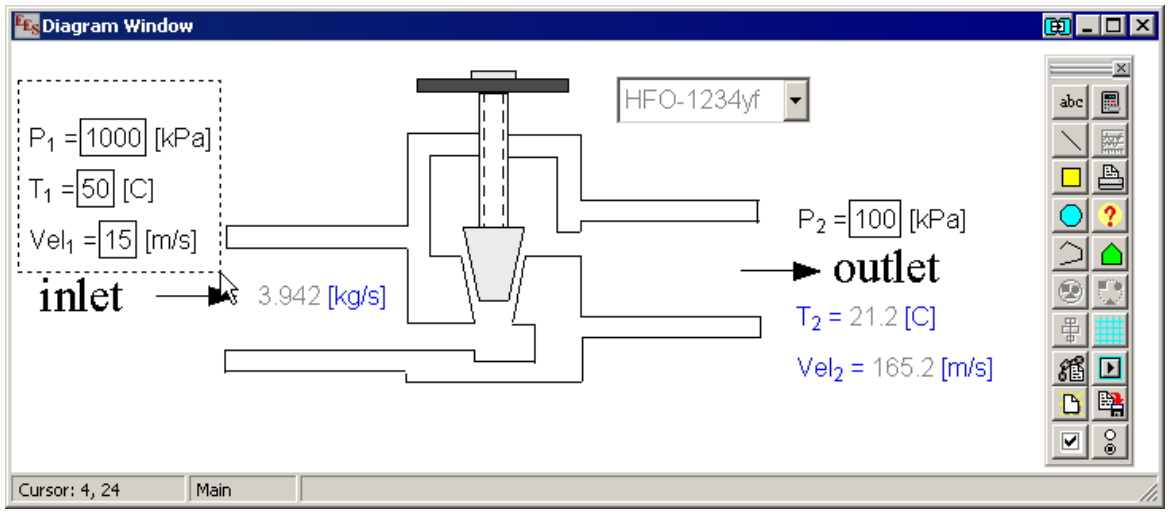


Figure 15-40. Selecting multiple objects in the Diagram Window with a selection rectangle.

Change Window Size Dialog

Clicking the right mouse button on a location in the Diagram Window that is not on an object will cause the Change Window Size dialog shown in Figure 15-41 to appear. This command will scale every object on the Diagram Window by the specified amount. A convenient way to use this command is to select the desired window size and let the command scale the objects to fit in the window.

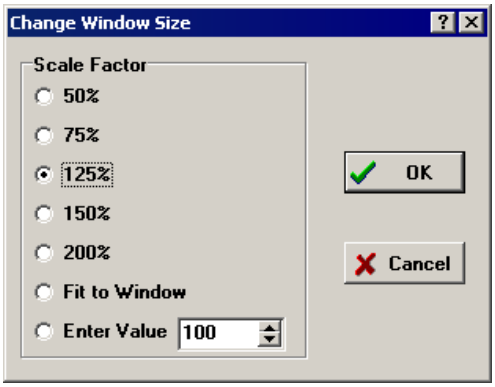


Figure 15-41: Change Window Size dialog.

Clearing the Diagram Window

To clear the Diagram Window, use the Select All command from the Edit menu followed by the Delete command from the Edit menu while the Diagram Window is in development mode.

15.5 Adding Calculate, Print, Plot Access, and Help buttons

The combination of being able to place graphic object and text on the Diagram Window provides the beginning of an effective graphic user interface within EES. This section describes some additional capabilities that are available in all versions of EES in order to further enhance the use of the Diagram Window for this purpose.

Adding a Calculate Button

EES provides keyboard shortcuts to initiate calculations. These keystrokes can be entered to start calculations while the Diagram Window is in the forefront. For example, pressing F2 is equivalent to selecting the Solve command from the Calculate menu. Pressing F3 starts a Solve Table calculation. Other shortcuts are available. However, a user who is viewing the Diagram Window may not know which solve command is needed to initiate the calculations. An elegant way to provide the needed information is to place a Calculate button onto the Diagram window. This can be done by clicking the Add calculation button ( ) on the Diagram Window tool bar shown in Figure 15-1. This action will cause a Calculate button to immediately appear on the diagram window in the Commercial version.

In the Professional version, the dialog window shown in Figure 15-42 will appear, offering a choice between a Calculate button and an Animation Control bar. The animation capabilities provided in the Professional version are described in Chapter 16.

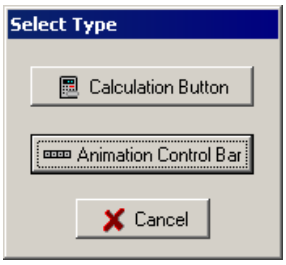


Figure 15-42: Select Type dialog provides a selection of calculate buttons in the Professional version.

Moving and Resizing the Calculate Button

The Calculate button will appear at a default location. The button can be moved to a desired location while the Diagram Window is in development mode by moving the cursor over the button and then pressing and holding the left mouse button down while moving the cursor to drag the Calculate button to a new location. The Calculate button can also be precisely moved by pressing the arrow keys. Each key press moves the button one pixel in the direction of the arrow. Hold the arrow key down to speed the moving process. If the Ctrl key is held down while the arrow keys are pressed, the width or height of the button is changed, rather than its position.

Calculate Button Characteristics Dialog

By default, the Calculate button will have a width of 90 pts and a height of 25 pts with a caption that reads “Calculate”. Also, when this button is clicked, it will initiate calculations just as if the F2 button were pressed. However, all of these defaults can be changed by double-clicking the left mouse button or clicking the right mouse button on the Calculate button while the Diagram Window is in development mode. Either action will bring up the Calculate Button Characteristics dialog shown in Figure 15-43.

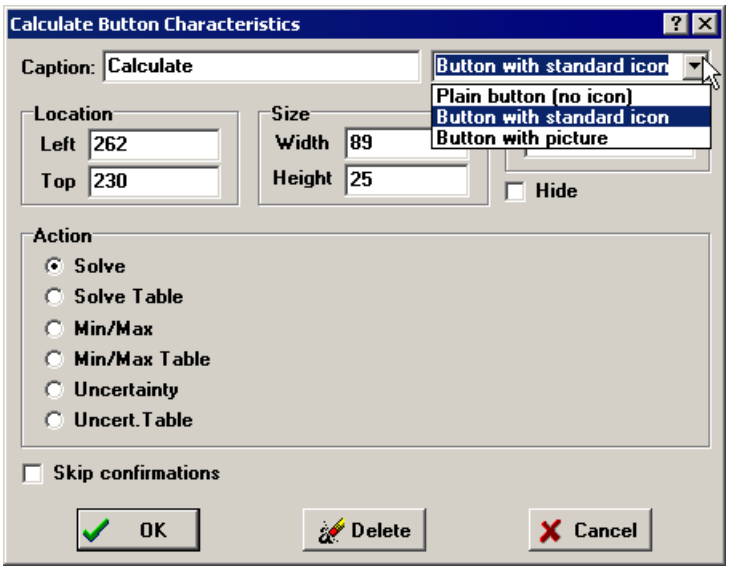


Figure 15-43: Calculate Button Characteristics dialog

The button caption and button type can be changed with the controls at the top of the dialog. The Commercial version provides the options of a plain button or a button with a standard icon.

The Professional version offers the option of having a user-provided picture displayed on the button. The button location and size can be specified. The option for providing a name for the button is available in the Professional version. Naming the button is used for animation, as described in Chapter 16.

The action that the Calculate button initiates when it is pressed in application mode is specified by selecting one of the radio button options. Normally, EES displays a dialog before it starts or after it ends any type of calculations, such as solving a Parametric table or doing an optimization. Checking the Skip confirmation button will suppress the appearance of the dialog.

The graphic user interface developed for the throttle in Section 15.2 will benefit from a Calculate button, as shown in Figure 15-44. Use the default options shown in Figure 15-43, but select the Skip confirmation check box. When the Calculate button is clicked while the Diagram window is in application mode, the calculations will be initiated and the Diagram Window will be updated without displaying the confirmation dialog after the calculations are completed.

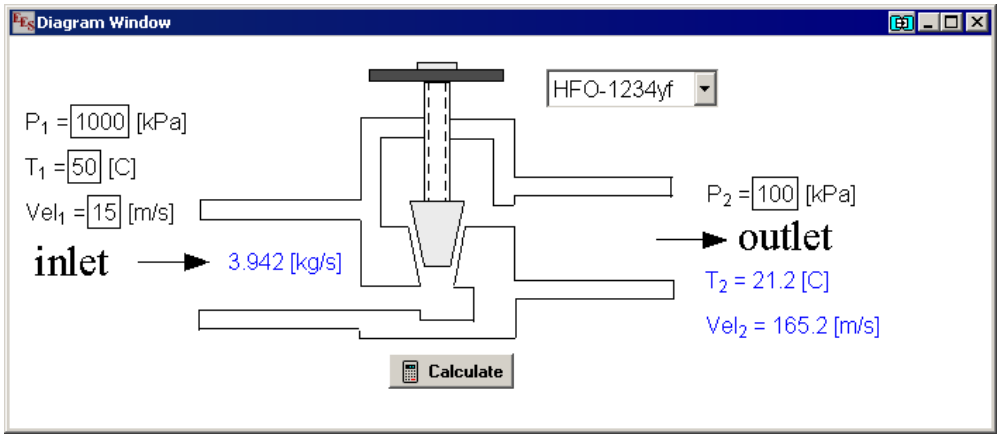


Figure 15-44: Diagram window with a Calculate button

Adding a Print Button

The Diagram Window can be printed by selecting the Print command from the File menu. However, it is more convenient to provide a Print button directly on the Diagram Window. The Add Print button ( ) on the tool bar shown in Figure 15-1 will place a Print button in a default location. The button can be moved to the desired location or resized in the same manner as described for the Calculate button. Double-clicking the left mouse button or clicking the right mouse button when the cursor is positioned on the Print button will bring up the Print Button Characteristics dialog shown in Figure 15-45.

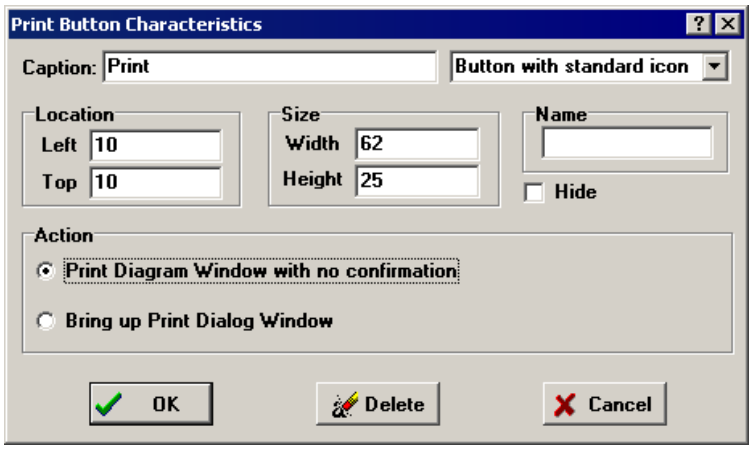


Figure 15-45: Print Button Characteristics dialog.

Adding a Plot Access Button

The best way to display calculated results is often with a plot. A plot access button can be placed on the Diagram Window in order to allow easy access to a particular plot window. As an example, we will create a plot of the inlet and outlet states of the refrigerant for the throttle. Click the Calculate button for the inputs shown in Figure 15-44. Next, select Property Plot from the Plots menu and create a property plot for refrigerant R1234yf. It is not possible to automatically change the property plot for different refrigerants with the Commercial version, although this capability is available through the use of a macro or using animation capabilities that are provided in the Professional version. Select R1234yf from the list of fluids and click the P-h radio button. The Property Plot Information dialog should appear as seen in Figure 15-46. Click the OK button to create the plot.

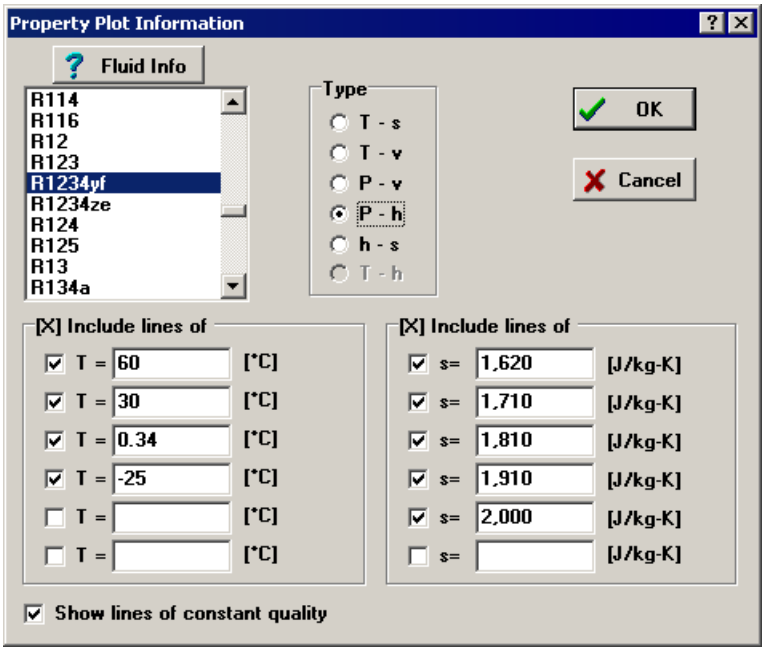


Figure 15-46: Property Plot Information dialog for a P-h plot for R1234yf.

Next, select the Overlay Plot menu command from the Plots menu. Select h[i] for the X-Axis variable and P[i] for the Y-Axis variable, as shown in Figure 15-47. Select the Automatic update and Show array indices check boxes so that the plot will identify states 1 and 2 and automatically change when the values of pressure or specific enthalpy are changed. Click the OK button.

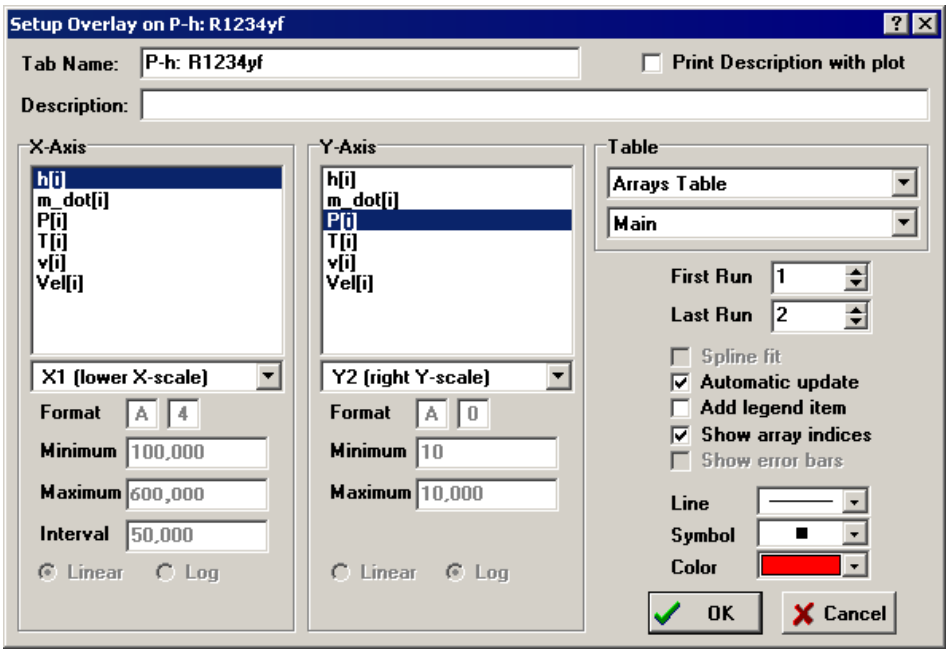


Figure 15-47: Overlay Plot dialog to overlay the pressure and enthalpy on the property plot.

Now, return the Diagram Window and press Ctrl-D to enter development mode. Select the Add plot access button ( ) to access the Show Plot Window Button dialog shown in Figure 15-48(a). The plot that will be displayed when the button is clicked is selected from the drop-down list at the top of the dialog. The Add button to link to Plot window check box is selected by default. (The option to Show plot in Diagram window is only available in the Professional version.) Click the OK button and a Plot Window access button will appear on the Diagram window. The button can be moved or resized in the same manner as described for the Calculate button. Double-click the left mouse button or click the right mouse button when the cursor is on the button to bring up the dialog in Figure 15-48(a) where characteristics such as the button caption and size can be changed.

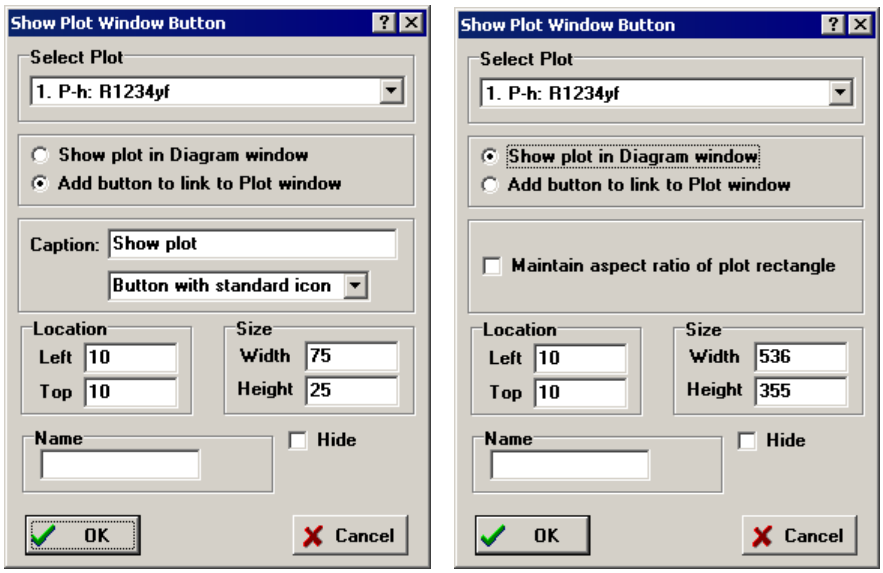


Figure 15-48: Plot window access dialog with (a) plot access via a button and (b) plot shown on the Diagram Window.

Hide the Diagram tool bar (or press Ctrl-D) to enter application mode. Clicking the Show Plot button will bring the selected plot window to the front of all other windows and update its contents. A small arrow button ( ) will be visible at the lower right corner of the plot window, as shown in Figure 15-49. Clicking this button will hide the plot window and bring the Diagram Window back to the front.

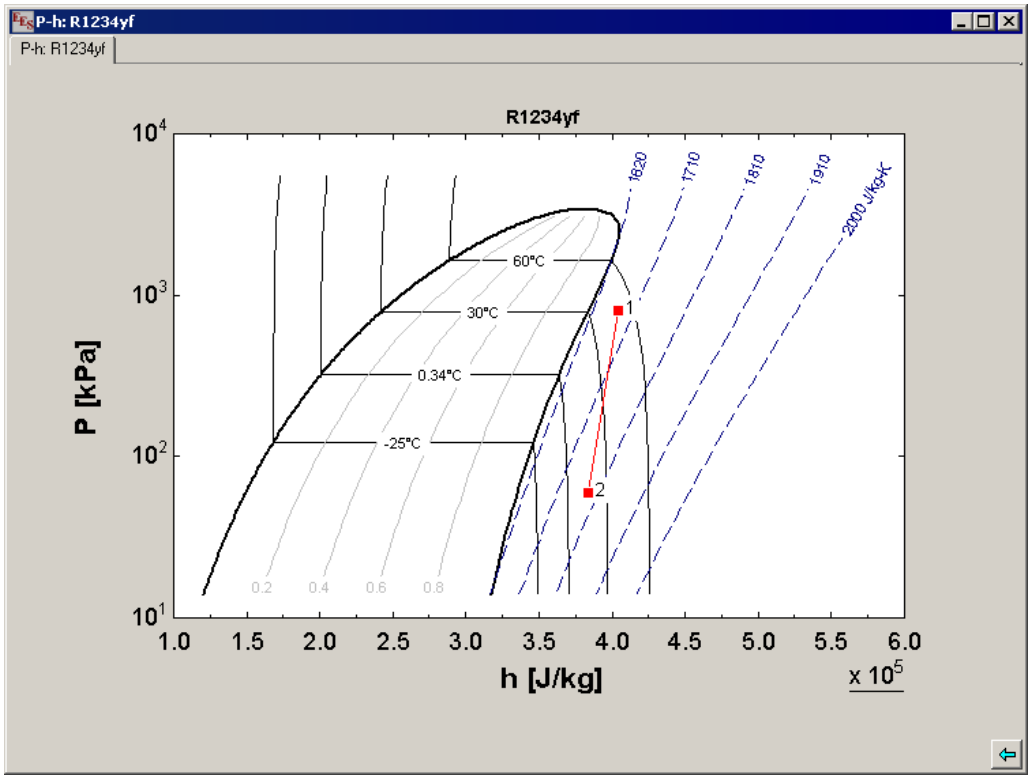


Figure 15-49: Property plot accessed from a button on the Diagram Window.

Showing a Plot on the Diagram Window

The option to show an updated plot in the Diagram Window is provided in the Professional version. Clicking the Show Plot in Diagram window radio button will change the dialog to appear as shown in Figure 15-48(b). Click the check box to maintain the aspect ratio of the plot and then click the OK button. The selected plot will appear in the Diagram Window, drawn with 8 handles. The plot can be moved to the desired location. It can be resized by moving the cursor to one of the handles and then pressing and holding the left mouse button down while moving the mouse. Alternatively, double-clicking the left mouse button or clicking the right mouse button when the cursor is in the plot will bring up the dialog window shown in Figure 15-48(b), where the width and/or height can be entered. Hide the tool bar and change any of the input variables. Click the Calculate button and the updated plot will be directly redrawn in the Diagram Window, as shown in Figure 15-50. Note that the throttling process results in a slight reduction in the specific enthalpy of the refrigerant because of the increase in kinetic energy. The kinetic energy effects can be excluded from the calculations with a check box control, as discussed in Section 15.6.

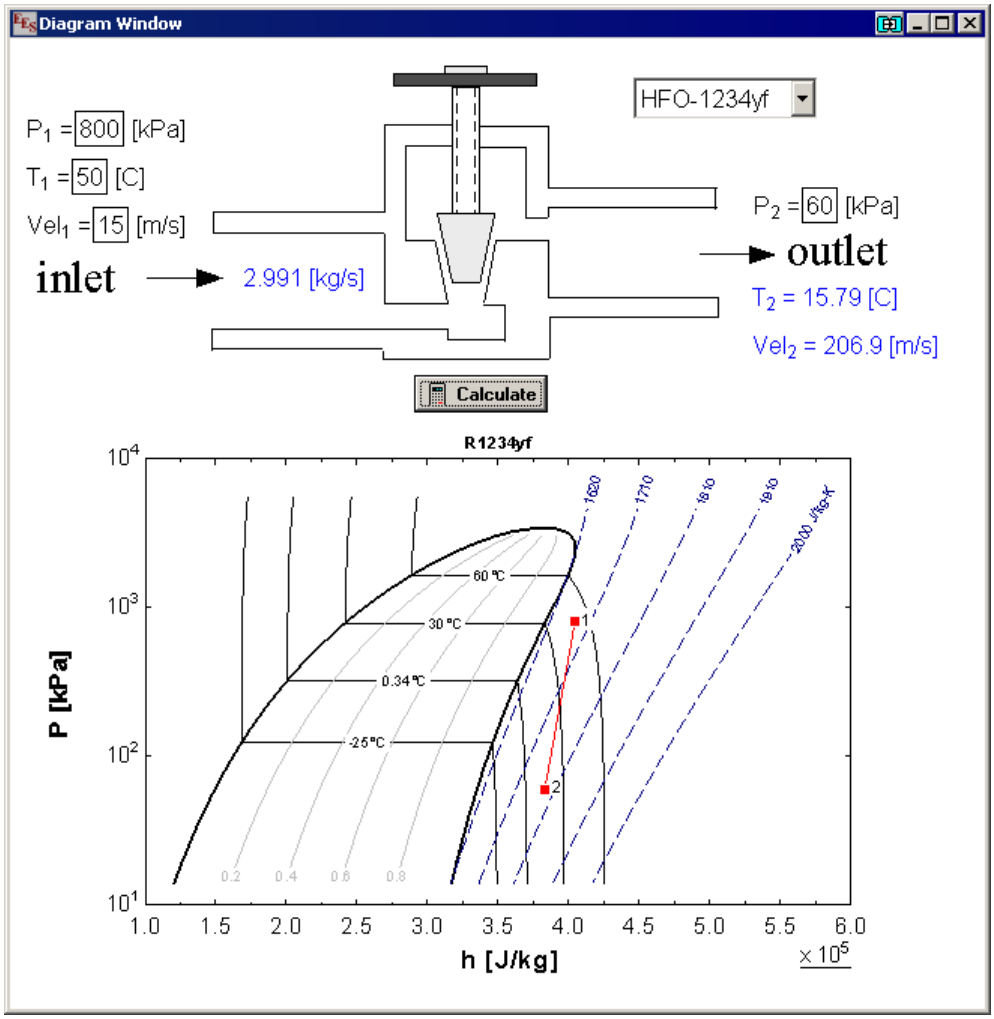


Figure 15-50: Diagram window with the plot window displayed.

Adding a Help Button

Easy access to on-line help or documentation is important in any application. The Add Help button ( ) on the Diagram Window tool bar provides this capability. Clicking this button will bring up the Help Button Information dialog shown in Figure 15-51.

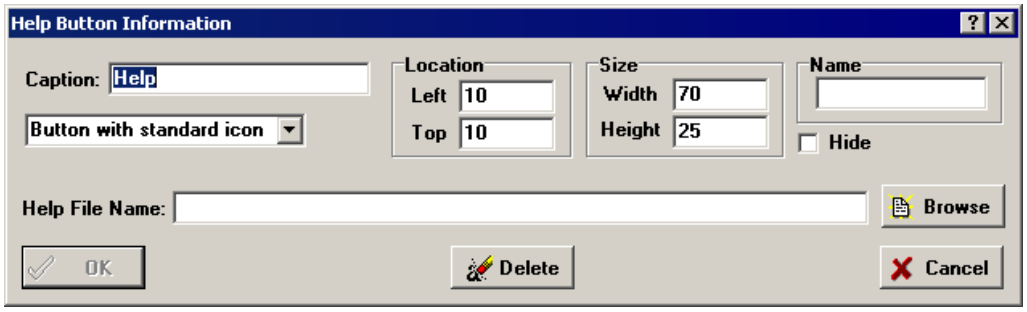


Figure 15-51: Help Button Information dialog.

The button caption, type, location, and size can be specified, in the same way as for the other buttons described in this section. The Help button will display a file when it is clicked in application mode. The file that it will display must be entered in the Help File Name field. Click the Browse button to bring up a standard file open dialog that will assist you in finding the proper file. EES expects the help file to have a file name extension of .chm, .pdf, .htm, or .txt. Click OK to place the button on the Diagram Window. It can be moved in the usual manner. Double-clicking the left button or clicking the right button when the cursor is positioned on the button in development mode will bring up the dialog window shown in Figure 15-51, where additional changes can be made.

15.6 Professional Version Enhancements

The Diagram Window is of major importance in the Professional version because it provides the capability to provide animation (Chapter 16) and stand-alone (distributable) programs (Chapter 17). For these reasons, many additional capabilities for the Diagram Window have been developed for the Professional version. These additional capabilities are summarized in this section.

Child Diagram Windows

The Commercial version of EES supports only a single Diagram Window. The Professional version has no limit on the number of Diagram Windows that you can have. A "hot" area can be created on the main Diagram Window that, when clicked, will bring up a Child Diagram Window. Child Diagram Windows provide all of the capabilities of the main Diagram Window, including the ability to have their own Child Diagram Windows. In addition to clicking on the "hot" area, Child Diagram Windows can also be accessed with Link buttons or by selecting the window from the Diagram Window menu command from the Windows menu.

Child Diagram Windows are very useful for systems that consist of multiple components. The Diagram Window can be designed such that a click on any component within a schematic brings up a Child Diagram Window that contains information (either inputs or outputs) that are related to that component. A simple example of this capability will be demonstrated using the Diagram Window developed for the throttle in the previous sections of this chapter.

A Child Diagram Window is created by pressing and holding the Shift and Ctrl keys down simultaneously while using the mouse with the left button depressed to outline a rectangular area, called a "hot area". We will create a hot area around the handle of the valve, as shown in Figure 15-52. Note that the Diagram window must be in development mode in order to create the "hot area".

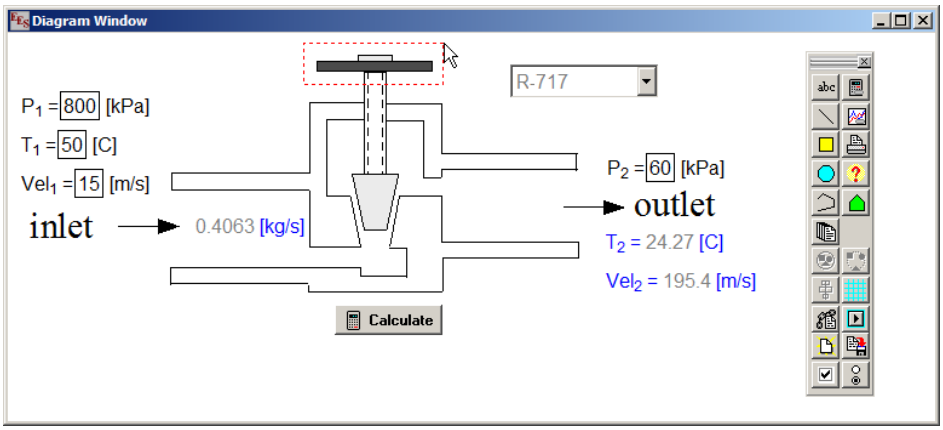


Figure 15-52: Creating a "hot area" by pressing the Shift and Ctrl keys while dragging a rectangular area.

When you release the mouse button, a small dialog window will appear as shown in Figure 15-53 where you must enter a name for the hot area. Click the OK button and a Child Diagram Window having the name that you entered in the dialog will appear, as shown in Figure 15-54. Note that the Child Diagram Window has its own tool bar and all of the capabilities that the main Diagram Window has. You can enter any information or objects you wish in this window, including graphics, text, and EES variable inputs and outputs. A small "go home" navigation button ( ) is initially placed in the lower right corner of the Child Diagram window. Clicking this button will close the Child Diagram Window and bring up the main Diagram Window. If the parent of the Diagram Window is not the main Diagram window but rather another Child Diagram Window then a second button with a blue left arrow will also be displayed in the lower right corner of the window. Clicking this button will close the Child Diagram Window and open its parent Diagram Window. The location and size of these navigation buttons can be changed using the arrow and Ctrl-arrow keys, respectively. A caption can be added to either button by right-clicking on the button while it is in development mode.

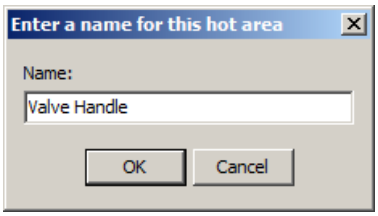


Figure 15-53: The name of the Child Diagram Window is entered into this dialog.

Clicking the left mouse button when the cursor is in the "hot area" in either the development or the application modes will open the Valve Handle Child Diagram window and shift control to that window. The Child Diagram Window can also be accessed using the flyout menu item in the Diagram Window menu command from the Windows menu, as shown in Figure 15-55.

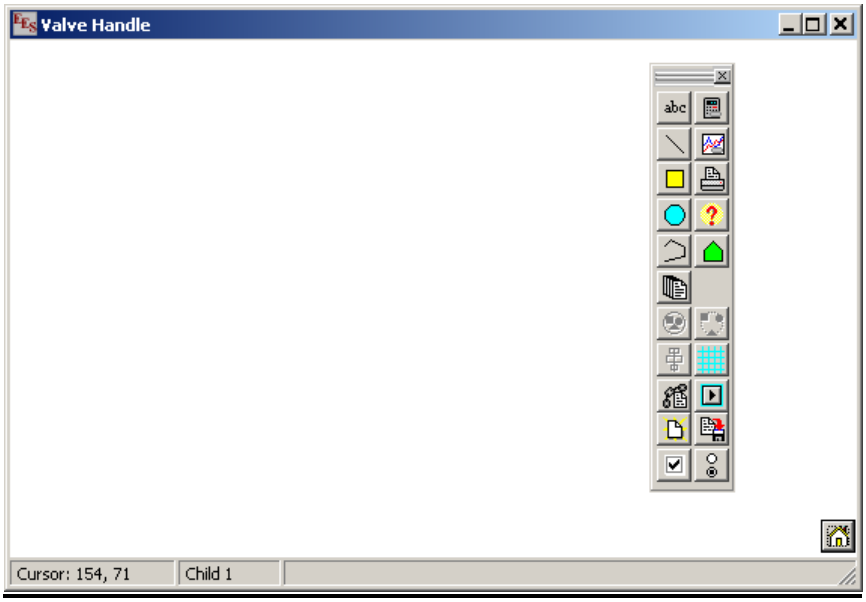


Figure 15-54: Valve Handle Child Diagram Window.

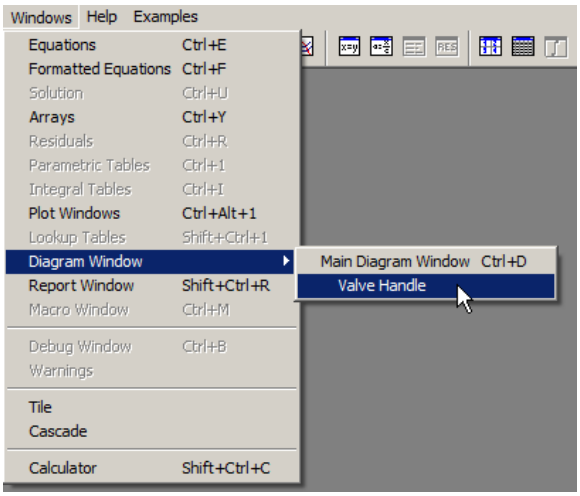


Figure 15-55: Diagram Window Menu item in the Windows menu provides access to the Child Diagram Windows.

Modify Hot Area Information Dialog

Pressing and holding the Ctrl and Shift keys down while in Development mode will display a red dotted outline around all defined hot areas. Double-clicking the left mouse button when the Ctrl and Shift keys are pressed and the cursor is positioned in the hot area rectangle will access the Modify Hot Area Information dialog shown in Figure 15-56. This dialog allows the name of the Child Diagram Window and the location and size of the hot area that activates the Child Diagram Window to be changed. If you do not wish to provide access to the Child Diagram Window by clicking on the associated hot area then you can make the width and height small and move the hot spot to an obscure location. However, do not set the hot area off of the screen or make it have zero size in case you later need to access the dialog in order to modify its properties or delete the Child Diagram Window.

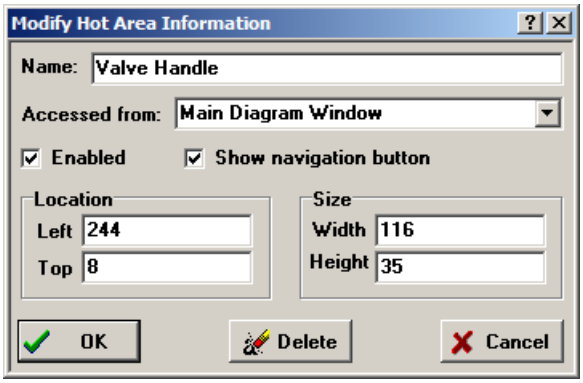


Figure 15-56: Modify Hot Area Information dialog.

Group and Ungroup Selected Items Buttons

The Group Selected Items button ( ) becomes enabled when the Diagram Window is in development mode and two or more graphic or text items are selected; note that only simple text items can be included in a group (not input or output text items). Grouped items act like a single item; they can be resized, moved, copied and pasted. A group can be included in another group to form nested groups. To create a group, select the items that are to be in the group and then click the Group button on the toolbar. Alternatively, right click on one of the selected items and select the Group menu command from the pop-up menu that appears. Group information is saved with other file information when the Save command is issued. To ungroup a previously defined group, click on the grouped item and then click the Ungroup button ( ) on the toolbar. Alternatively, right click on the grouped item and select the Ungroup menu command from the pop-up menu.

Align Selected Items Button

The Align Selected Items button ( ) becomes active when two or more graphic or text items are selected; note that both simple text items and input output text items can be aligned. Clicking the Align Selected Items button on the Diagram Window tool bar will access the Alignment dialog shown in Figure 15-57, which provides a number of ways in which the selected objects can be aligned.

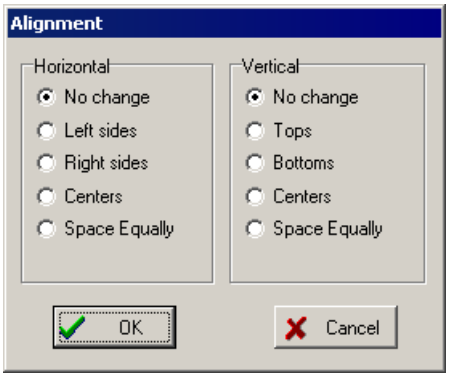


Figure 15-57: Align Selected Objects dialog.

Showing a Grid on the Diagram Window

The Show Grid button ( ) on the Diagram tool bar draws a grid when the Diagram Window is in development mode. Clicking this button will display the Diagram Grid Information dialog in Figure 15-58 which allows the grid spacing and color to be specified. The Show Grid button will appear in a depressed state while the grid is visible. To eliminate the grid, simply press the button again.

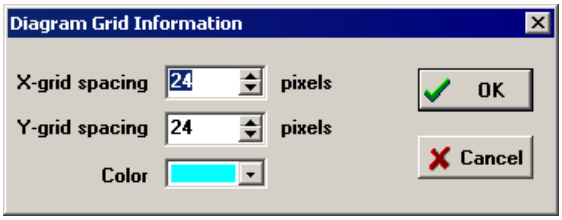


Figure 15-58: Diagram Grid Information dialog.

Creating Links

Link buttons can be placed on a Diagram Window or a Child Diagram Window by selecting the Add Link button ( ) in the tool bar. The Link Button Properties dialog shown in Figure 15-59 will appear after clicking the Add Link button on the tool bar. The caption and link properties can be specified and assigned to a button that will appear on the foremost Diagram Window. In the pull-down window that lists the types of link buttons there is an option for a transparent button that can be placed anywhere in the Diagram or Child Diagram windows. The transparent button allows part or all of existing graphic objects or text to serve as a button. The caption and properties can be changed at any time when the Diagram Window is in development mode by either double-clicking the left mouse button or clicking the right mouse when the cursor is positioned over the button.

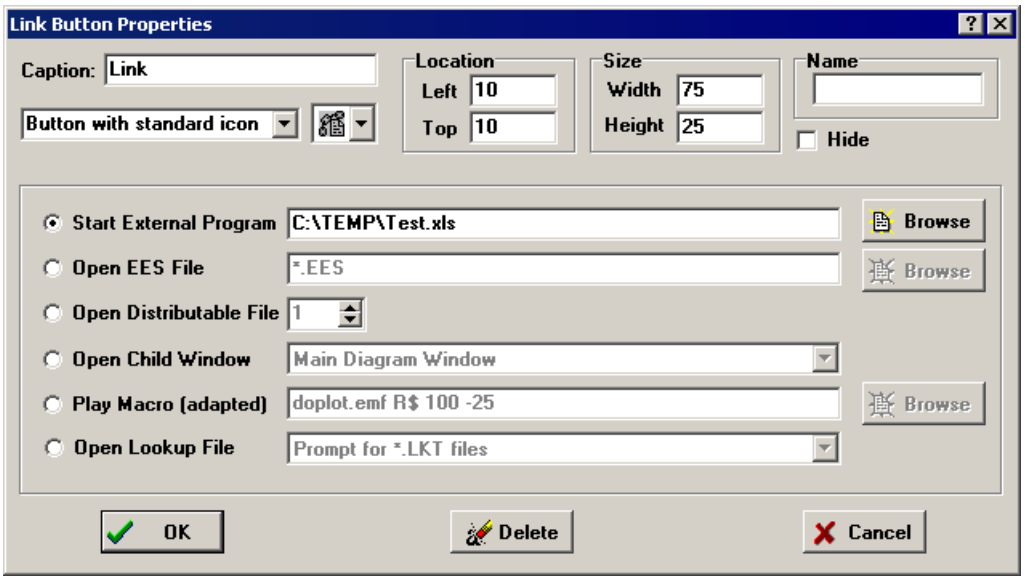


Figure 15-59: Link Button Properties dialog.

Six different types of links can be created, as discussed below.

Link type 1: Start an External Program

This link type allows any program, including another instance of EES, to be started when the user clicks the link button in the Diagram Window while in application mode. The link button property can be the full filename and optionally, the name of the file that is to be opened when the application starts. Alternatively, just the name of the file that is to be opened is needed if the application information is provided in the operating system registry, as is usually the case. As an example, entering C:\Temp\Test.xls in the space to the right of the Start External Program button will start Microsoft Excel and load file C:\Temp\Test.xls, when the Link button is clicked, assuming that this file exists.

Link type 2: Open EES File

This link type will close the existing EES file and open the EES file specified as the link button property. EES will check to see that the current EES file is saved, and if not, ask for confirmation before closing the existing file. This option is useful when the results of one EES program provide data for use in another EES program. For example, information can be calculated and saved in a Lookup Table that is automatically loaded in the next EES program with the $OpenLookup directive.

Link type 3: Open Distributable File

This link type is applicable only to distributable programs created with the Make Distributable Program command, described in Chapter 17. Distributable programs can include up to 100 different EES files. As with option 2, this link type allows one EES file to open another, but in the case of a distributable program the EES file must be one of the files that can be included with the distributable program. The file is referred to by its number, 1 to 100.

Link type 4: Open Child Diagram Window

This link type is applicable if one or more Child Diagram Windows have been created. The list box to the right of the Open Child Window button will show the names of all existing Child Diagram Windows. Clicking the button will open the specified Child Diagram Window, just as if the user had clicked the left mouse within the hot area for the Child Diagram Window.

Link type 5: Play Macro adapted with EES variables

This link type will play a predefined Macro file that has been stored with a .emf filename extension. The capabilities of this link type will be clearer after you read the information concerning macros in Chapter 19. The first parameter that must be supplied in the edit box is the macro filename, including the .emf filename extension. All other parameters are optional. It is possible to provide information to the macro file that allows its capability to be programmed. This is done by providing parameters after the macro file name separated with a space or comma. These parameters can be EES variable names, numerical values, or strings. EES will process the Macro file looking for the character sequence %%N where N is an integer value. If this sequence is found, then the parameter %%N in the macro is replaced with the N^{th} parameter in the list.

For example, this capability can be used in the throttle example presented in this chapter to prepare a property plot that is specific to the fluid that is selected by the user from the drop-down menu. To do this, create a macro file with a text processing program such as Notepad in the same directory as the EES file for the turbine example. The name of the macro file should be doPropPlot.emf and it should contain the following two lines:

```
PropPlot %%1 PH 2 %%2 %%3 0 DoQLines
OverlayPlot Name=0 Table=ARR1 X=h[i] Y=P[i] Rows=1..2 Line=1 Symbol=10 Color=red RIGHT
```

Again, the macro commands will become more understandable after reading Chapter 19. Next, create a link button with the Play Macro link information shown in Figure 15-60.

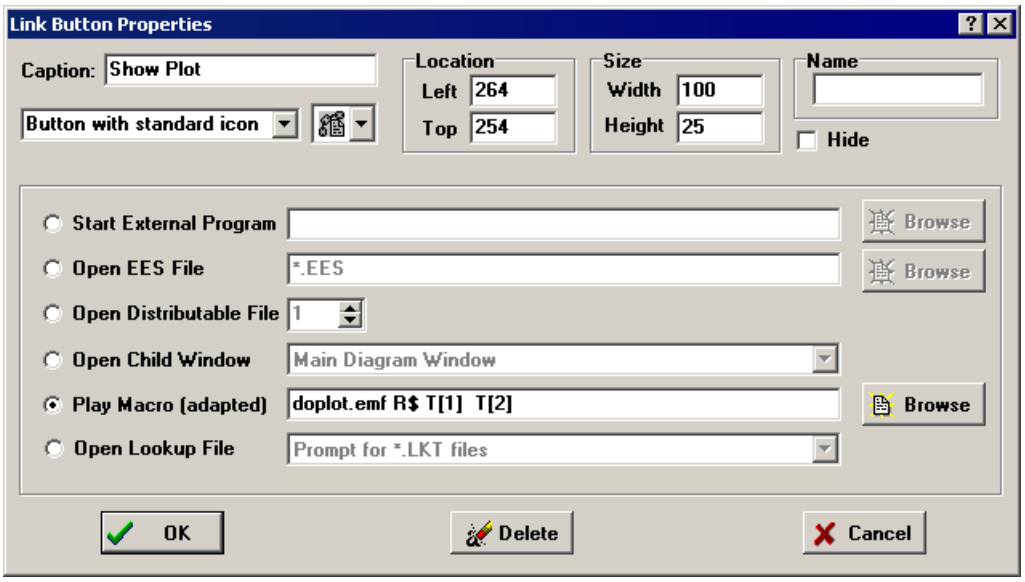


Figure 15-60: Configuring a link button to show a property plot for the selected fluid.

EES will replace %%1 in the macro file with the value of the EES string variable R$. The parameters %%2 and %%3 in the macro file will be replaced with the values T[1] and T[2], respectively. Solve the problem by clicking the Calculate button. Then click the Show Plot button, which will display a P-h plot for the selected fluid with a superimposed line connecting states 1 and 2.

Link type 6: Open a Lookup Table File

A Link button can be placed on the Diagram Window that, when clicked, will display the familiar Open File dialog. A user can then select a Lookup Table file having a filename extension of .lkt, .txt, or .csv and that Lookup Table file will be opened. After the selection has been made, an EES string variable having the same name as the caption of the Link button is created and its value is set to the name of the Lookup Table that has been read in. EES does not allow spaces in a variable name, so any spaces in the caption are set to underscores. Also, the maximum length of the string variable is 29. The last character of the EES variable name is set to $ to indicate that it is a string variable. As an example, if the caption of the Link button is "Open Lookup File", clicking the button will read the file, create variable Open_Lookup_File$ and set the value of this string variable to the name of the Lookup Table that was created when the button is clicked. If the Lookup Table already exists, it will not be re-read.

Clicking OK in the Link Properties dialog will create the link button. The button can be dragged to any location while the Diagram Window is in development mode or moved with the arrow keys. Right-clicking on the link button while in development mode will bring up the Link Properties dialog so that link properties can be changed or the button can be deleted. In this dialog, the Link button can be given a name. The name is used to dynamically control the button attribute for animation, as explained in Chapter 16.

Adding an Audio-Visual Item

Audio-visual items such as movie files (having an .avi, .wmv or .mpg file name extension) or sounds (having a .wav filename extension) can be placed on the Diagram Window or Child Diagram Windows by selecting the Add Audio-Visual item button ( ) in the toolbar. The Audio-Visual File properties dialog shown in Figure 15-61 will appear.

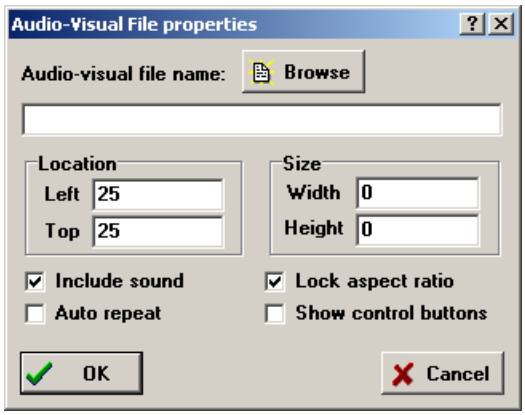


Figure 15-61: Audio-Visual object dialog.

The Browse button facilitates selection of the audio-visual file. If a movie file is selected, EES will automatically fill in the Width and Height of the movie. These fields can be changed. If the Lock aspect ratio button is selected, changing either width or height will result in an automatic change in the height or width to ensure that the original aspect ratio remains unchanged.

If the Show control buttons is checked then a control bar ( ) will be placed below the audio-visual item. The bar has the familiar play, pause, stop, step, previous control buttons. If the Show control buttons control is not checked then the movie is controlled by the left and right mouse buttons. To play, click the left mouse button within the item rectangle while in Application mode. To pause a playing item, click the left mouse button within the item rectangle. Clicking the right-mouse button stops a playing movie and resets it to the beginning. The audio-visual control information can be viewed or changed by right-clicking in the item rectangle while in development mode. There are many reasons to add an audio-visual file to a Diagram Window; for example, audio-visual materials can be prepared for training purposes and made accessible as a movie that can be viewed in the Diagram Window.

Saving and Loading User Inputs

Figure 15-44 shows a Diagram Window with input variables that are entered by the user. In some situations, it would convenient to save this set of inputs and later restore them. The Professional version allows Save and Load Inputs buttons (by , respectively) to be placed on the Diagram Window (but not on a Child Diagram Window) by selecting these buttons from the Diagram Window tool bar. Saving inputs is not required in most EES programs since EES saves all program information, including the inputs on the Diagram Window, when the Save command is selected from the File menu. However, in some cases it may be useful to save different sets of inputs corresponding to different designs or nominal conditions. In addition, the ability to save and reload inputs is essential in the development of Distributable programs

described in Chapter 17 since the Save command in the File menu is not enabled for Distributable programs.

Clicking the Add Save Inputs button on the tool bar will bring up the Save Button Information dialog shown in Figure 15-62. The Add Load Inputs button will display a similar dialog. The button caption, location, and size can be specified in the usual manner. The Name field is used for animation, as described in Chapter 16. The FileName$ fields shows the current value of the variable SaveButton.FileName$, which is automatically set by EES to the name of the file to which the inputs were last saved. This EES string variable can be display on the Diagram Window as a text item in the same way as any other EES variables. The name can also be shortened by removing the path information using the ShortFileName$ EES function.

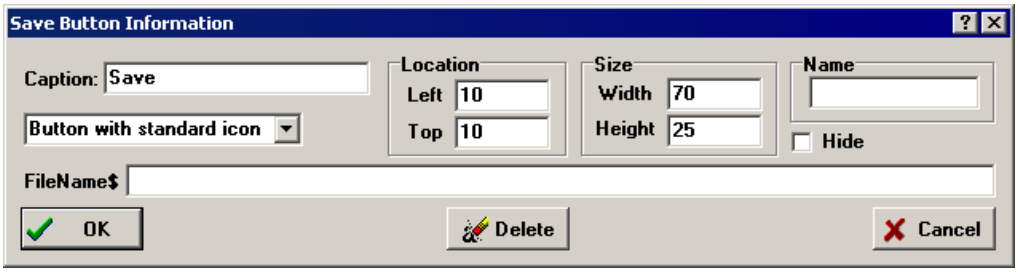


Figure 15-62: Save Button Information dialog.

Clicking OK will place the Save Inputs or Load Inputs button on the Diagram Window in a default location. The button can be moved to another location using the arrow keys or with the mouse in the usual manner.

The Save and Load buttons can only be used in application mode. The Save button is disabled until calculations have been successfully completed. In this way, it is not possible to write a set of input information that will not successfully calculate. Both the Save and Load buttons will be disabled if the Diagram Window does not contain any input variables.

When the user clicks the Save button in application mode after calculations have been successfully completed, a standard file save dialog will appear in which the name of a file can be entered. The default name is the same name as the EES file with a filename extension of .var. The filename and directory information can be changed but the filename extension must be .var in order for EES to recognize the file. If confirmed, a file containing the current values of all EES variables in the main program, including the current values of the inputs, is written to the disk. The .var file for the throttle example considered in this chapter is shown in Figure 15-63. A .var file is a text file that can be viewed and edited in any text editor. The information contained in this file includes the variable name, guess value, lower and upper limits, display format, highlight indicator and units for every variable in the main EES program.

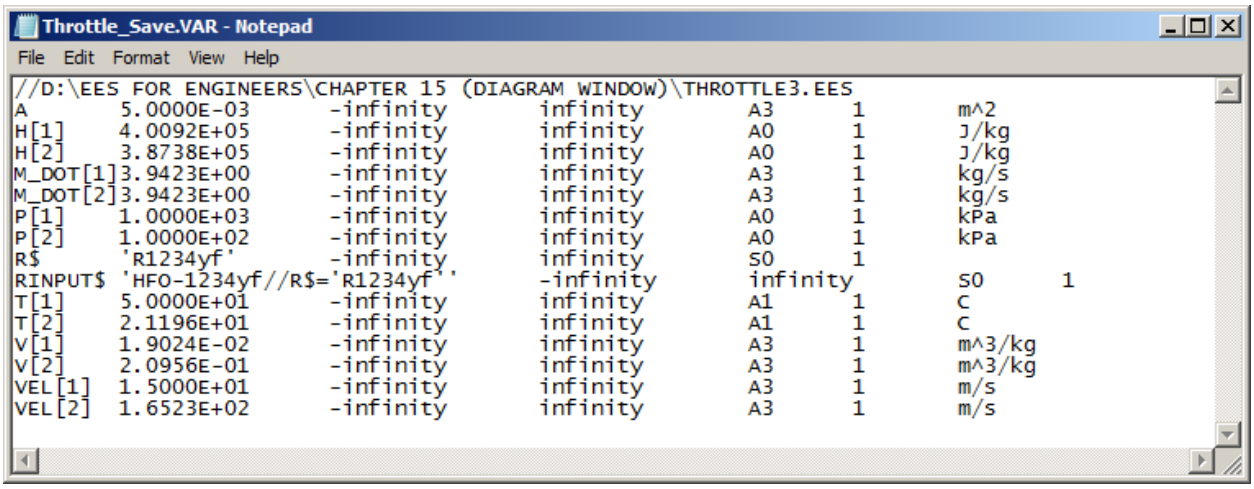


Figure 15-63: Contents of the .var file for the Throttle example.

This .var file can later be loaded with a Load button that has been placed on the Diagram Window. (The .var file can also be read in the Variable Information dialog by clicking the Open Variable Info file icon that is located at the upper right of the Variable Information window.) A standard file open dialog will appear in which a .var file can be selected. The Diagram Windows will be updated with the values of the variables found in this file.

Add Check Box Item

A check box input can be placed on the Diagram Window clicking the Add Check Box item button ( ) on the Diagram Window tool bar. A check box allows the equation set that EES solves to be modified depending on the check box setting.

As an example, we will add a check box to the Diagram Window shown in Figure 15-50 that will allow us to include or exclude the kinetic energy effects in the throttle calculation. Place the Diagram Window into development mode by pressing Ctrl-D if necessary. Select the Add Check Box button on the toolbar. A Check Box Characteristics dialog will appear that allows the caption text, location on the Diagram Window, and font to be specified. In our example, the caption should read Include kinetic energy effects. Next we need to enter the equations that EES will solve (together with the all of the other equations in the Equations Window) when the check box is unchecked and when it is checked. In our case, we need to enter the energy balance for the throttle without kinetic energy effects when the button is unchecked and with kinetic energy effects when it is checked, as shown in Figure 15-64. Note that one or more equations can be entered for each check box condition including comments. If "Execute EES commands when check status is changed" is selected, EES will attempt to execute the commands in the unchecked or checked window, depending on the setting of the check box, directly after the change is made to the check box in application mode. This capability is useful when the check box setting affects the display in the Diagram Window in some way, but it is not needed in this example.

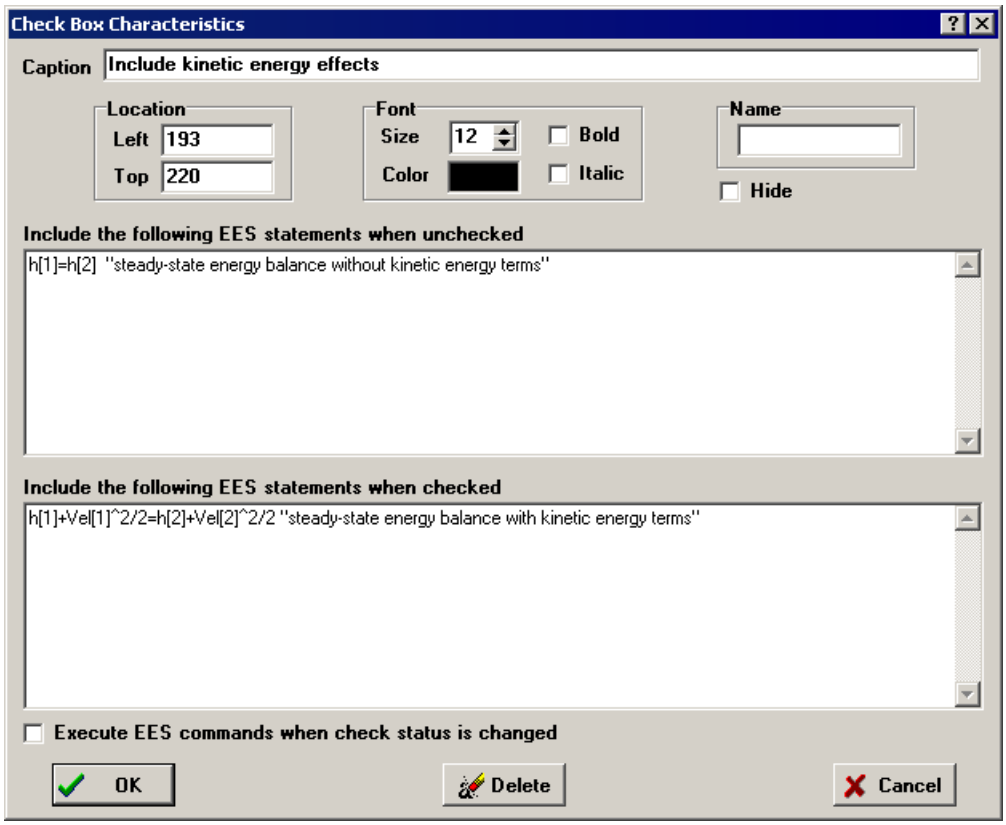


Figure 15-64: Check Box Characteristics dialog with information entered in order to selectively consider kinetic energy effects.

Click OK. Drag the check box to the desired location. Changes to the check box characteristics can be made later by either double-clicking the left mouse button or clicking the right mouse button on the check box in development mode after selecting the check box. Alternatively, click the check box to select it and then click the Add Check Box item button on the Diagram tool bar to edit the check box.

Hide the tool bar. It is next necessary to comment out the energy balance that was entered in the Equations Window, or alternatively, surround it with $IfNot DiagramWindow / $EndIf directives.

```
$IfNot DiagramWindow
    h[1]+Vel[1]^2/2=h[2]+Vel[2]^2/2                    "steady-state energy balance"
$EndIf
```

Return to the Diagram Window and test the interface with the check box control. Figure 15-65 shows the Diagram Window after calculations are completed without including kinetic energy effects. Note that the specific enthalpy of the refrigerant does not change in this case. Compare this result to the result shown in Figure 15-50 which considered kinetic energy effects.

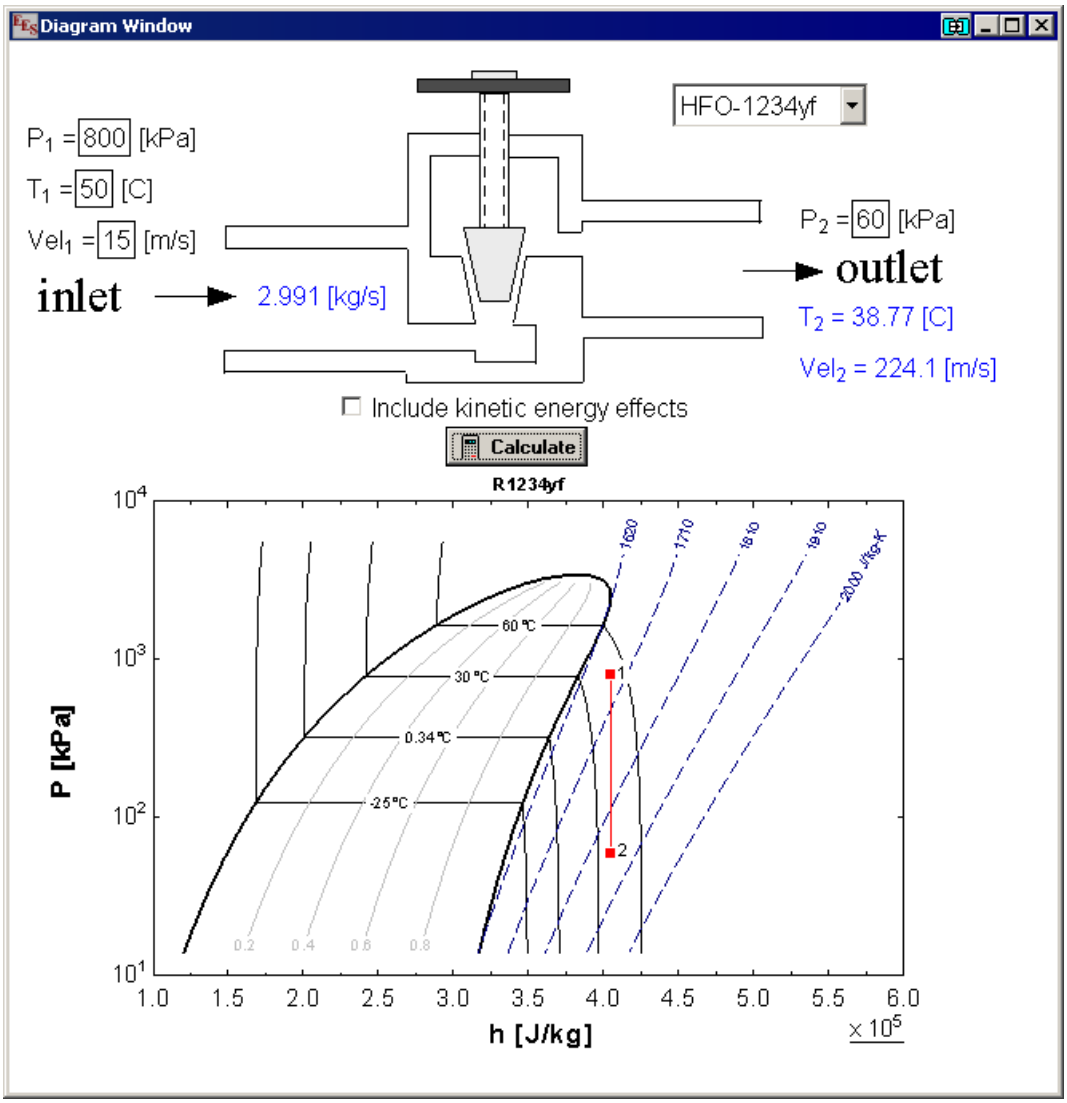


Figure 15-65: Diagram Window with results that exclude kinetic energy effects. Compare this result to the result shown in Figure 15-50.

Creating and Using Radio Button Groups

A radio button group is placed on the Diagram Window using the Add Radio Button Group button ( ) on the Diagram Window Tool bar. A radio button group provides up to 8 radio buttons. Each button is associated with a set of EES commands that are enabled when the corresponding radio button is selected. The Radio Button Characteristics dialog shown in Figure 15-66 will appear after clicking the toolbar button.

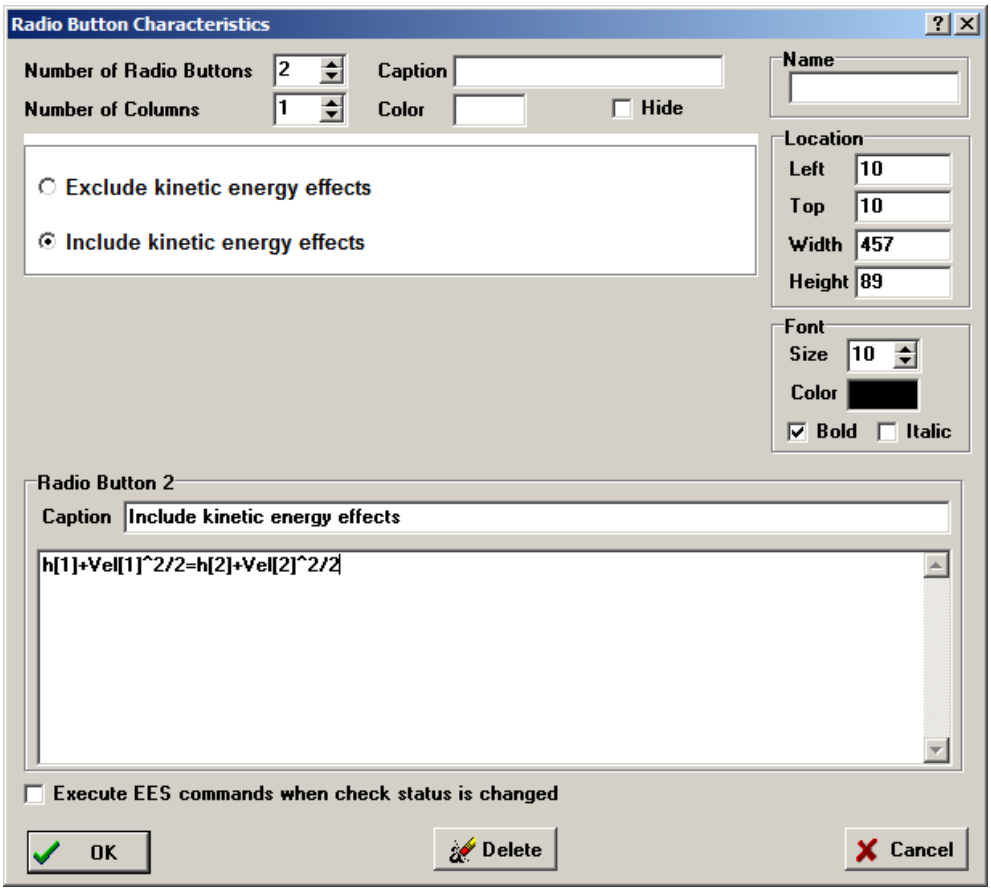


Figure 15-66: Radio Button Characteristics dialog.

The number of radio buttons is set with the spin control in the upper left corner of the dialog, or by entering the value. The number of buttons can be between 2 and 8. The number of columns controls how the buttons are positioned on the Diagram window. The number of columns can be between 1 and 4. An optional group caption can be entered for the radio button group. The background color, width, height, location, and font characteristics can also be specified. Changing any of these controls will cause the radio button in the dialog window to be redrawn as it will appear in the Diagram Window.

A radio button group operates in a manner that is similar to a check box. Each button is associated with EES equations. The caption and EES equations for the currently selected button appear at the bottom of the dialog. Both the caption and the EES equation for this button can be modified. For example, we could have implemented the selective consideration of kinetic energy using a Radio Group instead of a check box. To do so, we would first select the number of radio buttons to be 2. Next we need to change the caption for Button 1 to be Exclude kinetic energy effects and enter the energy balance without kinetic energy terms in the edit box below the caption. Click on Button 2 and change its caption to be Include kinetic energy terms and enter the energy balance with kinetic energy terms in the edit box below the caption. The result should be as shown in Figure 15-66.

If "Execute EES commands when check status is changed" is selected in the dialog then EES will attempt to execute the commands associated with the selected button immediately after it is selected. This capability is most useful when the check box setting affects the display in the Diagram Window in some way. For example, a graphic or text item button can be hidden by setting the EES variable, Name.Hide=true# where Name is the name give to the object. This type of animation capability is discussed in Chapter 16.

After clicking the OK button, the radio button group will appear in the Diagram Window. In development mode, the radio button group can be dragged to a new position by pressing and holding the mouse on the group border while moving the group to a new location. If the left mouse button is pressed when the cursor is on the lower right corner of the radio button group then the cursor will change to a resize arrow. Dragging the mouse at this point will allow the width and height to be changed. Right-clicking (or double-clicking the left mouse button) on the border of the radio button group while in development mode will brings up the Radio Button Characteristics dialog so that changes can be made to the item. Alternatively, click the radio button group to select it and then click the Add Radio Group button on the Diagram tool bar.

Using Sliders for Inputting Values

In the Professional version of EES, numerical inputs can entered in the Diagram Window using slider controls, as well as the conventional edit boxes. Slider controls simplify the entry of values in addition to providing control over the range of values that the user can enter. Also, slider controls provide a way to specify integer-only values for an input.

For example, we can use a slider to enter the value of the variable P[2], the downstream pressure for the throttle, in place of the input edit box shown in Figure 15-65. Put the Diagram Window into development mode by pressing Ctrl-D. Select the input box for P[2] and double-click within the selection (or click the Add Text button in the Diagram tool bar with the input box selected) in order to access the Modify Diagram Text dialog shown in Figure 15-67. Click the Slider input check box. The dialog window will expand to display the parameters for the slider control at the bottom of the dialog. In this example, we need to restrict the downstream pressure between 100 kPa and the inlet pressure. Enter 100 kPa for the lower limit and P[1] for the upper limit on the pressure. Note that the limits entered for a variable in this dialog are the lower and upper bounds that are normally entered with the Variable Information Window.

With the slider set up as shown in Figure 15-67, only integer values will be provided for P[2]. The slider control width will be 100 pixels (approximately 1 inch). The width can be varied from 60 to 200 pixels. Also, the value that the slider control provides, along with the name of the variable and its units will be displayed directly above the slider control. Click OK to exit the dialog. The Diagram Window with the slider control is shown in Figure 15-68.

The slider operates just like other Diagram Window objects. It can be dragged to a new position by pressing and holding the left mouse button down on the control while sliding the control to the desired location while the Diagram window is in development mode. The characteristics of the slider control can later be changed while in development mode by double-clicking the left mouse button (or by clicking the right mouse button) while the cursor is positioned on the slider

control. Sliding the control from its left position to its right position linearly changes the value of the variable from its minimum to its maximum value. The changes in the variable value occur immediately.

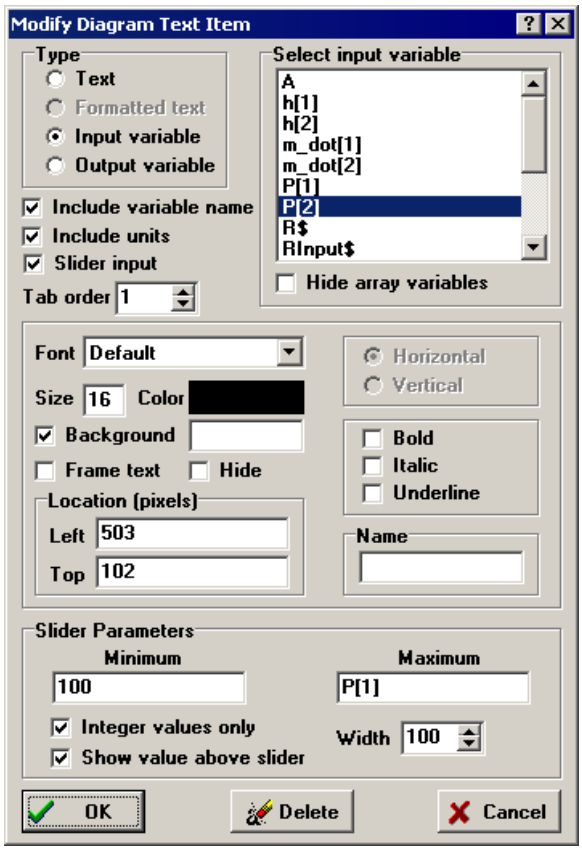


Figure 15-67: Entering slider input information for variable P[2].

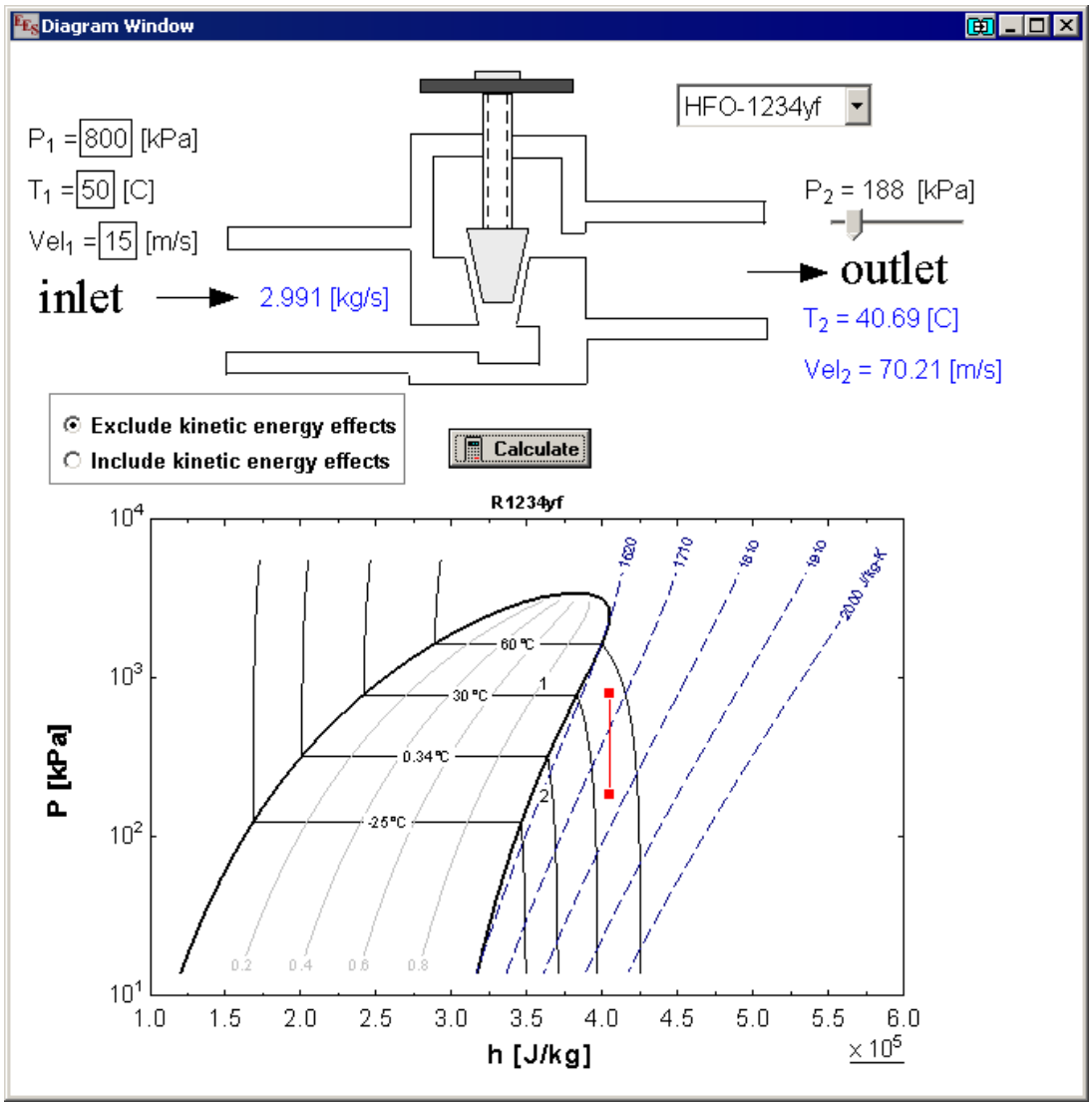


Figure 15-68: Diagram window with a slider control used to input the downstream pressure.

16 DIAGRAM WINDOW ANIMATION

Chapter 15 describes how graphic and text objects can be placed on the Diagram Window. EES input and output variables can also be placed on the Diagram Window, allowing it to serve as a graphical user interface to an EES program. This chapter presents additional capabilities of the Diagram Window that are provided in the Professional version of EES. Every graphic and text object has attributes, such as its size, location, rotation angle, visibility, and color. In the Professional version, these attributes can be assigned to EES variables so that the characteristics of the objects and therefore the appearance of the Diagram Window can be controlled programmatically. At the simplest level, this capability can be used to hide or show an object, depending on other choices that are made in the Diagram Window. At the other extreme, a detailed animation that is based on an analysis programmed in the Equations Window can be displayed in the Diagram Window.

16.1 Controlling the Attributes of Diagram Window Objects

Every object in the Diagram Window can be optionally assigned a name in the Professional version of EES. The attributes for the object can then be accessed and controlled using this name. For example, start EES and bring the Diagram Window to the front by selecting Diagram Window from the Windows menu. The Diagram Window should be in development mode (with the Diagram Window tool bar showing); if it is not in development mode then enter Ctrl-D.

Attributes of Rectangle and Ellipse/Circle Objects

Click the Add/edit rectangle button () on the tool bar and then draw a rectangle on the Diagram Window by pressing the left mouse button at the upper left corner and moving the mouse to the lower right corner while holding the mouse button down. Double-click the left mouse button within the rectangle to bring up the Diagram Object Characteristics dialog, shown in Figure 16-1. This dialog provides access to all of the attributes of the rectangle object. Set the line type to the thicker line and the fill type to Opaque.

The Name edit box located in the lower right corner of the dialog allows you to enter the name of this object. Enter a name in this field, such as Rect1. Note that, as soon as you enter the name, many of the attribute labels in the dialog are redisplayed with an underline, as shown in Figure 16-1. The underlined labels are associated with those attributes that can be assigned to EES variables. A list of the programmable attributes associated with the rectangle object is provided in Table 16-1. (The attributes for an object created with the Add ellipse or circle button on the tool bar are the same as those of the rectangle object.)

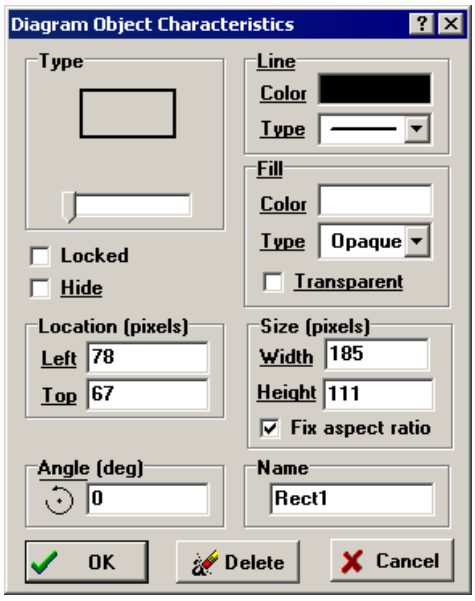


Figure 16-1: Diagram Object Characteristics dialog.

Table 16-1: Programmable attributes for a rectangle or ellipse object.

Attribute Name[1]	Description
Name.LineColor	Color variable indicating the color of the border
Name.FillColor	Color variable indicating the color of the fill
Name.LineType	Integer ranging between 1 and 4 as defined in Table 16-2
Name.FillType	Integer ranging between 0 and 7 as indicated in Table 16-3
Name.Hide	Boolean (either True# if hidden or False# if visible)
Name.Transparent	Boolean (either True# if transparent or False# if opaque)
Name.Left	Integer (expressed in points) indicating location of left edge
Name.Top	Integer (expressed in points) indicating location of top edge
Name.Width	Integer (expressed in points) indicating width of rectangle
Name.Height	Integer (expressed in points) indicating height of rectangle
Name.Angle	Integer (0 to 360°) indicating the rotation of the object, increasing angle turns the object more counter-clockwise

1. The parameter Name indicates the name of the rectangle or ellipse object.

The programmable attributes have a compound name, with the first part being the name entered in the Name field for the object (shown as Name in Table 16-1) and the second part, separated by a decimal point, being the attribute descriptor. Note that these attribute names are EES variables. The name you provide in the Name field must conform to the EES variable convention which requires that the name begin with a letter and not include any spaces. Also, the total length of the compound variable name must be less than 30 characters; therefore, the names that you enter in the Name field should be relatively short. Click OK to close the dialog. Note that array variables can be used; in this case, the array index must be placed at the end of the variable name so that EES recognizes the variable name as an array variable. If, for example, you enter a name for the object to be Rect[1] then the line color attribute would accessed using EES variable Rect.lineColor[1].

The LineColor and FillColor attributes are color variables corresponding to the RGB color standard. Many common colors are provided as EES constants, e.g., Black#, Blue#, Green#, Red#, White#,

and Yellow#. Note that all constants have a # character as the last character in their name. Colors can also be assigned using the RGB function. The RGB function returns the value of color formed by providing the red, green, and blue color component values. The component values can range from 0 to 255. For example, RGB(255,0,0) returns pure red, RBG(0,255,0) returns pure green, and RGB(0,0,255) returns pure blue. The RGB function allows combinations of red, green, and blue so that any color can be described.

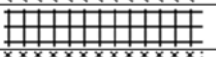
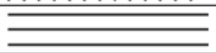
The LineType attribute is an integer ranging between 1 and 4 corresponding to the line types defined in Table 16-2.

Table 16-2: Line types for a rectangle, ellipse or polyline/polygon object.

LineType	Description
1	Thin solid line
2	Thick solid line
3	Thin dotted line
4	No line

The FillType attribute is an integer ranging between 0 and 7 corresponding to the fill patterns defined in Table 16-3.

Table 16-3: Fill types for a rectangle, ellipse or polyline/polygon object.

FillType	Description
0	Clear
1	
2	
3	
4	
5	
6	
7	

The hide attribute is a Boolean value that should be either 0 (for false) or 1 (for true). Note that any value that is non-zero will be assumed to imply true. It is convenient and more readable to use the EES constants True# and False# when setting the hide attribute.

The position of the rectangle in the Diagram Window is specified with its left and top attributes. These attributes are expressed in pixels. Most computer systems are configured so there are 96 or 120 pixels per inch. A left position of zero corresponds to the left side of the Diagram Window and increases moving to the right. A top position of zero corresponds to the top of the Diagram Window and increases moving downwards. The Width and Height attributes determine the size of the rectangle and are also specified in pixels. EES provides the functions DiagramHeight# and DiagramWidth# that return the height and width of the specified Diagram or Child Diagram Window. The PixelsperInch# function allows the position and size to be computed in other units and converted to the pixel values needed in the Diagram Window.

Specifying the angle attribute rotates the object about its center. The angle is specified in degrees, regardless of the degree-radian setting in the Unit System settings. The object turns counter-clockwise as the angle is increased.

The attributes of a rectangle object can be set directly in the Equations Window. First, hide the tool bar so that the Diagram Window is in application mode. Then arrange the positions and sizes of the Diagram and Equations Windows so that you can see both windows on your screen, as shown in Figure 16-2. Enter one or more of the equations shown in Figure 16-2 and press F2 to initiate the calculations. You should see the Diagram Window instantly update so that its attributes are as specified in the Equations Window. This capability provides the basis of the animation capabilities in EES.

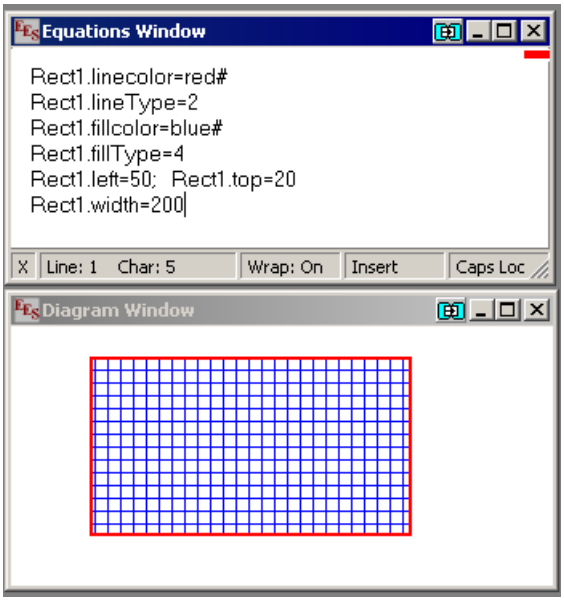


Figure 16-2: Specifying the attributes of a rectangle.

Attributes of Line Objects

A line object is placed on the Diagram Window using the Add line button ( ) from the Diagram Window tool bar. Double-clicking the left mouse button while the line is selected allows access to the Diagram Line Characteristics dialog where the line can be assigned a name in the Name field. The attributes for a line object are listed in Table 16-4 where Name represents the name of the line object. The X1, Y1, X2, and Y2 attributes allow the locations of end points of the line to be specified.

Table 16-4: Programmable attributes for a line object.

Attribute Name[1]	Description
Name.X1	Horizontal location (in pixels) of left end of line
Name.Y1	Vertical location (in pixels) of left end of line
Name.X2	Horizontal location (in pixels) of right end of line
Name.Y2	Vertical location (in pixels) of right end of line
Name.Color	Color defined with the RGB function or color constants.
Name.LineType	Integer ranging between 1 and 8 as defined in Table 16-2
Name.Hide	Boolean (either True# if hidden or False# if visible)

1. The parameter Name indicates the name of the line object.

Table 16-5: Line types for a line object.

LineType	Description
1	————————
2	————————
3	- - - - - - - - - - - -
4	— — — — — — —
5	- - - - - - - - - - - -
6	— — — — — — —
7	- — - — - — - — - -
8	- — - — - — - — - -

Attributes of Polyline and Polygon Objects

The attributes of polygon and polyline objects are listed in Table 16-6 where Name represents the name of the polyline or polygon object.

Table 16-6: Programmable attributes for polyline and polygon objects.

Attribute Name [1]	Description
Name.LineColor	Color variable indicating the color of the border
Name.FillColor	Color variable indicating the color of the fill
Name.LineType	Integer ranging between 1 and 4 as defined in Table 16-2
Name.FillType	Integer ranging between 0 and 7 as indicated in Table 16-3
Name.Hide	Boolean (either True# if hidden or False# if visible)
Name.Transparent	Boolean (either True# if transparent or False# if opaque)
Name.Left	Integer (expressed in pixels) indicating location of left edge
Name.Top	Integer (expressed in pixels) indicating location of top edge

1. The parameter Name indicates the name of the polyline or polygon object.

Attributes of Text Objects

Text items entered with the Add text button on the Diagram Window tool bar have programmable attributes that are listed in Table 16-7, where Name indicates the name of the text object.

Table 16-7: Programmable attributes for text objects.

Attribute Name [1]	Description
Name.Color	Color variable indicating the color of the text
Name.Hide	Boolean (either True# if hidden or False# if visible)
Name.Left	Integer (expressed in pixels) indicating location of left edge of text
Name.Top	Integer (expressed in pixels) indicating location of top edge of text
Name.Text$	String variable with the text for a plain text object

1. The parameter Name indicates the name of the text object.

A text object provides some of the same attributes as the graphic objects, e.g., Left, Top, Color and Hide. In addition, the text itself can be programmatically specified using the Text$ attribute. This capability can be used to display customized messages in the Diagram Window.

A very simple program will be used to illustrate this capability. Open EES and enter:

```
X=1
```

in the Equations Window. Now, move to the Diagram Window and place it in development mode by entering Ctrl-D so that the tool bar is visible. We will add two text items by selecting the Add Text button on the tool bar. The first text item is an input variable text object for the variable X. The second item is a text object that displays a customized message depending on the value of X. In this example, one message will be displayed if the variable X is less 0 and a different message will be displayed if X > 100. Name the text object Message, as shown in Figure 16-3

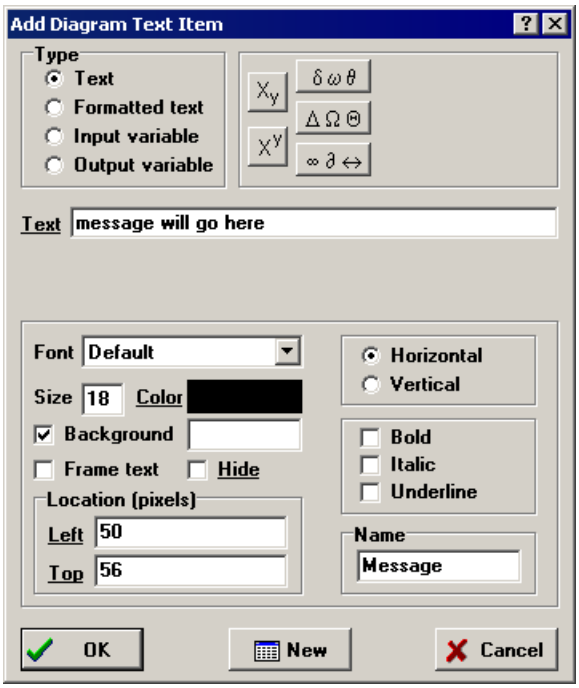


Figure 16-3: Add Diagram Text Item dialog showing entry for a text item named Message.

Hide the tool bar to enter application mode and return to the Equations Window. Delete the equation that sets X = 1 and enter the following Procedure and Call statement. Note that the

procedure SetMessage has inputs X and sets the values of two outputs, Message$ and Color. The string variable Message$ is assigned a string corresponding to the message to be displayed and the numerical variable Color is assigned a value corresponding to the color of that message. Calling the procedure SetMessage with Message.Text$ and Message.Color as the outputs sets their values and therefore changes the attributes of this text item in the Diagram Window.

```
$TabStops 0.25 0.5 0.75

Procedure SetMessage(X:Message$,Color)
      Message$=' '
      Color = Black#
      If (X > 100) Then
            Color = Red#
            Message$ = 'X must be less than or equal to 100'
      Else
            If (X < 0) Then
                  Color = Blue#
                  Message$ = 'X must be greater than or equal to 0'
            EndIf
      EndIf
End

{X=1}
Call SetMessage(X: Message.Text$,Message.Color)
```

Return to the Diagram Window and place it in application mode in order to test the program. Enter a value of 200 for X and press F2 to solve. You should see a red message in the diagram window, as shown in Figure 16-4. A blue message will appear when you press F2 if a value of X less than zero is entered. There will not be a message is X is between 0 and 100.

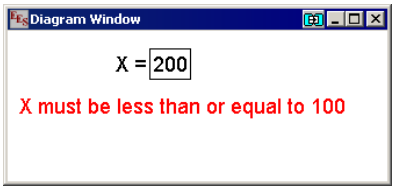


Figure 16-4: Customizable text message on the Diagram window.

Attributes of Button, Radio Group and Check box Objects

Several types of buttons can be placed on the Diagram Window, as explained in Chapter 15. For example, a calculate button can be used to initiate the calculations. A print button provides direct access to the printer. A help button can provide access to detailed information in a .pdf file or on the internet. A link button can provide access to other programs or run macros. Load and save buttons allow the input data on the Diagram Window to be saved and later restored. All of these buttons have the attributes summarized in Table 16-8, where Name indicates the name of the button. These attributes allow the location of the button and its visibility to be controlled. Radio groups and check boxes have these same attributes.

Table 16-8: Programmable attributes for a buttons, radio groups and check box objects.

Attribute Name[1]	Description
Name.Hide	boolean (either True# if hidden or False# if visible)
Name.Left	integer (expressed in pixels) indicating location of left edge of button
Name.Top	integer (expressed in pixels) indicating location of top edge of button

1. The parameter Name indicates the name of the button, radio group, or check box object.

Showing and Hiding Diagram Window Objects

All Diagram Window objects have a .Hide attribute that can be used to control the visibility of the object in application mode. The .Hide attribute is a boolean variable that can be assigned values of True# (1) or False# (0). For example, to hide an object that has been named ABC, you would enter

```
ABC.Hide=true#
```

You can alternatively enter:

```
ABC.Hide
```

To show the object, enter:

```
ABC.Hide=false#
```

or

```
ABC.Show
```

The alternative .Show and .Hide formats are particularily convenient when used within the Diagram window, as described in the following section.

Internally, EES converts ABC.Hide into ABC.Hide=1 and ABC.Show into ABC.Hide=0. Note that ABC.Hide is an EES variable in the main section of the Equations Window. The variable ABC.Hide will be recognized in the main block of code in the Equations window and anywhere in the Diagram windows. However, it will not be recognized within a function, procedure, module or subprogram. It is often necessary to use a function to conditionally control the visibility of an object. This can be easily done by having the function return a 0 (for false#) or 1 for true#) and assigning this return value to the object's hide attribute, e.g.,

```
function ShoworHide(X,Y)
  if (X>Y) then ShoworHide=True# else ShoworHide=False#
end
...{Set X and Y}
...
ABC.Hide=ShoworHide(X,Y)
```

16.2 Setting Attributes within the Diagram Window

All objects that can be placed on the Diagram (or Child Diagram) Window have attributes that become available when the object is given a name, as explained in the previous section. The attributes can be set using equations in the Equations Window and their effect becomes visible directly after the solution process is completed. However, it is also possible to set attributes of Diagram Window objects directly from the Diagram Window using text drop-down lists, check boxes and radio button groups, as discussed in this section. The capability to set attributes in this way is very powerful and allows the user to develop very sophisticated graphical user interface features and animation capability. This capability will be illustrated with very simple examples in this section.

Setting Attributes with Text Drop Down Lists

As explained in Section 15.2, text strings can be input from a drop-down list in the Diagram Window. The text string can contain hidden text entered after the characters // that is interpreted to be one or more EES equations. The maximum length of this string is 256 characters. These equations can be used to control Diagram Window objects. This capability will be illustrated for a very simple example in which we decide whether to use the Solve command or the Solve Table command based on information provided in the Diagram Window.

Start EES and enter the following equation into the Equations window.

```
Y=X^2
```

Of course, this equation can not be solved without more information. One way to provide this information is with a Parametric Table. Select the New Parametric Table command from the Tables menu and create a Parametric Table with 10 rows containing columns for the variables X and Y. Enter values for X in the table that range from 1 to 10. Select the Solve Table command from the Calculate menu (or press F3) in order to solve the table, as shown in Figure 16-5.

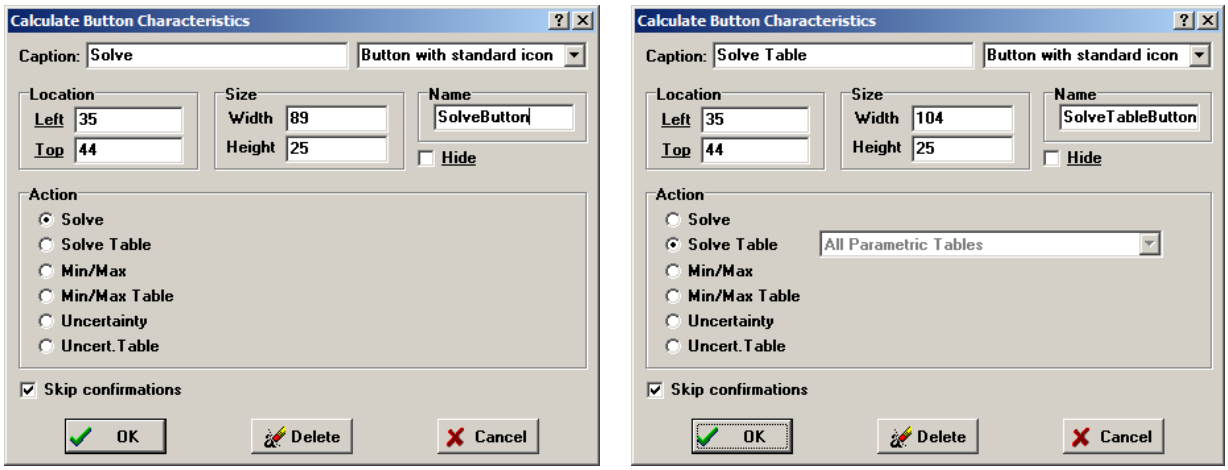
Parametric Table

Table 1

1..10	1 X	2 Y
Run 1	1	1
Run 2	2	4
Run 3	3	9
Run 4	4	16
Run 5	5	25
Run 6	6	36
Run 7	7	49
Run 8	8	64
Run 9	9	81
Run 10	10	100

Figure 16-5: Parametric Table.

Select the Diagram Window command from the Windows menu (or enter Ctrl-D). Make sure that the Diagram Window is in development mode. We are going to place five objects on the Diagram Window. First, add a Calculate button by clicking the Add Calculate button on the tool bar. Set the Caption to be Solve and also name the button SolveButton, as shown in Figure 16-6(a). Note that the left position of the button is set to 35 pixels and the top is set to 44 pixels. Add a second calculate button with the caption Solve Table and give it the name SolveTableButton, as shown in Figure 16-6(b). Set its top position to 44 pixels. Select the Skip confirmations check box for both buttons.



(a) **(b)**

Figure 16-6: Calculate Button dialog for the (a) Solve button and (b) Solve Table button.

Use the Add Text button to make the variable X an input variable. Give the input text item a name, e.g. Xinput as shown in Figure 16-7(a). Use the Add Text button and add an output text item that displays Y; give this output text item a name, e.g., Youtput as shown in Figure 16-7(b).

Select the Add Text button one more time and click the Input variable button. Select *** New String Variable *** from the list and enter DD$ as the name for the string in the box that appears, as shown in Figure 16-8. Click the OK button and the variable DD$ will be displayed in the list. The Include units check box will change to Use string list with and Edit button, as shown in Figure 16-10. Click the Edit button and enter the strings, as shown in Figure 16-9.

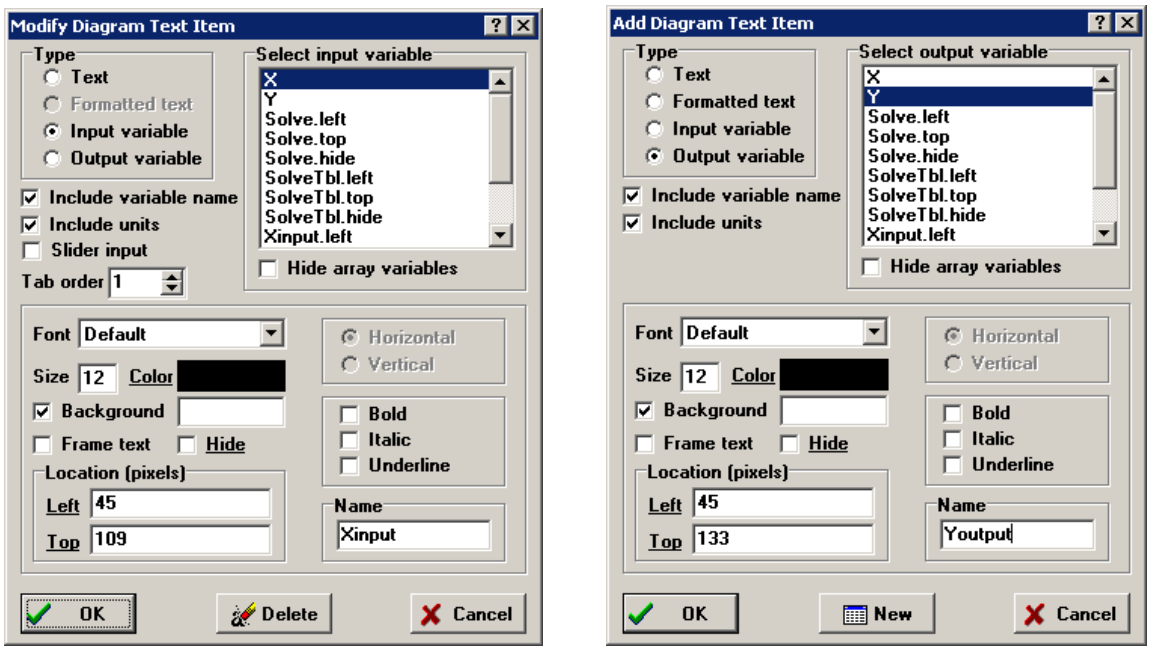


Figure 16-7: Modify Diagram Text Item dialog for (a) the input variable X and (b) the output variable Y.

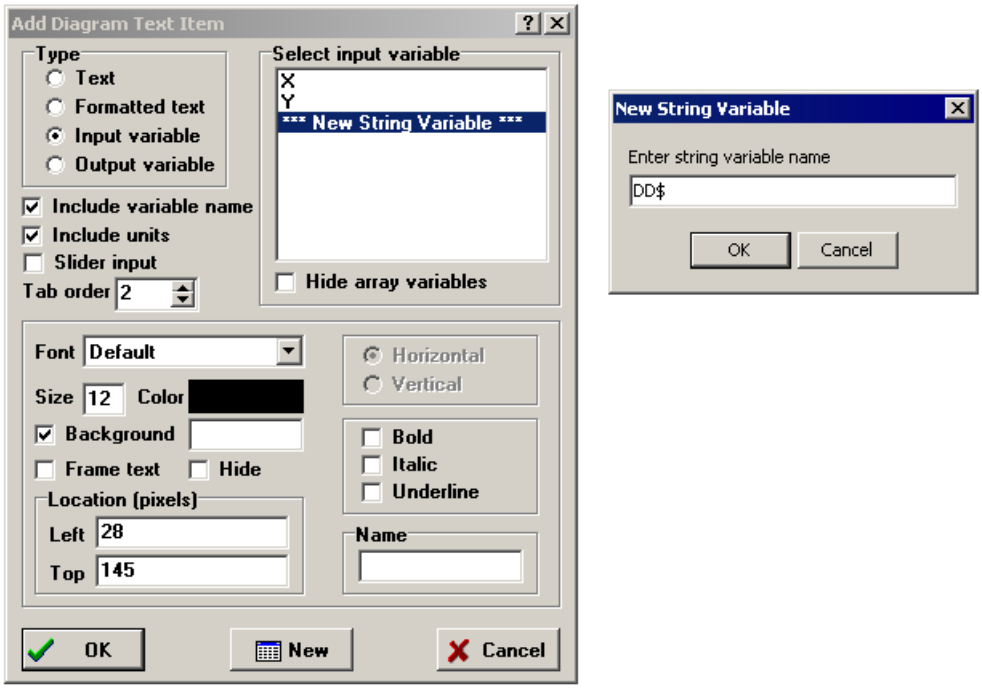


Figure 16-8: Add a new string variable that will be used for drop-down text.

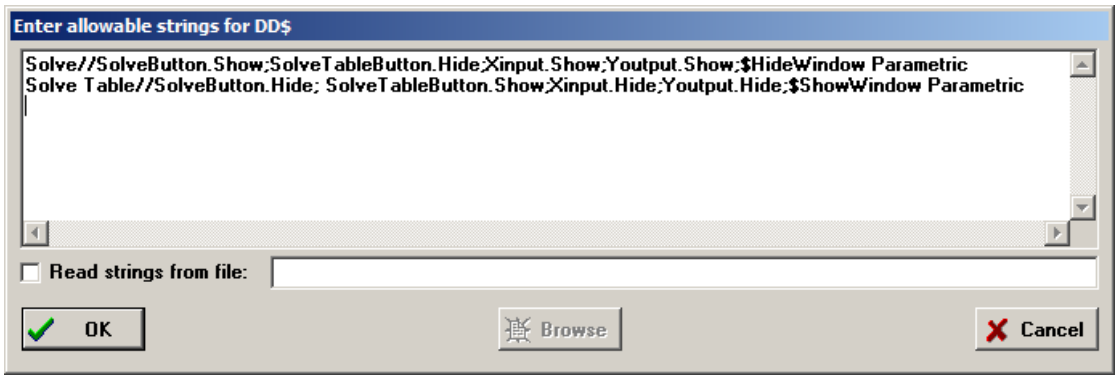


Figure 16-9: Drop-down strings for string variable DD$ which control objects in the Diagram Window.

There is one more change to make. Because the drop-down list is providing control information for other text items on the Diagram Window (Xinput and Youtput), it is necessary for the drop-down list to be the first object that is processed. This is accomplished by setting its Tab order to 1, as shown in Figure 16-10.

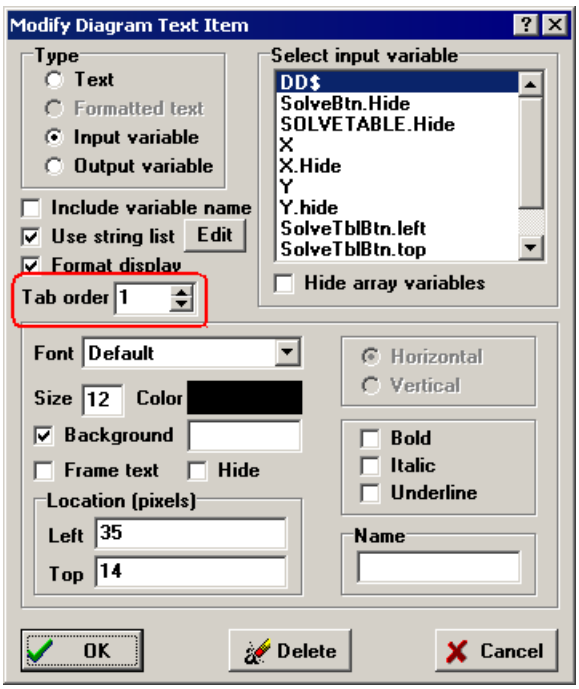
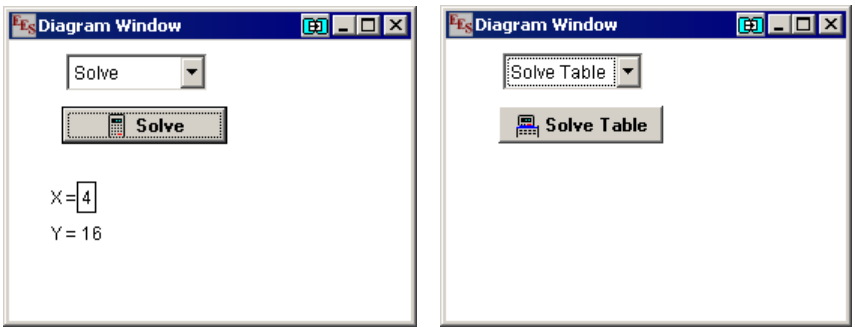


Figure 16-10: Dialog window for the DD$ variable with the Tab order highlighted.

Click OK and hide the Diagram Window tool bar. The Diagram Window that we have created can now be tested. When you select Solve from the drop-down control, it will immediately execute the EES commands to the right of the // characters in Figure 16-9 (which are not visible in the drop-down list). In this case, the Solve Table button will be hidden and the X variable input and Y variable output will be made visible so that you can enter a value for X. The Parametric Table is not used with the Solve command, so it can be hidden using the $HideWindow directive, discussed in Chapter 14. Clicking the Solve button will solve the equations in the Equations Window and display the corresponding value of Y, as shown in Figure 16-11(a).

Selecting Solve Table from the drop-down list will hide the Solve button as well as the text input for X and text output for Y. The Solve Table button will become visible. The left position of the Solve Table button is set to 35 pixels, which puts it exactly at the same position as the Solve button. The $ShowWindow Parametric directive displays the Parametric Table.



(a) **(b)**

Figure 16-11: Diagram Window appearance when the (a) the Solve and b) the Solve Table options are selected from the drop-down menu.

Setting Attributes with a Check Box Object

The attributes of text and graphic objects can be controlled with check box or radio button groups as well as with the drop-down text object. To illustrate, the example developed in the previous section will be revised to use a check box.

Set the Diagram window into development mode by pressing Ctrl-D until the tool bar appears. Next, click on the left edge of the drop-down group to select it and the press the Delete key. Click the Add Check Box button on the tool bar. The Check Box Characteristics dialog will appear. Set the caption to be Use the Parametric Table. The check box control will execute the equations in the upper edit box when its state is changed to unchecked. The equations in the lower edit box will be executed when its state is changed to checked. Enter the equations shown in Figure 16-12, which are the same as those used in the drop-down menu example. Hide the toolbar and test the operation of the check box control.

Setting Attributes with a Radio Button Group

A radio group control operates in a similar manner to the check box control, except that it can provide more than two options. Put the Diagram Window into development mode by pressing Ctrl-D so that the tool bar appears. Select the check box control and press the Delete key. Click the Add Radio Button group button on the toolbar, which will bring up the Radio Button Characteristics dialog. Set the controls so that there are two radio button arranged in two columns. Provide a name for this control, e.g., RadioButtonGroup, so that we can change its characteristics using equations. Edit boxes are provided for the caption of the first button and the equations that will be executed when this button is selected. Enter Solve for the caption of the first button and also enter the equations that should be solved when the Solve button is clicked, which are the same equation as shown in upper edit box in Figure 16-12. Also enter the equation RadioButtonGroup.Color=Yellow# as one of equations. The result is shown in Figure 16-13. Next,

click the second button and enter its information, including the command RadioButtonGroup.Color = Green#.

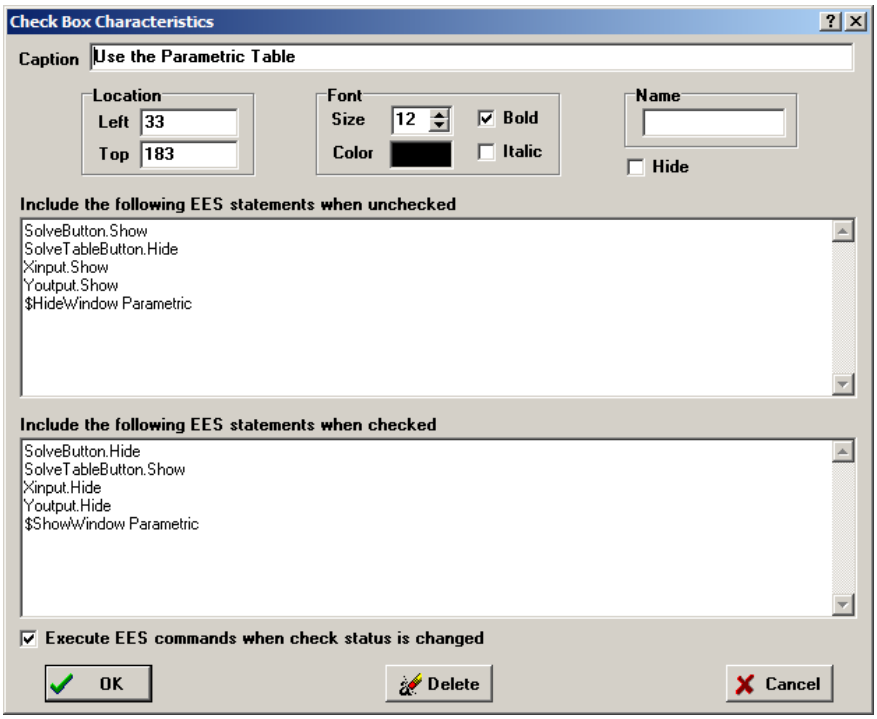


Figure 16-12: Check Box Object Dialog

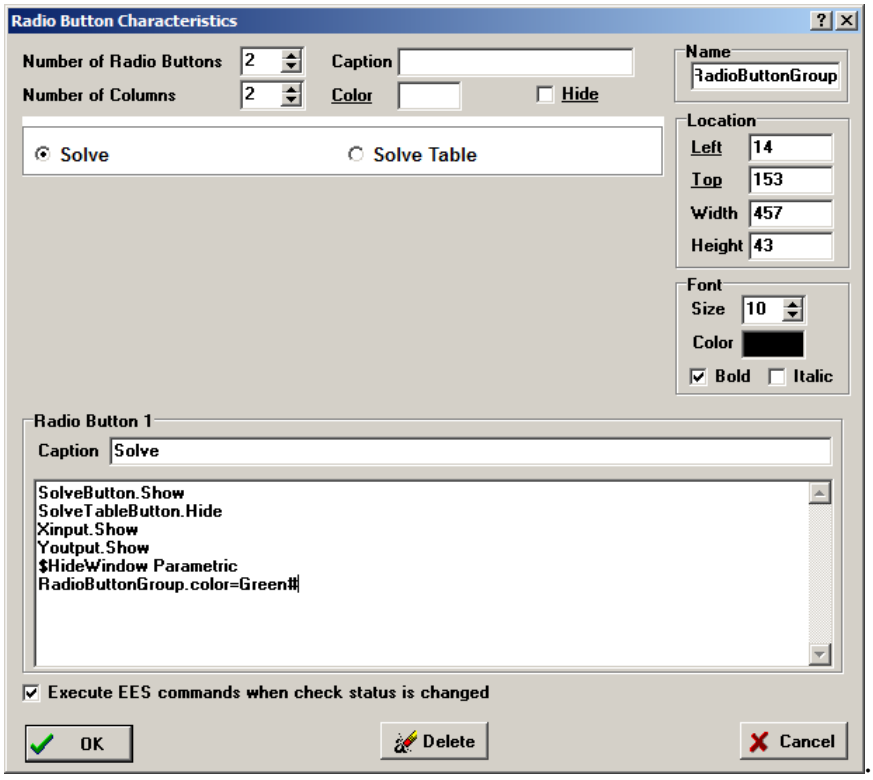
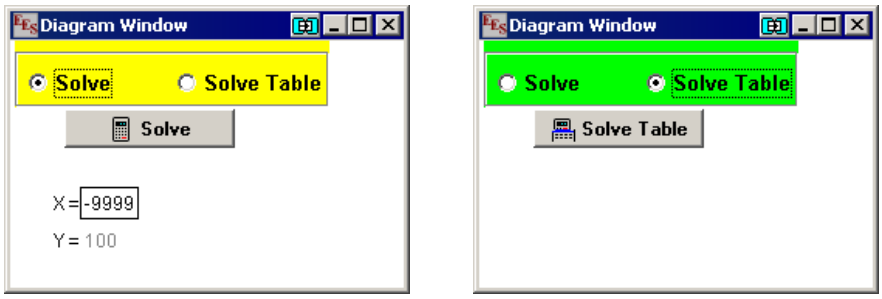


Figure 16-13: Radio Button Characteristics dialog showing information for the Solve button.

Hide the tool bar and test the radio button control. It should operate just as the check box control, except that the radio group will be yellow when the Solve command is selected and green when the Solve Table command is selected, as shown in Figure 16-14.



(a) **(b)**

Figure 16-14: Diagram window appearance for a) the Solve and b) the Solve Table radio button options.

Controlling the Equations Window with Check Boxes and Radio Button Groups

The check boxes and radio button groups can provide much more control over an EES program than is initially evident. On way to apply this control is to have the check box or radio group set a string variable that is used with a $IF directive to include or exclude code in the Equations window. The following very simple example demonstrates this capability.

Start a new instance of EES and enter the short program shown in Figure 16-15. The program will not run because string variable DoThis$ is not set.

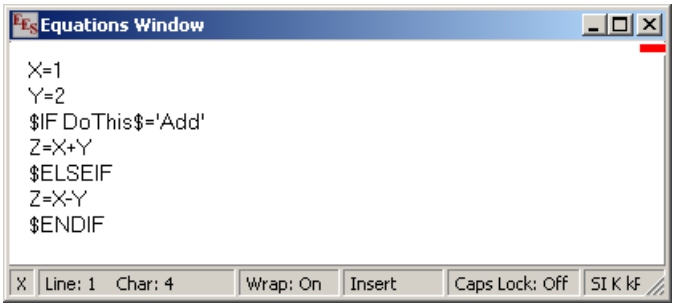


Figure 16-15: Equations Window using the $IF directive to include or exclude equations.

In this example, string variable DoThis$ will be set in a radio button group. Bring the Diagram Window to the front and put it into development mode by entering Ctrl-D if necessary. Select the radio group button from the Diagram Window toolbar and then configure a radio button group with two buttons, Add and Subtract. Only one line of code is needed for each button and that line sets DoThis$ to 'Add' or 'Subtract', as shown in Figure 16-16a. For convenience, also place a Calculate button on the Diagram window, so that the Diagram window appears as seen in Figure 16-16b. The $IF directive is a compile-time instruction. However, the objects in the Diagram window, such as the radio button group, are evaluated before compilation is initiated. Consequently, the setting of the radio button can control what equations are included or excluded, as demonstrated in this example.

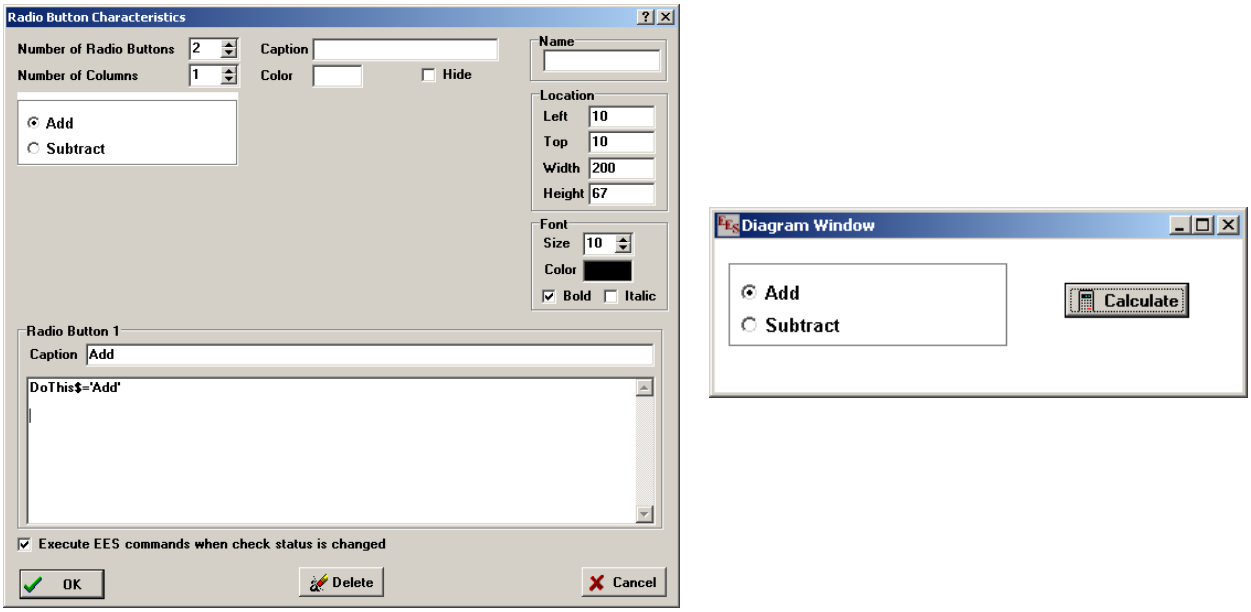


Figure 16-16: a) Radio button group that sets $DoThis$ and b) Diagram window with Radio Group.

16.3 Using Parametric and Integral Tables for Animations

Section 16.1 describes the attributes of Diagram Window objects that can be controlled with EES variables. Section 16.2 shows how these attributes can be changed using drop-down lists, check boxes and radio button groups. This section will demonstrate how the graphic objects in the Diagram Window can be controlled to provide animated graphics. Unlike most other animation programs, the animations in EES are controlled by calculated values. Therefore, the animations can be directly coupled to information from detailed simulations or plots. Animations can only be created by the Professional version, but once created, they can be displayed in any version of EES, including distributable programs (see Chapter 17).

Creating an animation with EES is simple. The first step is to draw the object in the Diagram Window using either the drawing tools provided in EES or an external drawing program such as Power Point, as described in Section 15.1. The object must be given a name in the name field of the object properties dialog so that its attributes become active, as described in Section 16.1. Equations must be entered into the Equations Window to specify the attributes as needed. The Diagram Window display is continuously updated as the attributes change. Therefore, attributes that are changed within a Parametric or Integral Table will cause objects to become animated, as demonstrated with the examples provided in this section. The final step is to add an animation control bar to the Diagram Window. This is done by selecting the Add Calculate button in the Diagram Window tool bar when the Diagram Window is in development mode. The dialog shown in Figure 16-17 will appear.

.

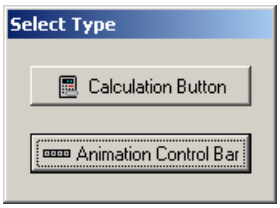


Figure 16-17: Add Calculate Button dialog.

Click the Animation Control Bar button to place an animation control bar on the Diagram Window. The animation control bar can be dragged to the desired location on the window by pressing and holding the left mouse button while the cursor is positioned anywhere on the first four buttons by sliding the mouse to a new location. The functions of the controls on the animation bar are identified in Figure 16-18.

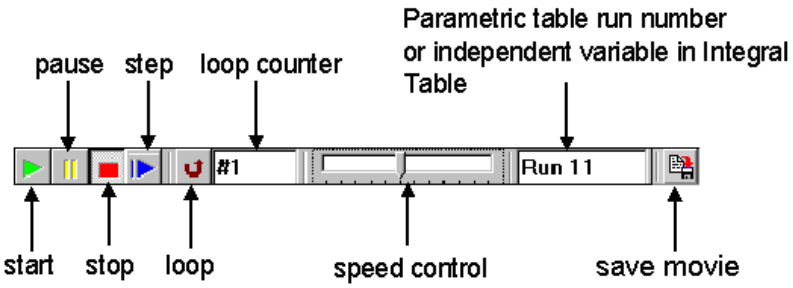


Figure 16-18: Animation Control Bar.

Setting up an Animation Using the Parametric Table

As a first example, we will animate the position of a ball in the Diagram Window. Bring the Diagram Window to the front and place it into development mode. Click the Add ellipse/circle button on the tool bar and create a red ball. Name the object Ball in the Name field, as shown in Figure 16-19.

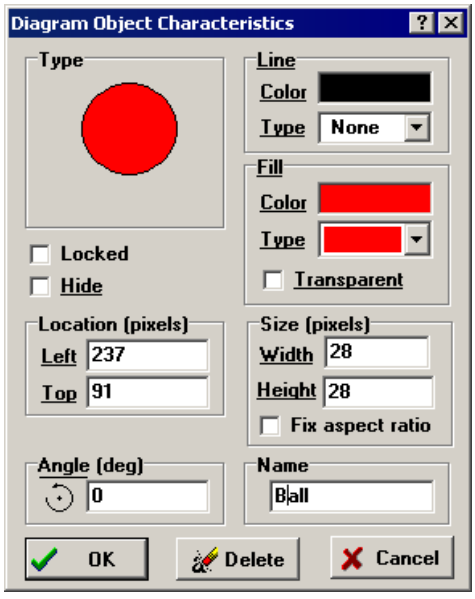


Figure 16-19: Diagram Object Characteristics dialog for an Ellipse/Circle.

In the Equations Window, enter the following equations.

```
$UnitSystem Degree
Ball.Left=X
Ball.Top=150 + 100*sin(X)
```

Create a Parametric Table with 361 rows that has columns for the variables X, Ball.Left, and Ball.Top. Enter values for X that range from 0 and 360. Return to the Diagram Window. Click the Add Calculate button on the tool bar and select the Animation Control Bar from the dialog that is shown in Figure 16-17. Right-click on one of the buttons in the animation control bar to bring up the Animation Characteristics dialog in Figure 16-20. Select the Solve Table button. Since there is only one Parametric Table, the table choice should already be set to Table 1.

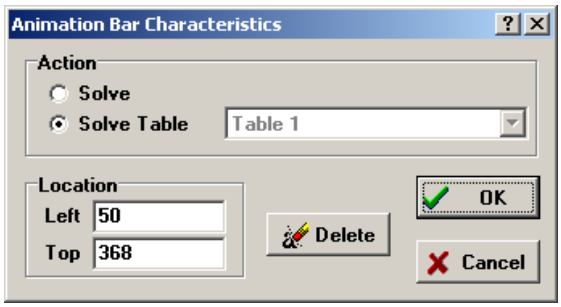


Figure 16-20: Animation Control Bar Characteristics dialog.

Hide the Diagram Window toolbar so that the Diagram Window is in application mode. You may also want to hide the Parametric Table. Click the start button ( ) on the animation control bar and watch the show. As each row of the Parametric table is solved, new values are assigned to the variable Ball.Left and Ball.Top. The position of the object Ball will change as its attributes change. The slider in the animation control bar controls the speed at which animation proceeds. Click the Pause button ( ) to pause the animation. The Stop button ( ) halts the animation. When paused or stopped, clicking the step button ( ) will move the animation forward by one step, which in this case means that the next row in the Parametric Table will be solved and the Diagram Window updated accordingly. Note that the current row in the Parametric Table is displayed as Run# in the animation control bar. If you click the Loop button ( ) then the animation will restart when it reaches the end of the Parametric Table. The number of times that the animation has been restarted is displayed to the right of the Loop button in the animation control bar. All graphic and text items in the Diagram Window can be animated in this same way.

The animation can also be coupled to one or more plots. To demonstrate this capability, create a new plot in which the X-axis is set to the variable Ball.Left and the Y-axis is set to the variable Ball.Top. Right click on the plot and select the Automatic Update checkbox. Start the animation with it set to loop and watch the plot. Notice the red circle that shows the current point in the simulation as the animation progresses, as shown in Figure 16-21.

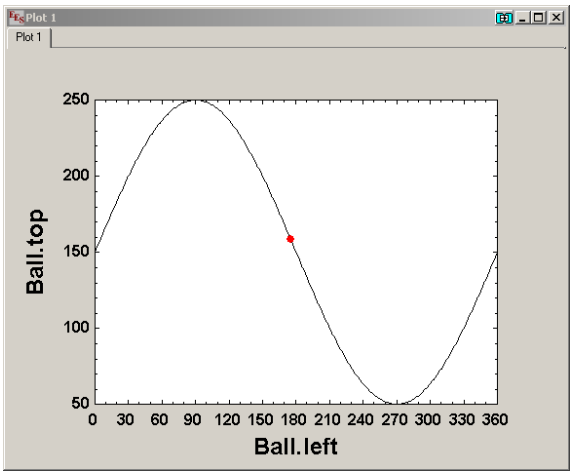


Figure 16-21: Plot window for the animation of the graphic object Ball.

Saving Animation Files for the Generation of a Movie

Animations generated in the Diagram Window can be saved in a sequence of files that can subsequently be assembled into a stand-alone movie. This capability is enabled by clicking the Save Movie button that appears at the right end of the animation control bar shown in Figure 16-18. The Save Diagram Window Animation Files dialog shown in Figure 16-22 will appear.

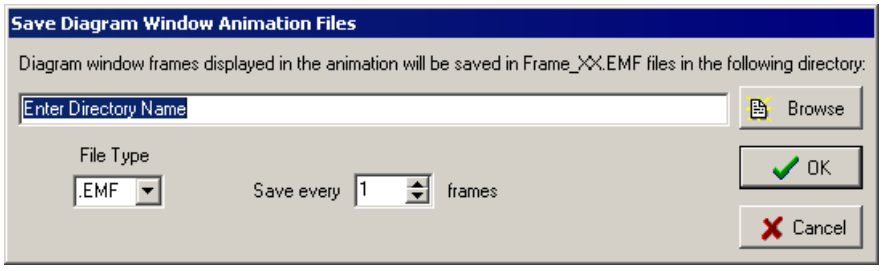


Figure 16-22: Save Diagram Window Animation Files dialog.

Each animation file that is saved will have the file name Frame_XX.yyy where XX is a sequential number that starts at 1 and ends with the last frame saved. The extension yyy corresponds to the file type extension which can be .emf, .jpg, or .bmp, as selected from the File Type drop-down control in Figure 16-22. It is best to place these files in a separate directory. Enter the directory name in the edit box field or use the Browse button to locate an existing directory. If the Save every control is set to 1 then a file will be saved for every frame in the animation. If it is set to 2 then a file will be saved for every second frame, and so on. Click OK to generate the files. Once the files are created in the specified folder, they can be used to create a movie file using Microsoft Movie Maker or other software, as described in Section 9.2.

Setting up an Animation Using the Integral Table

This next example is a little more complicated, but a lot more fun. It will provide an animation of a cannon shooting a cannonball. The position of the cannon ball will be based on Newton's law of motion. The diameter of the cannon ball is D = 0.1 m and it is composed of iron. The cannon is positioned so that the ball is shot at angle θ = 45° relative to the ground. The initial

velocity of the cannon ball is $\tilde{V}_{ini}$ = 100 m/s. The cannon ball experiences standard gravity and air resistance during its flight. The drag coefficient is C_d = 0.6.

Let's solve this problem and then animate the solution. The inputs are entered in EES.

```
$UnitSystem Degree
$TabStops 3.5 in

D=0.1 [m]                    "diameter of cannon ball"
theta = 45 [degree]          "initial angle"
Vel_ini=100 [m/s]            "initial velocity"
C_d=0.6 [-]                  "drag coefficient"
```

The density of iron (ρ_{ball}) is obtained using the rho_ function which accesses the properties of incompressible substances and the density of air (ρ_a) is obtained using the Density function which accesses the properties of compressible substances. The use of property functions in EES is discussed in Chapter 4. The volume and projected area of the cannon ball (V_{ball} and A_{ball}) are obtained from:

$$V_{ball} = \frac{4}{3}\pi\left(\frac{D}{2}\right)^3 \tag{16-1}$$

$$A_{ball} = \pi\frac{D^2}{4} \tag{16-2}$$

The mass of the cannon ball is given by:

$$m_{ball} = V_{ball}\,\rho_{ball} \tag{16-3}$$

```
rho_ball=rho_(Iron,T=300 [K])                  "density of cannon ball"
rho_a=Density(Air,T=300 [K],P=100000 [Pa])     "density of air"
Vol_ball=4*pi*(D/2)^3/3                        "volume of cannon ball"
A_ball=pi*D^2/4                                "projected area of the ball"
m_ball=rho_ball*Vol_ball                       "mass of cannon ball"
```

The initial velocity of the ball in the *x*- and *y*-directions is computed:

$$\tilde{V}_{x,ini} = \tilde{V}_{ini}\cos(\theta) \tag{16-4}$$

$$\tilde{V}_{y,ini} = \tilde{V}_{ini}\sin(\theta) \tag{16-5}$$

```
Vel_x_ini=Vel_ini*cos(theta)     "initial x-velocity"
Vel_y_ini=Vel_ini*sin(theta)     "initial y-velocity"
```

We are going to solve this problem by animating the results of a simulation carried out using the numerical integration capability in EES that is associated with the Integral command, discussed in

Chapter 7. The first step in this process is to set the state variables and integration variables to some arbitrary values and then verify that it is possible to calculate the derivatives of the state variables. For this problem the state variables are the *x*- and *y*-positions of the cannon ball and the velocity of the cannon ball in the *x*- and *y*-directions. The integration variable is time (*t*).

```
"arbitrary value of state and integration variables"
Vel_x=Vel_x_ini
Vel_y=Vel_y_ini
time=0 [s]
```

The derivatives of *x* and *y* position with respect to time are simply the velocity components of the ball.

$$\frac{dx}{dt} = \tilde{V}_x \tag{16-6}$$

$$\frac{dy}{dt} = \tilde{V}_y \tag{16-7}$$

```
dxdt=Vel_x                    "calculate dxdt"
dydt=Vel_y                    "calculate dydt"
```

The derivatives of velocity are computed using Newton's law of motion:

$$m_{ball}\frac{dx}{dt} = -C_d\,\rho_a\,A_{ball}\frac{\tilde{V}_x^2}{2} \tag{16-8}$$

$$m_{ball}\frac{dy}{dt} = -g - C_d\,\rho_a\,A_{ball}\frac{\tilde{V}_y^2}{2} \tag{16-9}$$

```
m_ball*dVel_xdt=-C_d*A_ball*rho_a*Vel_x^2/2      "calculate dVel_xdt"
m_ball*dVel_ydt=-m_ball*g#-C_d*A_ball*rho_a*Vel_y^2/2      "calculate dVel_ydt"
```

With the state equations implemented it is possible to comment out the values of the state variables

```
{"arbitrary value of state and integration variables"
Vel_x=Vel_x_ini
Vel_y=Vel_y_ini
time=0 [s]}
```

and use the Integral command to numerically carry out the integrations:

$$x = \int_0^{t_{sim}} \frac{dx}{dt}\,dt \tag{16-10}$$

$$y = \int_0^{t_{sim}} \frac{dy}{dt} dt \tag{16-11}$$

$$\tilde{V}_x = \tilde{V}_{x,ini} + \int_0^{t_{sim}} \frac{d\tilde{V}_x}{dt} dt \tag{16-12}$$

$$\tilde{V}_y = \tilde{V}_{y,ini} + \int_0^{t_{sim}} \frac{d\tilde{V}_y}{dt} dt \tag{16-13}$$

The result of the integration is stored in an Integral Table using the $IntegralTable directive.

```
t_sim=12 [s]                                          "simulation time"
x=Integral(dxdt,time,0,t_sim)
y=Integral(dydt,time,0,t_sim)
Vel_x=Vel_x_ini+Integral(dVel_xdt,time,0,t_sim)
Vel_y=Vel_y_ini+Integral(dVel_ydt,time,0,t_sim)

$IntegralTable time,Vel_x,Vel_y, x, y
```

A circle and rectangle are drawn and placed on the Diagram Window to form an object that looks like a cannon, as shown in Figure 16-23. These two objects are selected and grouped by right-clicking and selecting the Group option from the pop-up menu. The group is given the name Cannon by double-clicking on the object and entering Cannon in the name field, as shown in Figure 16-24(a) and (b). Input text items are placed on the Diagram Window for the variables theta and Vel_ini, as shown in Figure 16-23, and the equations specifying these variables are commented out:

```
{theta = 45 [degree]                    "initial angle"
Vel_ini=100 [m/s]                       "initial velocity"}
```

The Angle attribute of the object Cannon (Cannon.Angle) is specified according to the value of the variable theta that is set by the user:

```
Cannon.angle=theta
```

The Left and Top attributes of the object Ball (Ball.Left and Ball.Top) are specified according to the values of the variables x and y:

```
Ball.left=113+x
Ball.top=524-y
```

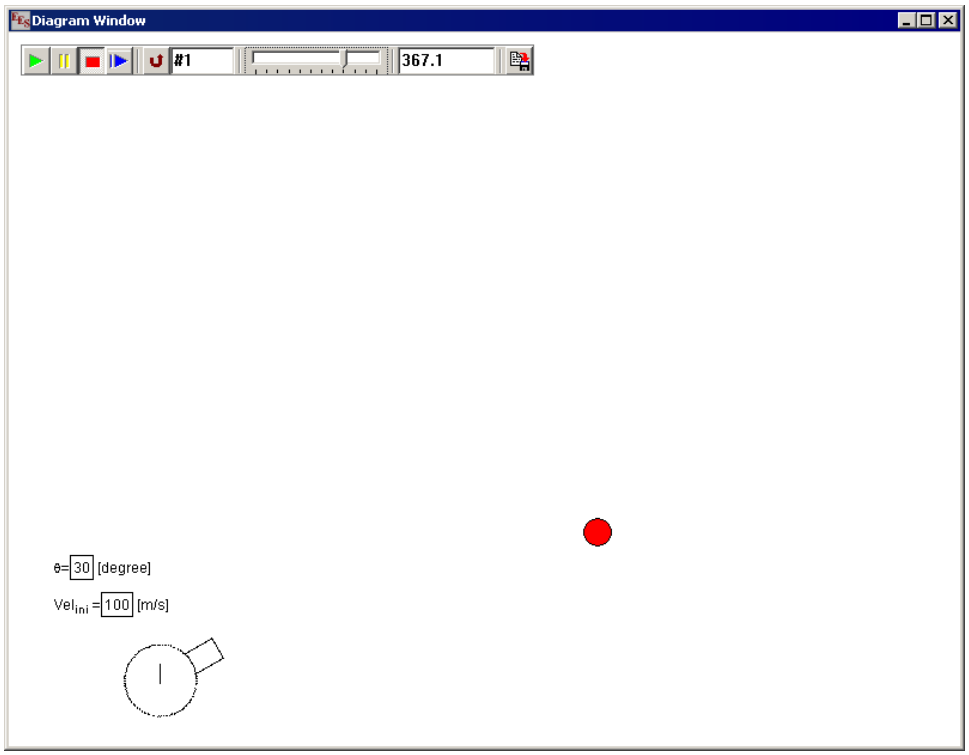
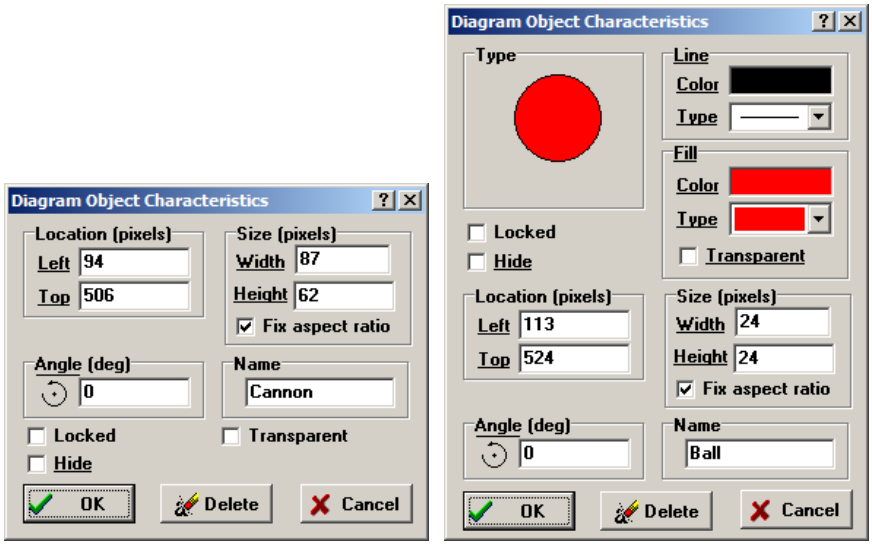


Figure 16-23: Diagram Window for the cannon ball animation.



(a) **(b)**

Figure 16-24: Diagram Object Characteristics dialog for (a) the cannon and (b) the ball.

The final step is to place an animation control bar on the Diagram Window. Click the Add Calculate button and select Animation Control bar from the dialog shown in Figure 16-17. The default action for the animation control bar is to Solve the equations (rather than Solve Table), so no changes are necessary. You can ensure that the control bar is set to Solve by right-clicking on it.

Hide the Diagram tool bar so the Diagram window is in application mode. Enter 30° for the variable theta and 100 m/s for the variable Vel_ini. Click the green arrow in the animation control bar to start the animation and select the loop button so that it continuously replays, shooting cannon ball after cannon ball. Note that the value of the integration variable (which is time in this example) is displayed in the animation control bar.

A little imagination can be applied to improve this program. For example, we could find a picture of a cannon on a website, copy it, and replace the circle/rectangle approximation of a cannon. We could also add a moving target that the cannonball is intended to hit.

There are many interesting animation examples in the Examples folder, which can be accessed from the Examples menu at the right of the menu bar in EES. Select the Animation menu item. You can look at the EES code associated with these examples to see how they work. The information in this chapter should provide you with the background to understand these examples and the ability to create your own animations.

17 DISTRIBUTABLE PROGRAMS

Chapter 15 describes how graphic and text objects can be placed on the Diagram Window. These capabilities allow a graphic user interface to an EES program to be created so that inputs can be entered and used to calculate results without requiring the user to view or understand the equations that are used to carry out the analysis. Chapter 16 describes further capabilities of this graphic user interface including the ability to modify the interface based on user input and display animated output. This chapter describes a very powerful capability provided in the Professional version in which one or more (up to 100) EES programs can be packaged into a Distributable Program. The Distributable Program can be provided to users who do not have EES installed on their computer. A license to use EES is not required. Distributable Programs provide an ideal method for distributing EES programs to a wide variety of users. This chapter will show how these Distributable Programs are made.

17.1 Distributable Program Setup and Startup Dialog

Distributable Programs are special purpose EES programs that have been provided with a graphic user interface implemented using the Diagram Window. These programs are developed for distribution to other users who typically do not know (or care to know) the details of the analysis that are provided by the equations entered into the Equations Window. In fact, the developer may not wish the user to see the equations for proprietary or other reasons. Users will normally enter input information, initiate the calculations and view the results entirely from the Diagram Window and, possibly, Child Diagram Windows. The first step in this process is to develop an EES program that uses the Diagram Window for input and output as discussed in Chapters 15 and 16.

As an example, we will convert the EES program that carries out the throttle analysis developed in Chapter 15 into a Distributable Program. This file is named Throttle.ees and it is provided with the EES program. You will find this file in the UserLib\Examples folder within the folder that EES has been installed. You can navigate to this file more quickly using the Examples menu at the right of the menu bar. Select the Professional Version and the open the file identified as Distributable example – Throttle.

Select Diagram Window from the Windows menu or enter Ctrl-D to bring the Diagram Window to the front in application mode, as shown in Figure 17-1. Note that Save and Load buttons have been added to the Diagram Window to allow the user to save and load inputs. It would be wise to save an input file named Throttle_Defaults.var that can be used to restore the default values for this problem.

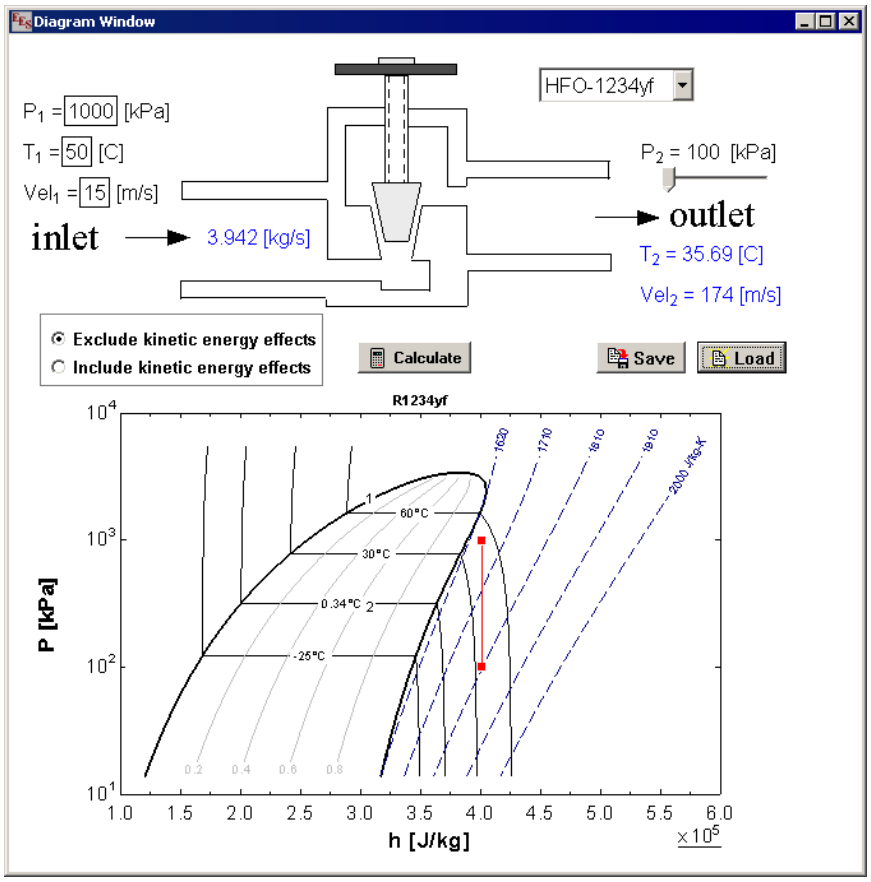


Figure 17-1: Diagram Window in the Throttle.ees program

This EES program has many of the characteristics that are appropriate for a distributable program:

1. The program provides useful information that can be understood by persons other than the developer of the program.
2. All of the inputs can be entered in the Diagram (or Child Diagram) Window.
3. Outputs from the analysis are displayed in the Diagram Window. They may also be displayed in other windows as plots or tables, but it is usually more convenient for the user to see the important results in the Diagram Window.

A limitation of this program is that it is written in EES, which (in theory) would mean that only users who have a license to use EES can benefit from running the program. However, this program can be converted into a special purpose Distributable Program that can operate independent of the EES software. The license agreement in EES requires that Distributable Programs cannot be sold by the Developer; they must be provided to users at no cost.

Once the program has been developed, tested, and provided with a graphical user interface using the Diagram Window, it is ready to be converted into a Distributable Program. To do this, select the Make Distributable Program menu item from the File menu. The message shown in Figure 17-2 will appear the first time that this command is used.

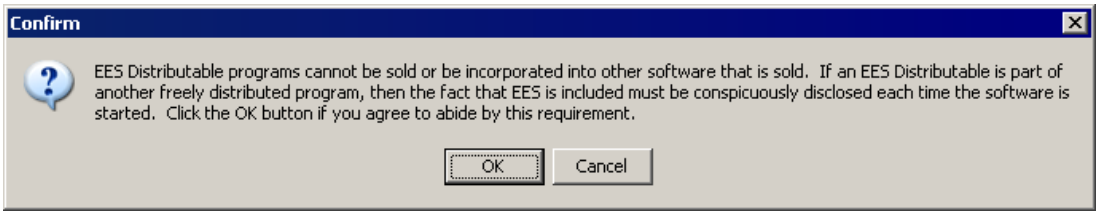


Figure 17-2: Confirmation that Distributable Programs cannot be sold by the developer.

Click the OK button to proceed. The Distributable Program Setup dialog shown in Figure 17-3 will appear. This dialog allows all of the information concerning the Distributable Program to be entered. This information should be entered with some care because it will be not be possible to change the information after the Distributable Program is released.

One consideration in creating a Distributable Program is which version of EES to use. The EES setup program provided two versions, one having 3-D plotting capability. The 3-D plotting routines that are used by EES employ graphic software that may not be compatible with all video cards. Unless your Distributable Program uses 3-D plotting, it would be best to create the Distributable using the version of EES that does not provide 3-D plotting capability.

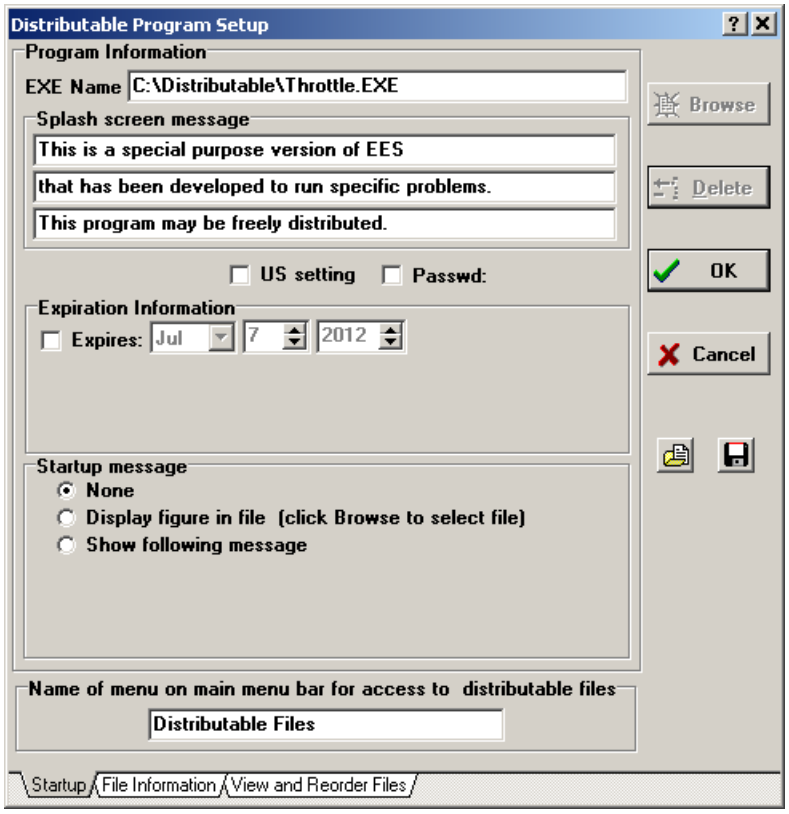


Figure 17-3: Distributable Program Setup dialog.

The distributable program must have a .exe file name extension. The default name that will initially appear in the EXE Name edit field is the complete name (with the directory) of the EES

file that was open when the Make Distributable command was issued, but with the .EES extension changed to .exe. Edit this field so that the distributable program is saved appropriately. It is recommended that you do ***not*** save the distributable file in the same directory as the EES.exe file since it will then load the Userlib folder that is included with EES. Note that if the directory that you enter does not exist then it will be created when you click OK.

A splash screen will appear each time that the Distributable Program is started. The text that appears on the splash screen can be entered on the three lines in the splash screen message box. You can enter whatever you wish here, but you should enter something.

EES can use either a decimal point (U.S. setting) or a comma for the decimal separator character. This choice is made when EES is started and controlled by the Regional setting choices made in the Control Panel of the operating system. However, if the US setting check box is selected in Figure 17-3 then the Distributable Program will use the U.S. setting (i.e., a decimal point separator) regardless of the regional settings associated with the computer that it is run on. This option avoids some compatibility problems that may occur if a Distributable Program is made and run with different regional settings.

EES provides several methods to restrict the use of the distributable file. The first is password protection. If the Passwd: check box is selected then a space for a password will appear and users will be required to provide this password in order to use the Distributable Program. When the user starts the Distributable Program, they will be challenged to enter a password in the dialog shown in Figure 17-4. Note that the password is case sensitive and there is no way to recover the password if you lose it, so be careful.

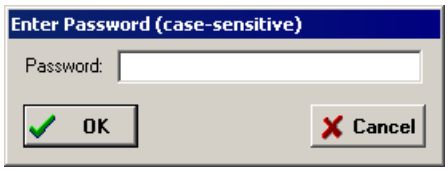


Figure 17-4: Password entry dialog.

Use of the distributable file can also be controlled using an expiration date. If the Expires check box is selected then the date field will become enabled and an edit control will become visible where you can enter the message that will be displayed if the Distributable Program is started after the expiration date, as shown in Figure 17-5. Providing an expiration date for the Distributable Program is recommended, as you may not wish to have the Distributable Program available for use years in the future when newer and improved versions are available.

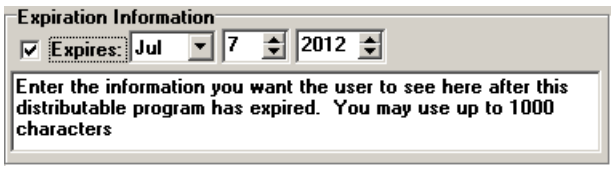


Figure 17-5: Information that can be entered if the Expires check box is selected.

As described above, a message consisting of three short lines of text can be placed in the splash screen that appears when the program is started. A more extensive message that appears after the splash screen is dismissed can also be shown by clicking either the Display figure in file or Show following message button in Figure 17-3. If the Display figure in file is selected then an edit box will appear where you can enter the name of a .bmp, .jpg or .wmf file. The Browse button will be enabled to help locate the file. The figure could provide any graphics and/or information that you wish to display. If the Show following message button is selected then an edit box will appear where you can enter a message with up to 1024 characters, as shown in Figure 17-6.

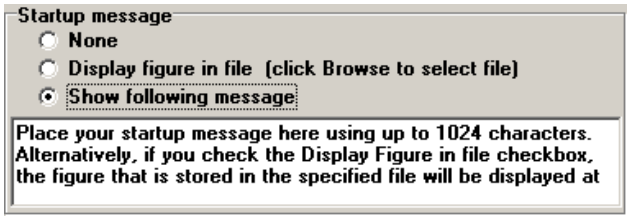


Figure 17-6: Entering a message that appears when the distributable program is started.

The message that is entered can also be accessed by the user while running the Distributable Program by selecting the View startup message menu item in the Help menu, as shown in Figure 17-7.

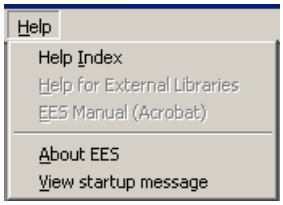


Figure 17-7: Access to the startup message is provided in the Help menu of the Distributable Program.

A Distributable Program file can provide access to one or more (up to 100) different programs. The method for entering the program information is provided in Section 17.2. If more than one program is included then a menu will be provided in the main menu bar when the Distributable Program is running, as shown in Figure 17-8. The menu will list the names of each of the programs that are included in the distributable file. Selecting the program from this menu closes the current program and opens the selected program. The default name of the menu is Distributable Files (as shown in Figure 17-8). However, you can enter a more descriptive name for this menu in the edit field at the bottom of the distributable startup dialog shown in Figure 17-3.

Figure 17-8: Menu provided when the Distributable Program includes more than one file.

Two small buttons are provided just below the Cancel button in the Make Distributable dialog shown in Figure 17-3. These buttons allow saving and loading of scripting files that simplify the process of creating Distributable Programs. The save button ()saves all of the information

contained in the Make Distributable dialog (including the file information that is described Section 17.2) into a file having an .mdi (Make Distributable Information) filename extension. Click this button after you have entered all of the information needed to create the distributable file and before you click the OK button. The load button ( ) will open an .mdi file and use the stored information in order to fill in all of the fields in the Make Distributable dialog. You will likely need several attempts to set up your distributable file so that it looks and behaves exactly as you want. Using these buttons assures that you will not have to repeatedly enter the same information each time.

17.2 Setting Distributable File Information

The next step in creating a Distributable Program is to enter file information for each EES program that is to be included. Click on the File Information tab located at the bottom of the dialog shown in Figure 17-3 in order to access the File Information Tab of the Distributable Program Setup dialog, shown in Figure 17-9.

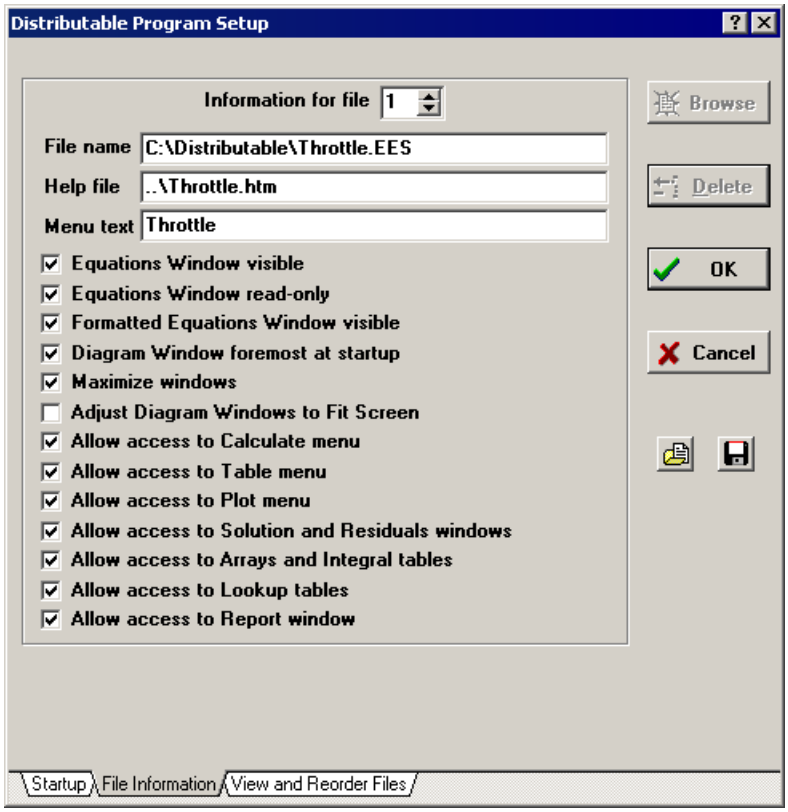


Figure 17-9: File Information tab of the Distributable Program Setup dialog; note that one of these must be filled out for each EES program that is included.

The first time that you display the File Information tab, it will show information for the first EES file that is to be included; by default this is the EES file that was open when the Make Distributable menu command was issued. The first edit box contains the name of the EES file

that will be integrated with the Distributable Program. Use the Browse button to help locate the file if the file name shown in the edit box is not the file that you wish to include.

The next edit box provides an (optional) opportunity to enter a help file for this program. The help file can provide instructions for using the program, background information, credits, or whatever you wish. The help file can be an .htm, .pdf, .txt, or .chm file. Use the Browse button to locate the file. Note that the Browse button will insert the Help file with ..\ in front of the file name. The ..\ refers to the directory that the distributable program is started in. The help file will need to placed in the same directory as the distributable file so that it can be opened as needed. Unlike the EES file, the Help file is not built into the distributable file and therefore it must be included as a separate file and installed in the correct directory with the distributable file.

Perhaps the easiest way to create a help file is to enter the information that you wish to be included into a Microsoft Word document and use the 'Save as web page' menu item from the File menu in order to save tit as a .htm file. This type of file can provide links to other files on the web, if you wish. The help file included for each program will be accessible in the Help menu in the menu bar and labeled using the text provided in the Menu text field in Figure 17-9. The result is shown in Figure 17-10 for the Throttle program. You can also provide a Help button on the Diagram Window that will directly access this file by clicking on the Add Help button ( ) on the Diagram Window tool bar.

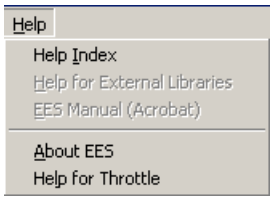


Figure 17-10: Help menu showing access to the help file for the Throttle distributable program.

The remaining file information is provided using check boxes. These selections are, for the most part, self-explanatory. By default, the equations in the EES code are built into the distributable file in a manner such that they can be viewed by the user. However, if you uncheck the Equations Window visible and Formatted Equations window visible check boxes then users will not be able to see your equations. This capability may be important if these equations contain proprietary information. The Equations Window can also be set so that it is read only. However, even if this option is not selected and the user is allowed to change the equations, the modified program cannot be saved as it is an executable file.

A Distributable Program is normally designed around the Diagram Window. Therefore, you will usually want to have the Diagram Window foremost at startup. If the Maximize windows check box is selected then the Diagram Window will fill the screen and you can then select the Adjust Diagram Windows to Fix Screen check box in order to increase or decrease the size of the objects in the Diagram Window so that they fit the adjusted side of the Diagram Window. The remaining check boxes control access to other EES windows.

All of the information in the File Information tab of the Distributable Program Setup dialog refers only to the specific EES program entered in the File name field. Additional EES programs can be included in the distributable file by incrementing the file number at the top of the file information dialog shown in Figure 17-9. The up and down arrows can be used to change the file number. For example, if you change the file number to 2 then the file name field will be blank. You can use the Browse button to select another EES file, e.g., the Ball animation file that was created in Chapter 16, to include in the distributable program. Click the save button () to save all of the file information.

The View and Reorder Files tab in the Distributable Program Setup dialog provides an overview of the programs that will be included in the distributable file. This dialog is only useful when the distributable file will include more than one EES program. All of the programs that will be included are displayed in a list, as shown in Figure 17-11. As shown, the menu names of the program are listed. Click the Show file names button to display the complete EES file name (including directory information) in the list. Click the Show menu names button to display the names of the files as they will appear in the Distributable Files menu that will appear to the right of the Help menu in the EES menu bar. It is possible to change the order of the files in this directory. For example, to make the Ball program the first program in the Distributable file, press and hold the left mouse down while the cursor is on the Ball entry and move the mouse so that the mouse cursor is above the Throttle entry. Up to 100 files can be placed in the distributable file, so this view can be helpful to ensure that the desired files have been selected and properly ordered.

Figure 17-11: View and Reorder Files dialog.

When the distributable program is started, the first program will automatically be loaded by default. However, any one of the files can be selected as the file that appears at startup by

providing /# as a parameter where # is an integer between 1 and 100. For example, entering MyPrg.exe /2 in the Windows Run dialog will open the distributable file named MyPrg.exe and start program 2.

17.3 Completing the Distributable File

At this point, the distributable file information required by the Startup tab has been entered and the attributes of the EES files that are to be included with the distributable file have been specified in the File Information tab. Be sure to save the file setup information by clicking the save button (). Next, click the OK button, which will display the notice shown in Figure 17-12.

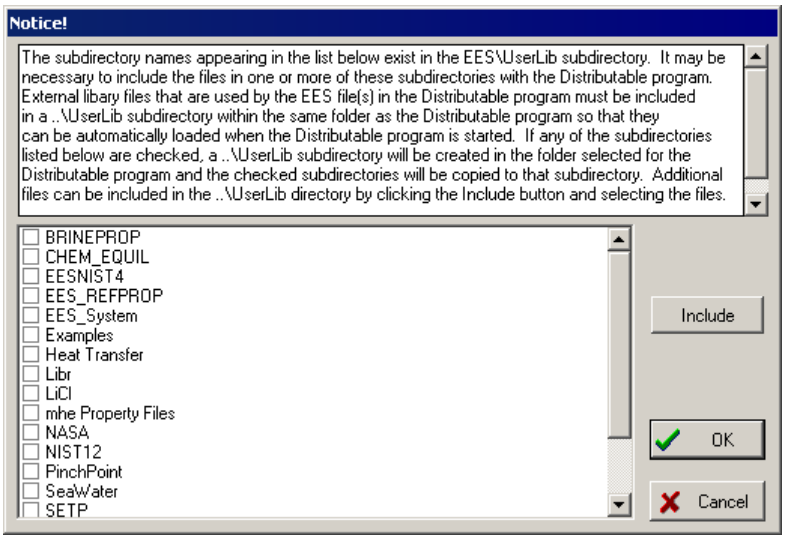


Figure 17-12: Dialog to allow selection to files to include in the ..\Userlib folder for the distributable file.

The Notice! dialog requires some explanation. The .exe file that is created includes most of the files needed to run the program in the distributable file, but it may not include all of the information that is needed. The files that are not automatically built into the .exe are:

1) the optional help files for each EES program;

2) any .var files that are created when you click the Save button on the Diagram window; and

3) any external library files that are used by the EES programs in the distributable file.

As noted in Section 17.2, it is best not to refer to a specific directory in the File Information dialog because a user may wish to install the distributable file in a different directory than you expected. Help files are usually placed in the same directory as the .exe file. The Help file name that begins with ..\ tells the distributable program to look for the file in the same directory as the executable file. Therefore, all of the help files that will be needed by the distributable program must be placed into the directory that the executable file will be saved in.

The only way for users to save input data entered in the Diagram Window is with a Save button placed on the Diagram window. The Save button saves a .var file that includes all current variable information. A Load button allows the user to read a selected .var file. If the distributable file provides a Save button then it is a good idea to provide a default .var file with the program that contains default data in case the user enters incorrect data and cannot run the distributable program. You may wish to make the default .var file read only so that it cannot be accidently modified. Copy all of the .var files that you wish to distribute with the executable into the directory that the .exe file will be saved in.

One or more of the EES files that you have included in the distributable file may require an external library file. These files are normally provided in the Userlib folder of the EES program. For example, perhaps one of your EES programs uses the functions that are included with the Heat Transfer library discussed in Chapter 12. If so, that particular external library must be provided in a Userlib folder that is installed in the same directory as the executable file. The dialog shown in Figure 17-12 lists all of the external library files that are installed with your version of EES. If any of the EES files in the distributable program uses one of these library files, click on the library name so that it appears with a check. After creating the .exe file, EES will create a Userlib folder in the same directory as the .exe file and copy all of the library files that you have checked into the Userlib folder. (This is another very good reason for saving your .exe file in a different directory than the directory that the EES.exe file is located in.) Click the Include button to navigate to and select any additional external files that are not displayed in the list. These additional files will also be placed in the Userlib folder in the directory with the .exe file.

The complete distributable package that should be provided to users must include the .exe file, any help files and .var files that you wish to include and the \Userlib folder containing all external library file that are used by the EES programs in the distributable file. These external files in the Userlib folder will automatically load when the .exe is started.

It is convenient to provide a single file to users. This can be accomplished by using WinZip® or other program to compress all of the files located in the directory that the .exe file is located, including all help files, .var files and all files in the Userlib directory, into one .zip file. You should copy this .zip file to another directory, unzip it and test that it operates as expected. This one file can then be distributed to users. The user can unzip this compressed file to restore the directory with all of the necessary files, ready to use.

18 MACROS

A macro is a set of instructions to EES that are read from an ASCII file or from the EES Macro Window. The macro commands can instruct EES to automatically open a file, solve the equations, create and solve a table, store calculated results in a file, print, plot, etc. In fact all of the capabilities that are provided with the menu commands and directives in EES can also be implemented using macro commands (as well as others that are unique to macros). There are several advantages to using macros. First, macros allow a set of repetitive tasks to be scripted so that they can be re-executed with a single click. They allow two or more EES programs to be run in series. Also, macros provide a way in which EES can interact automatically with other programs. Finally, macros can be used to control EES from another program, either by running EES with the macro file name, as described in Section 18.3, or by using dynamic data exchange (DDE) within the calling program, as described in Section 18.4.

18.1 The EES Macro Window

The easiest way to create an EES macro file is to select the Open or Create Macro command from the File menu. After selecting this command, EES will prompt the user to provide a name for the macro file. The file name extension for a macro file is .emf, which signifies EES Macro File. The .emf filename extension is required. After entering the file name, a small Macro Window will appear at the bottom of the screen. If the macro file name you provided already exists then the contents of the existing file will be displayed in the Macro Window. Otherwise the Macro Window will be blank.

Creating a Macro in the Macro Window

As you select commands from the EES menus, the equivalent macro commands will be automatically entered into the Macro Window. For example, select Open from the File menu and open the BasicEqn.ees file that can be found in the UserLib\Examples directory in your EES folder. When you apply the Open command, EES will add a line to the Macro Window as shown in Figure 18-1.

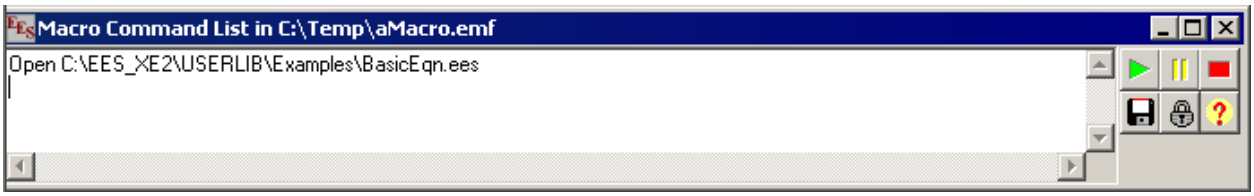


Figure 18-1: Macro Window with one macro command.

Note the control buttons that are located at the right of the EES Macro Window. The top row provides the capability to play ( ), pause ( ), or stop ( ) the execution of the macro. The bottom row of buttons saves the macro file ( ), locks/unlocks the Macro Window ( ), and displays help for macro commands ( ). The Play button also appears in the button ribbon below the main menu bar in the EES program whenever the Macro Window is open. The EES

Macro Window can be resized and moved to other locations in the usual manner. If the Macro Window is hidden by other windows, it can be brought to the front and made the active window by selecting it from the Windows menu.

Most of the EES menu commands will produce a macro instruction that will appear in the Macro Window. If you do not wish to have the commands appear in the Macro Window then click the lock button in the Macro Window toolbar. The text in the locked Macro window will be shown in a gray font.

As a simple example, solve the equations in the BasicEqn.ees file by pressing the shortcut key F2. The macro command Solve will appear in the EES Macro Window, as shown in Figure 18-2.

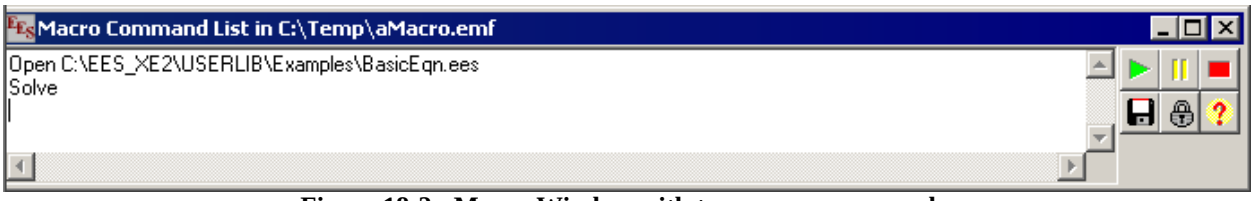


Figure 18-2: Macro Window with two macro commands.

The Macro Window is editable; you can modify the commands that appear in this window or delete them if you wish. A detailed list of all macro commands and their options is provided in Section 18.2. You can comment or uncomment one or more lines in the Macro Window by selecting the lines, clicking the right mouse button and selecting the Comment or Uncomment menu item from the pop-up menu. For example, if the EES file is already open and you do not wish to have the macro command open it again then select the first line in the Macro Window shown in Figure 18-2, right-click and comment it out; the result is shown in Figure 18-3. Note that a commented line is indicated with the characters // placed at the front of the line. You could alternatively just enter // at the start of the line to comment out a macro command.

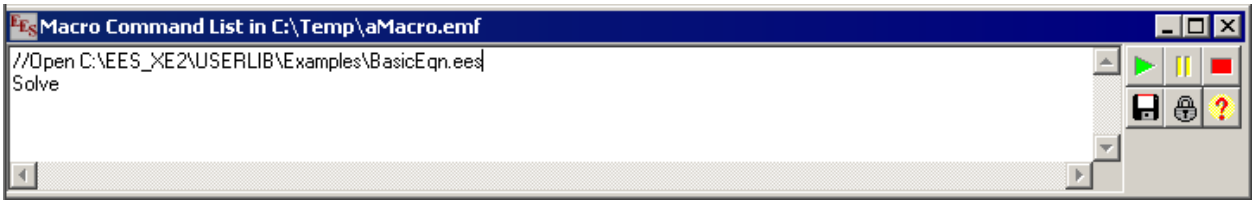


Figure 18-3: Macro Window with the first macro command commented out.

Uncomment the first line and select the Save button at the right side the Macro window in order to confirm the file name to save the macro file. Now we can test this macro. Lock the Macro Window by selecting the lock button. Then select New from the File menu in order to reset EES. Click the play button (the green arrow) that appears in the EES Macro Window tool set or at the right side of the button ribbon. EES will respond by opening the specified file and solving the equations, as instructed in the macro command set.

Repeated Solving Using a Macro

A macro can be used to automatically solve an equation set many times. To illustrate, we will use the simple equation set contained in the BasicEqn.ees example. The equations in this example are:

```
x^2+y^3=77
sqrt(x/(y^2+1))=2
alpha+x=1.234
```

Change the first equation so that the right side is a variable, z:

```
x^2+y^3=z
```

The resulting equation set cannot be solved as there are more unknowns than equations. We can set the value of the variable z to a value within the Macro Window and solve the equations. To do this, open a new Macro Window using the Open or Create Macro command in the File menu. Enter a file name for the macro when prompted to do so. Set the value of the variable z in the Macro Window and then use the macro command Solve to solve the equations:

```
z=77
Solve
```

The result is shown in Figure 18-4. The first line of the macro sets the value of the variable z in the equation set and the second line solves the equations.

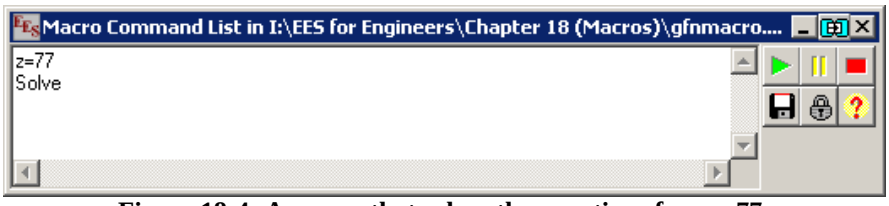


Figure 18-4: A macro that solves the equations for z = 77.

It is also possible to utilize a Repeat..Until construct to create a loop within a macro. The commands below create a macro that contains a loop in which the value of the variable z is incremented by 11 each time it executes. The loop is terminated when the value of variable z is greater than 99.

```
z=77
Repeat
    Solve
    z=z+11
Until (z>99)
```

Figure 18-5 shows the macro commands. Click the Save button in order to save the macro file as aMacro1.emf.

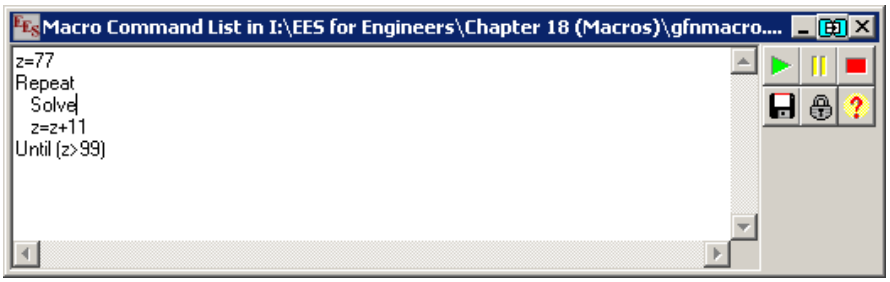


Figure 18-5: A macro that solves the equations three times for different values of z.

Now, when you click the Play button, the equation set will be solved three times for values of z that are equal to 77, 88, and 99. Note that we were able to set (and reset) the value of z in the macro file and its current value in the macro is used in the solution of the equations. Of course, the equations are solved so quickly that you cannot see the results and only the last solution will be displayed in the Solution Window. There are several remedies for this problem. We could pause between each calculation, so that there is time to see the result or we could save the results in a file. The modifications to the macro command that implement both of these alternatives are shown in Figure 18-6.

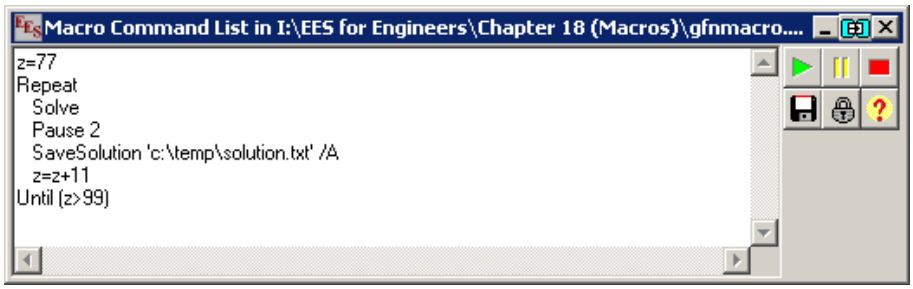


Figure 18-6: Modification to pause and save the solution for each value of z.

The macro command Pause 2 waits 2 seconds before continuing. The SaveSolution macro command saves the Solution Window in the specified file. The /A option appends the results to the existing file. In this case, the result is that the Solution Window for all three runs will appear in the file solution.txt, as shown in Figure 18-7. It would also be possible to save the results in separate files using the string handling capability in the macro commands to compose the file names.

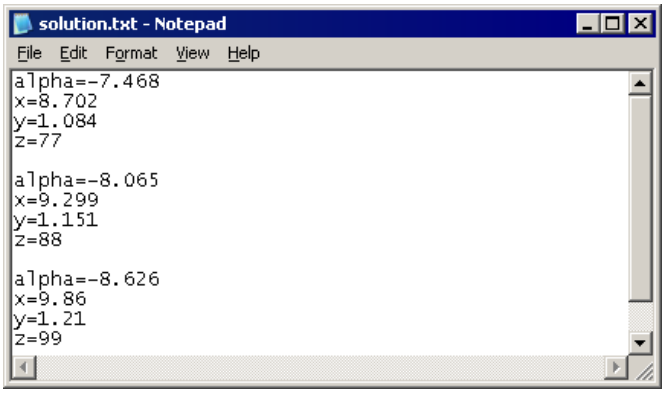


Figure 18-7: Contents of the solution.txt file that contains the solution windows for all values of z.

The EES Log File

EES macros are intended to run 'quietly' with little or no user intervention. This capability is particularly important when macros are run from another program or with Dynamic Data Exchange (DDE), as described in Sections 18.4 and 18.5. If an error occurs during the execution of an EES macro file, you may not see an error message or the error message may not be complete. However, EES writes a log file when it executes a macro. The name of the log file is EESMacro.log and the file is saved in the same directory that the macro file (.emf) is stored. (The name and directory for the Log file can be changed with the LogFileName macro command.) Note that the log file is a text file that can be read with any text editor, such as Notepad. For example, the log file written during the execution of the macro file shown in Figure 18-6 is shown in Figure 18-8. If a macro file does not run as expected, it may be helpful to review the log file to see what instructions were executed and whether any errors were reported.

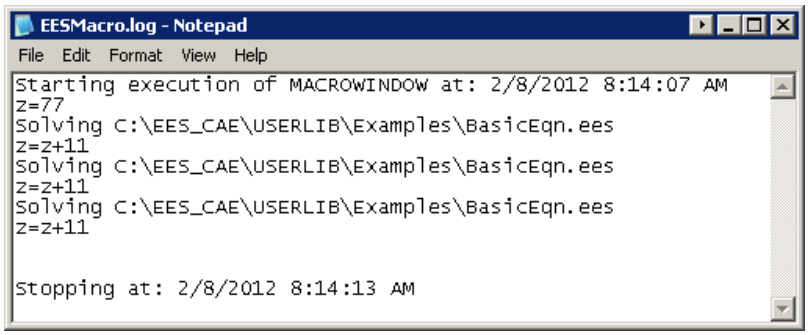


Figure 18-8: EES macro log file written for the execution of the macro in Figure 18-6.

18.2 EES Macro Commands

A complete list of all macro commands, organized by category, is provided in Appendix D. A discussion and example of each macro command (in alphabetical order) are provided in this section.

The most general calling protocol is displayed in bold font for each macro command in this section. Note that parameters that are enclosed in curly brackets are optional. The purpose of each macro command and the calling details are provided below the general protocol. An example is provided after each command. The example macro command(s) can be entered directly in the Macro Window or in a text file with a text editor such as Notepad. Multiple macro commands can be placed on one line, provided that a line separator character (; in the U.S. format, : for the European format) is placed between the commands.

Comments may appear in the macro file. A comment begins with characters // and continues for the remainder of the line. Comments are displayed in blue font. The macro command file must be saved with a file name extension of .emf. Note that macro commands are not extensively checked for syntax. An incorrectly formulated macro command may not execute and halt the execution of the remaining macro commands. The log file provides the best source of information relating to the execution of a macro command set.

AddPlotText PlotName$ Xpos Ypos Text$ {TextSize FontStyle FontColor}

The AddPlotText macro command places the text in the string Text$ on the plot window that is identified by the string PlotName$. The parameters PlotName$ and Text$ can either be string constants (surrounded with single quotes) or string variables that have been previously defined. The parameters Xpos and Ypos specify the lower left corner of the text. Values of Xpos = 0 and Ypos = 0 correspond to the lower left corner of the plot rectangle while values of Xpos = 1 and Ypos = 1 correspond to the upper right corner of the plot rectangle. The parameters TextSize, FontStyle, and FontColor specify the size, style, and color of the font. These parameters are optional; the font will be formatted using the default values if they are not provided. The TextSize parameter can be supplied as a numerical constant or as an EES variable. FontStyle must be one of the keywords plain, bold, or italic. FontColor can be supplied as a string constant or string variable. The colors names are identified in Table 18-1.

Example:

```
AddPlotText 'Plot 1'  0.1  0.1  'This is a label'  16 plain red
```

Table 18-1: Color Names and associated Color Numbers.

Color Number	Color
0	Black
1	Blue
2	Red
3	Green
4	Purple
5	Maroon
6	Yellow
7	Lime
8	Aqua
9	Gray
10	Fuchsia
11	Navy
12	Teal
13	Silver
14	Orange
15	White

AlterValues Table$ Var Rows=RowStart..RowEnd First=FirstValue Last=LastValue {Inc=IncValue} {Mult=MultValue} {Clear}

The AlterValues macro command enters values into a column of an existing Parametric Table named Table$ for the column holding variable Var from rows RowStart through RowEnd. The remaining parameters specify the values that are placed in the table. The statement First=FirstValue sets the value in row RowStart to FirstValue. The statement Last=LastValue sets the value in row RowEnd to LastValue and implicitly sets all values between these rows so that they vary linearly. Alternatively, the statement Inc=IncValue sets the increment used to vary the values to IncValue. The statement Mult=MultValue sets the multiplier to MultValue. Only one of the three choices Last=, Inc=, or Mult= can be used. The keyword Clear removes any values that currently exist in the specified rows in the specified column. Variables can be defined in the macro and used in place of any of the parameters Table$, RowStart, RowEnd, FirstValue, LastValue, IncValue, or MultValue.

Example:

```
Assign TB$='Table 1'
Assign R1=1
Assign R2=10
Assign XN=100
AlterValues TB$  X  Rows=R1..R2  First=0  Last=XN  //set values for column X
AlterValues TB$  Y  Rows=R1..R2  Clear             //clear values for column Y
```

Assign EESVariable=Value

The Assign macro command assigns the value Value to the variable EESVariable. The variable remains assigned to this value until it is reassigned. A variable assigned with a macro command cannot be reassigned in the Equations Window, Parametric Table or Diagram Window. The Assign command can also be used to assign a string variable (identified with a $ as the last character in the name) to a string constant or string variable. The keyword Assign is optional.

Example:

```
Assign X=10
Y=10                    //note that the Assign keyword is optional;
Assign TB$='Table 1'  //Assign can be used for string variables as well.
```

Beep {Type}

The Beep macro of this command will result in an audible sound. The optional Type parameter is an integer between 1 and 5 that can optionally follow the Beep keyword in order to specify the type of sound; the choices are defined in the Sound tab of the Preferences dialog and listed in Table 18-2. If the Type parameter is not provided then the default sound will be played.

Table 18-2: Sound types.

Type	Sound
1	da da da
2	chord
3	notify
4	completed
5	uh oh

Example:

```
Beep 1
```

Check

The Check macro command provides the same function as selecting the Check/Format command from the Calculate menu. The equations in the Equations Window are recompiled and checked. A summary statement is not issued. This command is useful only in rare situations in which it is necessary to have EES review the existing equations before processing subsequent macro commands.

ClearLookupColumn Table$ Column$ {ColumnNumber} RowStart.. RowEnd

The ClearLookupColumn macro command clears the values in a column within Lookup Table Table$. The column that is cleared can be identified by its name, Column$, or by its column number (ColumnNumber). The parameters RowStart and RowEnd are optional and indicate the range of rows to be cleared. The parameters Table$ and Column$ can either be string constants or string variables. The parameters ColumnNumber, RowStart, and RowEnd can either be integers or EES variables with values that are set before the macro command is executed.

Example:

```
ClearLookupColumn 'Lookup 1' 'Column1' 1..7 //clear the first 7 rows in Column1 of Lookup 1
```

Copy {/H} ArraysTable {Table$} {RX CY : RXX CYY}

The Copy ArraysTable macro command copies values from the Arrays Table onto the clipboard. Typically there is only one Arrays Table corresponding to the main equation window which has the name 'Main'. However, if the $ArraysOn directive is placed within a function or procedure then there may be multiple Arrays Tables. The optional parameter Table$ allows the user to specify the name of the Arrays Table to be copied. Tab characters are used to delimit cells on one row. A carriage return line feed character appears at the end of each row. The data in this format can be pasted directly into a spreadsheet. The optional /H parameter will copy the column names and units, in addition to the values in the table.

The range of values to copy can optionally be specified using the nomenclature RX CY : RXX CYY. If these parameters are not provided then the entire table will be copied. The key letters R and C indicate row and column, respectively. The parameters X and Y indicate the number of the row and column, respectively, corresponding to the upper left cell in the range of cells to be copied and the parameters XX and YY indicate the row and column of the lower right cell in the range to be copied. For example, specifying R1 C1 : R10 C4 will copy the Arrays Table values starting in row 1, column 1 through row 10 column 4. It is possible to specify the selection range with string variables. For example, R1$ and C1$ can be defined in the macro commands or within the EES program as 'R1' and 'C1', respectively. String variables R2$ and C2$ can be defined as 'R10' and 'C4', respectively. Note that the strings must include the leading 'R' or 'C' characters and they can be generated using the Concat$ and String$ functions. For example, the statement X=10; R2$ = concat$('R',string$(X)) will set R2$='R10'.

Example:

```
Copy /H ArraysTable //copy the main Arrays Table with headers
```

Copy Diagram

The Copy Diagram macro command copies the Diagram Window to the clipboard in Windows metafile format. Note that only the Main Diagram Window is copied. Child Diagram Windows cannot be copied with this macro command.

Copy {/H} IntegralTable {RX CY : RXX CYY}

The Copy IntegralTable macro command copies the contents of the Integral Table to the clipboard. The optional /H parameter will copy the column names and units, in addition to the values in the table. The optional parameters RX CY : RXX CYY allows the user to specify the range of cells to be copied. If these parameters are not included then the entire Integral Table is copied. The range specification and copy format are explained in the Copy ArraysTable macro command.

Example:

```
Copy /H IntegralTable    //copy the Integral Table with headers
```

Copy EquationWindow {LX CY LXX CYY}

The Copy EquationWindow macro command copies the characters in the Equations Window to the clipboard. The parameters LX CY LXX CYY are optional and can be used to specify the region of the Equations Window to be copied. If these parameters are not included then the entire Equations Window is copied. The parameters L and C indicate line and character, respectively. The parameters X and Y indicate the starting line and starting character, respectively, and the parameters XX and YY indicate ending line and ending character, respectively. For example, specifying L1 C1 : L10 C10 will copy equations starting in line 1, character 1 through line 10 character 10. If the column number exceeds the length of the line then it is truncated to the line length. The key word EquationsWindow can be shortened to Equations.

Example:

```
Copy Equations L1 C1 L3 C999    //copy the first three lines of the Equations Window.
```

Copy {/H} LookupTable Table$ {RX CY : RXX CYY}

The Copy LookupTable macro command copies the contents of the Lookup Table Table$ to the clipboard. The optional /H parameter will copy the column names and units in addition to the values in the table. The optional parameters RX CY : RXX CYY allows the user to specify the range of cells to be copied. If these parameters are not included then the entire Lookup Table is copied. The range specification and copy format are explained in the Copy ArraysTable macro command.

Example:

```
Table$='Lookup 1'
Copy /H LookupTable Table$
//copy the entire contents of Lookup 1 to the clipboard, including the header and units
```

Copy {/H} ParametricTable Table$ {RX CY : RXX CYY}

The Copy ParametricTable macro command copies the contents of the Parametric Table Table$ to the clipboard. The optional /H parameter will copy the column names and units, in addition to the values in the table. The optional parameters RX CY : RXX CYY allows the user to specify the range of cells to be copied. If these parameters are not included then the entire Parametric Table is copied. The range specification and copy format are explained in the Copy ArraysTable macro command.

Example:

```
Table$='Table 1'
Copy /H ParametricTable Table$
//copy the entire contents of Table 1 to the clipboard, including the header and units
```

Copy PlotWindow Plot$ {PlotNumber}

The Copy PlotWindow macro command copies the graphics in the Plot Window named Plot$ to the clipboard in the format specified in the Plot tab of the Preferences dialog. Alternatively, the plot window can be indicated by the integer PlotNumber which specifies the number of the plot window to be copied.

Example:

```
Plot$='Plot 1'
Copy PlotWindow Plot$
```

Copy SolutionWindow

The Copy SolutionWindow macro command copies the entire contents of the Solution Window for the Main Equations Window to the clipboard in text format.

Delete File$

The Delete macro command deletes the file File$. The parameter File$ can either be a string variable or a filename enclosed in single quotes. If the directory information is not provided then EES will look for the specified file in all of the directories that have been opened during the EES session and delete the file in the first directory that it finds that contains a file with the name File$.

DeleteLookup Table$ {TableNumber}

The DeleteLookup macro command deletes the Lookup Table Table$. The parameter Table$ can either be a string variable or the table name enclosed in single quotes. Alternatively, the Lookup Table can be indicated by the integer TableNumber which specifies the number of the Lookup Table tab to be deleted.

Example:

```
Table$='Lookup 1'
DeleteLookup Table$
```

DeletePlot Plot$ {PlotNumber}

The DeletePlot macro command deletes the Plot Window Plot$. The parameter Plot$ can either be a string variable or the plot name enclosed in single quotes. Alternatively, the Plot Window can be indicated by the integer PlotNumber which specifies the number of the Plot Window tab to be deleted.

Example:

```
Plot$='Plot 1'
DeletePlot Plot$
```

DuplicatePlot Plot$ PlotCopy$

The DuplicatePlot macro command will create a copy of the plot Plot$ and name the copy PlotCopy$. The test for the plot tab name to be copied, Plot$, is case insensitive.

Example:

```
Plot$='Plot 1'
DuplicatePlot Plot$ 'Backup Copy'
```

Excel.Cell(C$,Text$)

The Excel.Cell macro command writes values to a specified cell in Excel. The command requires two parameters separated by a list separator (, for the U.S. configuration and ; for the European configuration). The parameter C$ indicates the cell location and should include a letter that designates the column and a number that designates the row. For example C$ = 'B3' corresponds to column B and row 3 in Excel. The parameter Text$ provides the text that will be placed in the specified cell within a previously opened Excel file.

Example:

```
Excel.FileNew
Excel.Show
C$='B3'
Text$='Test'
Excel.Cell(C$,Text$)
```

Excel.FileNew

The Excel.FileNew macro command starts Excel if it is not already open and creates a new, empty worksheet.

Excel.FileOpen(File$)

The Excel.FileOpen macro command starts Excel and opens the Excel file specified by File$. It is necessary to provide the complete file name, including the directory. You can use the EESFileDir$ function to return the directory name of the currently open EES file.

Example:

```
File$=Concat$(EESFileDir$,'Book1.xls')
Excel.FileOpen(File$)
Excel.Show
```

Excel.FileSaveAs(File$)

The Excel.FileSaveAs macro command saves the currently active Excel file. It is necessary to provide the complete filename including the directory information and extension. Note that Excel files can be saved with file extensions .csv, .xls, or .xlsx.

Example:

```
Excel.FileNew
Excel.Cell('B3','Test')
File$=Concat$(EESFileDir$,'Excel_Test.csv')
Excel.FileSaveAs(File$)
Excel.Quit
```

Excel.Hide

The Excel.Hide macro command hides the currently open Excel application.

Excel.Paste

The Excel.Paste macro command pastes the contents of the clipboard into Excel at the selection point defined by the macro command Excel.Range.

Excel.Range(Range)

The Excel.Range macro command sets the selection point within the currently open Excel worksheet. The parameter Range must consist of a column (letter) and row (number) that defines the upper left corner of the range and a column and row that defines the lower right corner of the range, separated by a colon (e.g., A1:C10). The range can also be provided with an EES string variable. The Excel.Paste macro command will paste the contents of the clipboard into Excel at the selection point. To specify the insertion point for a paste operation, set the upper left and lower right corners to be the same cell, corresponding to the desired upper left cell of the paste operation (e.g., 'A1:A1').

Example:

```
Excel.FileNew
Excel.Show
Copy ParametricTable 'Table 1'  //need to have a Parametric Table made
Excel.Range(B1:B1)
Excel.Paste
```

Excel.Quit

The Excel.Quit macro command closes the currently Excel application.

Excel.Sheet(Sheet$)

The Excel.Sheet macro command activates the Excel worksheet with name Sheet$ in the currently open Excel application.

Example:

```
Excel.FileNew
Excel.Show
Excel.Sheet('Sheet2')
Excel.Cell(A1, 'Test')
```

Excel.Show

The Excel.Show macro command makes the currently open Excel worksheet visible.

Export {/A} File$ X, Y, Z, ...

The Export macro command opens the file File$ and writes the values of the EES variables in the list X, Y, Z, ... to the file. If the value of File$ is 'clipboard', then the variables are written to the clipboard. The optional parameter /A appends the data to the file if it already exists. If the file does not exist then a new file with this name is created. Note that if File$ is a text file (.txt) then the format is consistent with a Lookup Table. If File$ is a .csv or .dat file then the data will be saved as a delimited text file. The Export macro has the same options as the $Export directive.

Example:

```
X=1; A$ = 'Test'
Export 'clipboard' X, A$
```

GetDirectory

The GetDirectory macro command returns the current Windows directory, which can be assigned to a string variable. Note that the GetDirectory macro does not provide \ as its last character.

Example:

```
D$=GetDirectory
F$=Concat$(D$,'\MyFile.ees')
Solve
```

Goto Line

The Goto macro command transfers execution to the macro line having label Line. The value of Line must be an integer between 1 and 30000 corresponding to a line label that has been assigned using the Label macro command. Goto and Label macro commands are usually used within If Then Else macro commands.

Example:

```
Solve                                   //the value of X must be set in the Equations Window
If (X<5) Then Goto 10 Else Goto 20
Label 10: Beep 1
Pause 1
Label 20: Beep 2
```

HideWindow WindowName

The HideWindow macro command hides the window specified by the parameter WindowName. The parameter WindowName can be any of the keywords: EES, Equations, Formatted, Solution, Residuals, Arrays, Parametric, Lookup, Integral, Plot, or Diagram. If the keyword is EES, the main EES window and all of the windows contained within it are hidden. Visibility can be restored with the ShowWindow macro.

Example:

```
Solve
HideWindow Equations
```

If (Conditional) Then MacroCommand1 {Else MacroCommand2}

The If-Then-Else macro command executes MacroCommand1 if the conditional statement Conditional evaluates to true. If the optional Else clause is included, then MacroCommand2 is executed if Conditional evaluates to false. The conditional test can refer to variables defined in the macro or variables set in the previous execution of the Equations Window. The conditional test can utilize any of the relational operators: < , <=, =, <>, >, or >=. If the conditional command involves string variables, then only the = and <> relational operators are valid. The entire If-Then-Else macro command must be on a single line. The If-Then-Else macro command is usually used with the Goto command and line labels set using the Label command.

Example:

```
Solve                                   //the value of X must be set in the Equations Window
If (X<5) Then Beep 3 Else Beep 5
```

Import File$ X, Y, Z, ...

The Import macro command opens the file File$ and reads the values of the EES variables in the list X, Y, Z, ... from the file. If the value of File$ is 'clipboard' then the variables are read from the clipboard. This macro command has the same format and options as the $Import directive. Note that the values imported in the macro should not be redefined in the Equations Window.

Example:

```
Import 'clipboard' X, Y          //the clipboard must contain two numbers
Z=X+Y
Solve
```

InsertLookupColumn Table$ After {'ColumnName\Units' ColumnFormat}

The InsertLookupColumn macro command inserts a new column into the Lookup Table Table$ at the location specified by the parameter After. The parameter After must be an integer and it specifies the column after which to insert the new column. If After is 0 then the new column will be the first column in the table. If After is greater than the number of columns in the table then the new column will be the last column in the table. The parameter 'ColumnName\Units' is optional. If provided then the name of the new columns is set to 'ColumnName' and the units is set to 'Units'. Note the backslash should separate the name from the units; if a backslash is not provided then units will not be set. The parameter ColumnFormat is an optional 2 digit field that specifies the format used to display the data in the column. The first digit can be A (auto), F (fixed decimal), E (floating decimal), or S (string). The second parameter is an integer between 0 and 9 that specifies either the number of decimals or the number of significant figures depending on the format.

Example:

```
InsertLookupColumn 'Lookup 1' 999 'Mass\kg' F3
//Creates a new column with name Mass and units kg at the right of the Lookup Table named
//'Lookup 1'.  The format of the new column is fixed decimal with 3 decimal places displayed.
```

InsertLookupRows Table$ After {NRows}

The InsertLookupRows macro command inserts a new row into the Lookup Table Table$ at the location specified by the parameter After. The parameter After must be an integer and it specifies the row after which to insert the new row. If After is 0 then the new row will be the first row in the table. If After is greater than the number of row in the table then the new row will be the last row in the table. The NRows parameter is an optional parameter; if included then multiple rows can be inserted. The number of rows is specified by NRows.

Example:

```
InsertLookupRows 'Lookup 1' 999 5  //inserts 5 rows at the end of the Lookup Table 'Lookup 1'
```

InsertParametricRows Table$ After {NRows}

The InsertLookupRows macro command inserts a new row into the Parametric Table Table$ at the location specified by the parameter After. The parameter After must be an integer and it specifies the row after which to insert the new row. If After is 0 then the new row will be the first row in the table. If After is greater than the number of row in the table then the new row will be the last row in the table. The NRows parameter is an optional parameter; if included then multiple rows can be inserted. The number of rows is specified by NRows.

Example:

```
NRows=5
InsertParametricRows 'Table 1' 0 NRows
//inserts 5 rows at the beginning of the Parametric Table 'Table 1'
```

Label Line

The Label macro command creates a line with the label provided by the parameter Line. The parameter Line must be a number between 1 and 30000. If a colon follows the parameter Line then a macro command can follow the line label on the same line. See GoTo for an example.

Load #N

The Load macro is useful only for macro files that control the execution of a distributable file (discussed in Chapter 17). Distributable files can contain up to 100 separate programs. The Load command clears the memory space and then loads program N where N is an integer between 1 and the number of programs provided with the distributable.

Example:

```
Load #2
```

LoadLibrary File$

The LoadLibrary macro loads the library file File$. Library files must have a file name extension of .lib, .dll, .dlf, .dlp, or .fdl. Library files with the .lib extension are developed in EES, as discussed in Chapter 11. All of the others must be developed in a compiled language such as C or FORTRAN, as discussed in Chapter 19. Note that all library files that are placed in the Userlib directory within the EES folder are automatically loaded.

Example:

```
LoadLibrary 'C:\MyLibrary.lib'
```

Log Message$ (or On/Off)

The Log macro places the string Message$ in the log file EESMacro.log that is created when the macro is played. Macro commands are processed sequentially. Therefore, the Log macro command can be helpful for debugging a macro command file. If the Off keyword appears after the Log command then logging of information to the macro log file will cease. If the keyword On appears after the Log command then logging will be resumed.

Example:

```
T$='Test'
Log T$
```

LogFileName File$

EES writes a log file showing each of the macro commands that it executes. The log file is named, by default, EESMacro.log, and it is placed in the same directory as the macro file. However, you can change the file name to File$ using the LogFileName macro. The file name will typically have a .log extension, but any extension representing a text file is appropriate. The directory information can also be contained in File$; if none is specified then the log file is placed in the same directory as the macro file. Note that File$ cannot be represented with an EES string variable. If a filename is not provided after the LogFileName macro command (i.e., if the file name is blank) then a log file will not be written.

Lookup[Table$, Row, Column] = Value

The Lookup macro places Value into Lookup Table Table$ at position Row and Column. Note that brackets rather than parentheses are required. Also, mathematical operations cannot be used in place of Value (e.g., Lookup['Lookup 1', 1, 1] = i^2 will not work).

Example:

```
R=4
C=2
Lookup['Lookup 1', R, C]=20 //Sets the value of row 4, column 2 in 'Lookup 1' to be 20
```

LookupColInfo Table$ Column Name$ {Units$}

The LookupColInfo macro sets the name of a column in the existing Lookup Table Table$. The number of the column is specified by the parameter Column which must be an integer between 1 and the number of columns in the table. The name of the column is set to Name$. The optional parameter Units$ can be used to set the units of the column.

Example:

```
Table$='Lookup 1'
LookupColInfo Table$ 1 'Mass' 'kg' //Sets the name and units of column 1 to 'Mass' and 'kg'
```

MATLAB.Execute(Command$)

The MATLAB.Execute macro executes the MATLAB command in the string Command$. The command is executed in the MATLAB command window. Note that the MATLAB command is case-sensitive and must be entered exactly as it would be entered in the command window. The result of the command is displayed in the EESMacro.log file. Note that the MATLAB.Open command must be used before the MATLAB.Execute macro is issued.

Example:

```
Command$='X=1'
MATLAB.Open
MATLAB.Execute(Command$)
//Note that the variable X should be set to 1 in the MATLAB command window.
```

MATLAB.Open

The MATLAB.Open starts MATLAB and opens the MATLAB command window.

MATLAB.Quit

The MATLAB.Quit closes the instance of MATLAB that was opened with the MATLAB.Open macro command.

Maximize G a b c {/Method=MethodKeyword}

The Maximize macro command maximizes the dependent variable G with respect to the independent variables in the list a b c. Note that all variables must be defined in the Equations Window and the dependent variables must have non-infinite upper and lower limits; the VarInfo macro command can be used to set these limits. The optimization method can optionally be specified by the /Method=MethodKeyword parameter. The value of MethodKeyword can be Golden or Quadratic for single degree of freedom optimization problems. For problems involving multiple independent variables, the value of MethodKeyword can be Direct, Variable, Genetic, or Nelder. If the /Method=MethodKeyword parameter is not provided then EES will use the optimization method that was used for the last optimization.

Example:

```
Open C:\EES\USERLIB\Examples\MaxPower.ees          //change this based on your directory
Maximize W_dot T_1 /Method=Golden
//maximize W_dot by varying T_w using the Golden Section method
```

MaximizeTable G a b c {/Method=MethodKeyword} { /Table=Table$} {/Rows=RowStart..RowEnd}

The MaximizeTable macro command maximizes the dependent variable G with respect to the independent variables in the list a b c for each of the rows in a Parametric Table. Note that all variables must be contained in the Parametric Table and the dependent variables must have non-infinite upper and lower limits; the VarInfo macro command can be used to set these limits. The optimization method can optionally be specified by the /Method=MethodKeyword parameter. The value of MethodKeyword can be Golden or Quadratic for single degree of freedom optimization problems. For problems involving multiple independent variables, the value of MethodKeyword can be Direct, Variable, Genetic, or Nelder. If the /Method=MethodKeyword parameter is not provided then EES will use the optimization method that was used in the last optimization. The name of the table can optionally be provided with the /Table=Table$ parameter; if the name is not provided then the currently active Parametric Table is used. Finally, the rows in the table that will be run can be specified using the optional parameter /Rows=RowStart..RowEnd.

Example:

Enter the following code in the Equations Window:

```
F=1+b*x-0.5*x^2+b*y-0.5*y^2+0.1*x*y
```

In the Macro Window enter:

```
Table$='Test'
NewTable Table$  Runs=10 b F x y //Create a Parametric Table with 10 runs with b, F, x, & y
AlterValues  Table$  B  Rows=1..10  First=0.5 Last=2.5 //set the values for b
MaximizeTable F x y /Method=Variable /Table=Table$ /Rows=1..5
//for rows 1 to 5 optimize F by varying x and y using the Variable Metric Method
```

MessageDialog([Caption1$ {,Caption2$, Caption3$}], Message$)

The MessageDialog macro command displays the message text Message$ and provides a button with caption Caption1$. Optionally, two or three buttons can be displayed in which case their captions should be provided in parameters Caption2$ and Caption3$. The button captions should be less than 16 characters. The message text can reference a URL; in this case, the URL name is hot so that clicking on it will start your browser and point it at the

specified URL. Note that the backslash characters (\\) must be used in the URL name because forward slash characters (//) are interpreted to be the start of a comment. The MessageDialog macro will provide the number of the button that is selected (1, 2, or 3) if it is assigned to a variable name, as shown in the example.

Example:

```
B=MessageDialog(['OK', 'Cancel'], 'Please visit http:\\fchart.com and then select OK or Cancel')
If (B=1) Then Solve Else Beep
```

Minimize G a b c {/Method=MethodKeyword}

The Minimize macro command operates in the same way as the Maximize macro command except that the objective function is minimized.

MinimizeTable G a b c {/Method=MethodKeyword} {/Table=Table$} {/Rows=RowStart..RowEnd}

The MinimizeTable macro command operates in the same way as the MaximizeTable macro except that the objective function is minimized.

ModifyAxis Axis Name=Plot$ {Min=MinValue Max=MaxValue Linear (or Log) Int=IntValue} {Ticks239=On (or Off)} {Grids=On (or Off)} {ShowScale=On (or Off)} {Size=SizeValue} {Style=StyleKeyword} {Format=FormatValue} {Color=ColorValue}

The ModifyAxis macro command allows changes to be made to the appearance of the axis of a plot; the plot name is indicated by the string Plot$ that appears after the Name= keyword. The specific axis that will be affected is indicated by the Axis parameter which can either be X, Y, X2, or Y2. (X2 and Y2 refer to the upper *x*-axis and right *y*-axis.) Many attributes of the axis can be changed depending on the keywords that are specified, as summarized in Table 18-3. Note that the order of the keywords does not matter and not all of the keywords need to be specified.

Table 18-3: Keywords for the ModifyAxis macro command.

Keyword	Description
Min = MinValue	Sets the minimum value of the axis to the parameter MinValue.
Max = MaxValue	Sets the maximum value of the axis to the parameter MaxValue.
Linear or Log	Specifies whether a linear or logarithmic scale is used.
Int = IntValue	Sets the interval value for a linear scale to IntValue.
Ticks239 = on (or off)	Specifies whether ticks are shown at locations 2, 3, ... 9 on a logarithmic scale.
Grids = on (or off)	Specifies whether grid lines are shown on the plot.
ShowScale = on (or off)	Specifies whether the scale is shown.
Size = SizeValue	Sets the size of the text used in the labels.
Style = StyleKeyword	Sets the style of the text used in the labels. The acceptable values of StyleKeyword are None, Bold, Italic, and BoldItalic.
Format = FormatValue	Sets the display format for the text used in the labels. The first digit of the FormatValue can be A (auto), F (fixed decimal), or E (floating decimal) and the second parameter is an integer between 0 and 9 that specifies either the number of decimals or the number of significant figures, depending on the format.
Color = ColorValue	Sets the color for the text used in the labels. The value of the parameter ColorValue can be any of the first 13 color names in Table 18-1 (e.g., Black, Red, etc.).

Example:

```
ModifyAxis X Name='Plot 1' Min=0 Max=2 Linear Int=0.2 Grids=On Format=E3 Color=Red
```

ModifyPlot Name=Plot$ Width=WidthValue cm (or in) Height=HeightValue cm (or in) {GridColor=ColorValue}

The ModifyPlot macro command changes the size of the plot rectangle for the plot named Plot$. The size is specified by the values of the parameters WidthValue and HeightValue in either cm or in. Optionally, the grid color can also be specified using the GridColor=ColorValue statement where the parameter ColorValue is any of the colors recognized in Table 18-1. The keywords can appear in any order.

Example:

```
ModifyPlot Name='Plot 1' Width = 20 cm Height = 12 cm  GridColor=Blue
```

New

The New macro command clears the work space and brings up an empty Equations Window. Note that the New macro will not display a dialog asking you to save an existing file; rather, the existing file will be closed without saving.

NewContourPlot Table=Table$ (or TableType#) X=XColumn Y=YColumn Z=ZColumn Type=TypeValue {Legend=yes (or no)} {Resolution=Res} {Xmin=XminValue} {Xmax=XmaxValue} {Xint=XintValue} {Ymin=YminValue} {Ymax=YmaxValue} {Yint=YintValue} {Zmin = ZminValue} {Zmax = ZmaxValue} {Zint=ZintValue} {Name=Plot$}

The NewContourPlot macro command creates a new contour plot. The table containing the data used for the plot is provided by the string Table$. If Table$ is the name of both a Parametric Table and a Lookup Table then the data will be plotted from the Parametric Table. (The search order is Parametric Table, Lookup Table, and then Arrays Table.) Alternatively, the table can be identified by specifying an identifier for the type of table (PAR for Parametric, LOOK for Lookup, ARR for Array and INT for Integral) followed by a number that indicates its position; e.g., LOOK1 is the first Lookup Table. The columns in the table that are used for the *x*, *y*, and *z* data are indicated by the XColumn, YColumn, and ZColumn parameters, respectively. The TypeValue parameter indicates the type of contour plot: Isometric, Colorbands, or Gradient. The remaining keywords are optional and summarized in Table 18-4. If these parameters are not specified then default values are used.

Table 18-4: Keywords for the NewContourPlot macro command.

Keyword	Description
Legend = on (or off)	Activates or deactivates the legend for the contour plot.
Resolution = Res	Sets the resolution of the contour plot. The value of Res must be between 0 (lowest resolution) and 100 (highest resolution).
Xmin = XminValue	Specifies the lower limit of the *x*-axis to XminValue.
Xmax = XmaxValue	Specifies the upper limit of the *x*-axis to XmaxValue.
Xint = XintValue	Specifies the intervals of the *x*-axis to XintValue.
Ymin = YminValue	Specifies the lower limit of the *y*-axis to YminValue.
Ymax = YmaxValue	Specifies the upper limit of the *y*-axis to YmaxValue.
Yint = YintValue	Specifies the intervals of the *y*-axis to YintValue.
Zmin = ZminValue	Specifies the lower limit of the contours to ZminValue.
Zmax = ZmaxValue	Specifies the upper limit of contours to ZmaxValue.
Zint = ZintValue	Specifies the intervals of the contours to ZintValue.
Name = Plot$	Specifies the plot name to be Plot$.

Example:

```
Open C:\EES_CAE\USERLIB\Examples\LinearRegression.ees
NewContourPlot Table=LOOK1 X=T_cond Y=T_evap Z=m_dot Type=ColorBands Name='MyPlot'
```

NewLookup Table$ Rows=NRows Cols=NCols

The NewLookup macro command creates a new Lookup Table with name Table$, NRows rows and NCols columns. The column names are assigned to be 'Column1', 'Column2', etc., but they can be changed using the LookupColInfo macro command.

Example:

```
Table$='MyLookupTable'
NewLookup Table$ Rows=10 Cols=2
```

NewPlot Name=Plot$ Table=Table$ (or TableType#) X=XColumn Y=YColumn {Rows=RowStart..RowEnd} {Line=LineType Symbol=SymbolType SymbolSize=SymbolSize Color = ColorType Legend}

The NewPlot macro command creates a new plot with name Plot$ using the data in the table Table$. If multiple tables are named Table$ then the search order is: Parametric Table, Lookup Table, Arrays Table, and Integral Table The table can also be identified by entering the TableType keyword indicating the type of table (PAR for Parametric Table, LOOK for Lookup Table, INT for Integral Table, or ARR for Arrays Table) followed by a number indicating the tab number of the table; (e.g., LOOK2 would indicate the Lookup Table in the second tab). The *x* data are taken from column XColumn and the *y* data from column YColumn. The remainder of the parameters are optional. The parameter Rows=RowStart..RowEnd specifies that the rows between RowStart and RowEnd are used to make the plot. The LineType parameter specifies the line using the code numbers summarized in Table 18-5. The SymbolType parameter specifies the symbols using the code numbers summarized in Table 18-6. The SymbolSize parameter sets the size of the symbol and the ColorType parameter sets the color using color names shown in Table 18-1. Including the Legend keyword causes a legend to be placed on the plot. The keywords can be included in any order.

Table 18-5: The LineType parameter and associated line types.

LineType	Line Type
0	No line
1	Solid line
2	Dotted line
3	Thick line
4	Dashed line
5	Thick dotted line
6	Thick dashed line
7	Dash-dot line
8	Thick dash-dot line

Table 18-6: The SymbolType parameter and associated symbol types.

SymbolType	Symbol Type
0	No symbol
1	Open circle
2	Open rectangle
3	Open triangle (up)
4	Open butterfly (horiz)
5	Open triangle (down)
6	Open butterfly (vert)
7	Open diamond
8	Closed circle
9	Closed rectangle
10	Closed triangle (up)
11	Closed butterfly (horiz)
12	Closed triangle (down)
13	Closed butterfly (vert)
14	Closed diamond
15	Cross

Example:

```
Open C:\EES32\Userlib\Examples\capvst.ees
NewPlot Name='My Plot' Table=PAR1 X=Q_H Y=T_amb Line=1 Symbol=1 Color=Red
```

NewTable Table$ Rows=NRow X Y Z ...

The NewTable macro command creates a new Parametric Table with name Table$. The number of rows in the table is specified by the parameter NRow. The variables included in the table are specified in the variable list X Y Z ... The variables in the list must appear in the Equations Window or Macro Window before the NewTable macro command is executed.

Example:

```
T$='myTable'
X=1 [kg]; y=2 [kg] ;  Z=3 [kg]      //define variables that will appear in the table (if needed).
Solve                               //registers the new variables
N=15
NewTable T$ Row=N  X Y Z
ShowWindow Parametric
```

Open File$

The Open macro command opens the file File$.

Example:

```
T$ = 'C:\EES32\USERLIB\Examples\CapvsT.ees'
Open T$
```

OpenLookup LookupFile$ {/Format FormatFile$}

The OpenLookup macro command opens the Lookup Table LookupFile$. If the character ? is provided in place of a file name then a select file dialog will appear in order to allow the file to be selected. A string variable can be provided after the ?; if provided, the string variable will be set to the name of the Lookup Table that is created. The name is valid for the remainder of the macro file execution. An optional /Format keyword specifies a format file FormatFile$ that contains the format specification that should be used to read the file. The details of how to create a format (.fmt) file can be found in the EES online help by searching for .fmt.

Example:

```
OpenLookup ? TableName$
```

OverlayPlot Name=Plot$ Table=Table$ (or TableType#) X=XColumn Y=YColumn {Rows=RowStart..RowEnd} {Line=LineType} {Symbol=SymbolType} {SymbolSize=SymbolSize} {Color=ColorType} {Right} {Legend}

The OverlayPlot macro command is used in the same manner as the NewPlot macro except that the plot is overlaid onto the existing plot Plot$. By default, the data are plotted using the left *y*-scale. However, the Right keyword indicates that the data should be plotted using the right (secondary) *y*-scale.

Example:

```
Open C:\EES_CAE\Userlib\Examples\capvst.ees
NewPlot Name='My Plot' Table=PAR1 X=Q_H Y=T_amb Line=1 Symbol=1 Color=Red Legend
OverlayPlot Name='My Plot' Table=PAR1 X=Q_H Y=COP Line=2 Color=Blue Legend Right
```

Parametric[Table$, Row, Col (or Col$)]=Value

The Parametric macro command sets the value of the cell in row Row and column Col within the Parametric Table named Table$ to Value. The column can either be specified with an integer (Col) or a string, Col$, containing the name of the column. Note that square brackets rather than parentheses are required.

Example:

```
Open C:\EES32\Userlib\Examples\capvst.ees
Row=2
Col$='T_amb'
Parametric['Table 1',Row,Col$]=555
```

Paste Lookup Table$ Row Col

The Paste Lookup macro command pastes the contents of the clipboard into the Lookup Table Table$ at row Row and column Col. The Row parameter must be the letter R followed by a number specifying the row. The Col parameter must be the letter C followed by a number specifying the column. For example, R2 C3 specifies an insertion point starting at row 2 and column 3.

Example:

```
Table$='MyLookupTable'
NewLookup Table$ Rows=10 Cols=2     //create a new Lookup Table
Show LookupTable Table$             //bring the Lookup Table to the front
//make sure that the clipboard contains one or more numbers
Paste Lookup Table$ R1 C1           //paste the contents of the clipboard into the table
```

Paste Parametric Table$ Row Col

The Paste Parametric macro command pastes the contents of the clipboard into the Parametric Table Table$ starting at row Row and column Col. The Row parameter must be the letter R followed by a number specifying the row. The Col parameter must be the letter C followed by a number specifying the column. For example, R2 C3 specifies an insertion point starting at row 2 and column 3.

Example:

```
//make sure the variables x and y exist in the Equations Window
NewTable 'Table 1'  Runs=10 x y          //create a new Parametric Table
//make sure that the clipboard contains one or more numbers
Paste Parametric 'Table 1' R1 C1         //paste the contents of the clipboard into the table
ShowWindow Parametric                    //bring the Parametric Table to the front
```

Pause Time

The Pause macro command pauses execution for the number of seconds specified by the Time parameter. The Pause macro is useful when an EES macro file is being repeated and EES is periodically reading a file using the Import macro that has been written by another application (e.g., a data acquisition program).

Print Window

The Print macro command prints the window specified by the Window parameter to the default printer. The default printer can be specified using the PrintSetup command. The recognized values of the Window parameter are summarized in Table 18-7. Note that multiple windows can be printed using a single Print macro command.

Table 18-7: Recognized values of the Window parameter for the Print macro.

Window parameter	Window to be printed
ArrN	The array in the Arrays Table with tab number N.
Diag	The Diagram Window.
Equ	The Equations Window.
For	The Formatted Equations Window.
LookN	The Lookup Table with tab number N.
ParN	The Parametric Table with tab number N.
PlotN	The Plot Window with tab number N.

Example:

```
Open C:\EES_CAE\USERLIB\Examples\CapvsT.ees
SolveTable 'Table 1' Rows=1..7
Print Equ Plot1 Plot2
```

PrintSetup Printer=Printer$ {Orientation=Portrait (or Landscape)} {Copies=NCopies}

The PrintSetup macro command sets the default printer to the printer with name Printer$. The Orientation keyword can be used to set the paper orientation to either portrait or landscape and the number of copies is set to the parameter NCopies. The Orientation and Copies keywords are optional.

Example:

```
PrintSetup Printer='Adobe PDF' Orientation=Portrait Copies=1
```

PropPlot Fluid$ Type NProp1 Val1Prop1 Val2Prop1 ... NProp2 Val1Prop2 Val2Prop2 ... {DoQLines}

The PropPlot macro command generates a property plot of the type indicated by the Type parameter for the fluid Fluid$ which can be any fluid in EES other than AirH2O (see the macro command PropPlot AirH2O Psy to generate a psychrometric plot). The recognized values of the Type parameter are summarized in Table 18-8. The NProp1 parameter specifies the number of lines of constant property 1 (see Table 18-8) that will be shown on the plot and the subsequent list (Val1Prop1 Val2Prop1 ...) specifies their values. The NProp2 parameter specifies the number of lines of constant property 2 that will be shown on the plot and the subsequent list (Val1Prop2 Val2Prop2 ...) specifies their values. The optional DoQLines keyword indicates that the plot should contain lines of constant quality within the vapor dome.

Table 18-8: Recognized values of the Type parameter and associated constant property lines.

Type	Type of property plot	Constant property 1	Constant property 2
Ts	temperature-specific entropy	pressure	specific volume
Tv	temperature-specific volume	pressure	specific entropy
Pv	pressure-specific volume	temperature	specific entropy
Ph	pressure-specific enthalpy	temperature	specific entropy
hs	specific enthalpy-specific entropy	pressure	-
Th	temperature-specific enthalpy	pressure	specific entropy

Example:

```
PropPlot 'Steam' Ts 4 10e6 5E+06 2E+06 0.5e6 6 0.007 0.2 1 7 40 200 DoQLines
```

PropPlot AirH2O Psy Nwb Twb1 Twb2 ... Nv v1 v2 ... P=Pval {Mollier}

The PropPlot AirH2O Psy macro command generates a psychrometric plot for the fluid AirH2O. The Nwb parameter specifies the number of lines of constant wet bulb temperature to place on the plot and the list Twb1 Twb2 ... specifies their values. The Nv parameter specifies the number of lines of constant specific volume to place on the plot and the list v1 v2 ... specifies their values. The value of Pval is the pressure to be used to generate the plot. The optional keyword Mollier specifies that the psychrometric chart should be in Mollier format.

Example:

```
PropPlot AirH2O Psy 5 285 290 295 300 305 4 0.825 0.85 0.875 0.9  P=101300.0 Mollier
```

Quit

The Quit macro command closes the EES application. The Quit macro command is not needed (or desired) when the macro is run from the Macro Command Window. However, the Quit macro command is useful when the macro is started from another program, as described in Section 18.3.

RenamePlot Plot$ NewPlot$

The RenamePlot macro command changes the name of the plot Plot$ to NewPlot$ provided that a plot named Plot$ exists. Note that the plot names are case insensitive.

Example:

```
PropPlot AirH2O Psy 5 285 290 295 300 305 4 0.825 0.85 0.875 0.9  P=101300.0 Mollier
RenamePlot 'Psychrometric Plot' 'My Plot'
```

Repeat
Command(s)
Until (Conditional)

The Repeat-Until macro command executes the commands in the macro file between the Repeat and Until keywords (which must be on separate lines) until the conditional test Conditional is true. Note that any legal expression involving existing variables that are defined in the main section of the Equations Window or the Macro Command Window can be used in the expression. If the expression Conditional is never true then the macro commands in the Repeat-Until loop will be repeated indefinitely or until the Escape key or Macro Window stop button is pressed.

Example:

```
//Put the equation Y=X^2 in the Equations Window
X=0
Repeat
    X=X+1
    Solve
    Pause 2
Until (X>5)
```

Reset X, Y ...

The Reset macro command will reset the status of the variables in the list X, Y ... to unassigned. This command is useful if a variable has previously been used in a macro command and is now to be used in a different manner in the same macro command file.

ResetGuesses

The ResetGuesses macro command resets the guess values for the variables in the Equations Window to their default values.

Run Statement

The Run macro command acts as if Statement were entered into the Windows Start menu. For example, to start Excel use the macro command Run Excel. (Note that there are separate macro commands that allow specific interactions with Excel, e.g., Excel.Hide). You can also start Excel and open a specific file. For example, the macro Run C:\temp\Test.xls will start Excel and open the file Test.xls. EES knows that the .xls file name extension is an Excel file, provided that Excel is installed properly. Using the Run command, it is possible to start external programs that read data exported by EES and then have EES import the data that is output by that program.

Example:

```
Run Excel
Run MATLAB
```

Save {File$}

The Save macro command saves the existing file. If the optional parameter File$ is included then the file will be saved with the name File$. Otherwise, the file will be saved with its existing file name.

Example:

```
Save 'C:\temp\temp.ees'
```

SaveArrays Array$ File$ {/A} {/T} {/N} {/E}

The SaveArrays macro command saves the contents of the Arrays Table having the name Array$ (Array$ = 'Main' refers to the variables in the Main Equations Window) to the file File$. If the character ? is used in place of File$ then a dialog will appear asking the user to enter a file name. The optional /A flag appends the information in the table to an existing file with name File$ if one exists. The optional /T flag causes the data to be transposed before it is written. The optional /N flag causes the column name and units to be written. The optional /E flag causes the data to be written in a format that can subsequently be read as an EES Lookup Table. Note that the /E flag is not compatible with the /T or /N flags.

Example:

```
//Put the equation A[1..5]=[1,2,3,4,5] in the Equations Window
Solve
File$='C:\Temp\Arrays.txt'
SaveArrays 'Main' File$
```

SaveIntegral File$ {/A} {/T} {/N} {/E}

The SaveIntegral macro command saves the contents of the Integral Table to the file File$. If the character ? is used in place of File$ then a dialog will appear asking the user to enter a file name. The optional /A flag appends the information in the table to an existing file with name File$ if one exists. The optional /T flag causes the data to be transposed before it is written. The optional /N flag causes the column name and units to be written. The optional /E flag causes the data to be written in a format that can subsequently be read as an EES Lookup Table. Note that the /E flag is not compatible with the /T or /N flags.

Example:

```
//Put the equations I=Integral(x^2,x,0,1) and $IntegralTable x,I in the Equations Window
Solve
File$='C:\Temp\Integral.txt'
SaveIntegral File$
```

SaveLookup Lookup$ File$ {/A} {/T} {/N} {/E}

The SaveLookup macro command saves the contents of the Lookup Table having the name Lookup$ to the file File$. If the character ? is used in place of File$ then a dialog will appear asking the user to enter a file name. The optional /A flag appends the information in the table to an existing file with name File$ if one exists. The optional /T flag causes the data to be transposed before it is written. The optional /N flag causes the column name and units to be written. The optional /E flag causes the data to be written in a format that can subsequently be read as an EES Lookup Table. Note that the /E flag is not compatible with the /T or /N flags.

Example:

```
Lookup$='My Lookup'
NewLookup Lookup$  Rows=5  Cols=1
i=0
Repeat
    i=i+1
    j=i*i
    Lookup[Lookup$,i,1]=j
Until(i=5)
File$='C:\Temp\Lookup.txt'
```

```
SaveLookup Lookup$ File$
```

SavePlot Plot$ (or Plot#) File$

The SavePlot macro command saves the plot Plot$ to the file File$. Alternatively, the plot can be identified by providing the integer Plot# that indicates the number of the plot tab (use 0 to indicate the foremost plot). The extension of File$ may be .jpg, .bmp, .tif, or .emf and EES will save the plot in the designated format.

Example:

```
SavePlot 'Plot 1' 'C:\temp\temp.jpg'
```

SaveSolution File$ {/A}

The SaveSolution macro command saves the contents of the main Solution Window to the file File$ which must have a .txt extension. The optional /A flag will append the solution to the bottom of the existing file.

Example:

```
File$ = 'C:\temp\solution.txt'
SaveSolution File$
```

SaveTable Table$ File$ {/A} {/T} {/N} {/E}

The SaveTable macro command saves the contents of the Parametric Table having the name Table$ to the file File$. If the character ? is used in place of File$ then a dialog will appear asking the user to enter a file name. The optional /A flag appends the information in the table to an existing file with name File$ if one exists. The optional /T flag causes the data to be transposed before it is written. The optional /N flag causes the column name and units to be written. The optional /E flag causes the data to be written in a format that can subsequently be read as an EES Lookup Table. Note that the /E flag is not compatible with the /T or /N flags.

Example:

```
File$ = 'C:\temp\table.txt'
SaveTable 'Table 1' File$
```

SetDirectory(Directory$)

The SetDirectory macro command sets the current Windows directory to Directory$. Note that the EESFileDir$ and EESProgramDir$ functions may be useful as they return the current directories for the EES file and EES program file, respectively.

Example:

```
Directory$=EESFileDir$          //Get the directory for the current file
SetDirectory(Directory$)        //set the directory to be that of the current file
```

ShowWindow Arrays

The ShowWindow Arrays macro command shows the Arrays Window and places it in front of all other windows. This command is ignored if an Arrays Table does not exist.

ShowWindow Diagram {Name$}

The ShowWindow Diagram macro command shows the Diagram Window and places it in front of all other windows. The name of a Child Diagram Window can optionally be provided as the parameter Name$; if this parameter is not provided then the Main Diagram Window will be shown.

ShowWindow EES

The ShowWindow EES macro command will change the visibility of the EES application from a minimized state to its normal viewing state.

ShowWindow Equations

The ShowWindow Equations macro command shows the Equations Window and places it in front of all other windows.

ShowWindow Format

The ShowWindow Format macro command shows the Formatted Equations Window and places it in front of all other windows.

ShowWindow Integrals

The ShowWindow Integrals macro command shows the Integral Table and places it in front of all other windows. This command is ignored if an Integral Table does not exist.

ShowWindow Lookup {Table$}

The ShowWindow Lookup macro command shows the Lookup Table Window and places it in front of all other windows. The name of a Lookup Table can optionally be provided as the parameter Table$ in which case the table named Table$ will be selected.

ShowWindow Parametric {Table$}

The ShowWindow Parametric macro command shows the Parametric Table Window and places it in front of all other windows. The name of a Parametric Table can optionally be provided as the parameter Table$ in which case the table named Table$ will be selected.

ShowWindow Plot {Plot$ (or Plot#)}

The ShowWindow Plot macro command shows the Plot Window and places it in front of all other windows. The name of a plot can optionally be provided as the parameter Plot$ in which case the plot named Plot$ will be selected. The plot can also be indicated by an integer in the parameter Plot# that indicates the position of the plot tab.

ShowWindow Progress

Normally, a series of EES commands that are initiated from a macro will not result in the Progress Window that otherwise appears during EES calculations. The ShowWindow Progress macro command causes the Progress Window to appear. This may be desirable to reassure the user that the calculations are proceeding normally for a long calculation.

ShowWindow Residuals

The ShowWindow Residuals macro command shows the Residuals Window and places it in front of all other windows.

ShowWindow Solution {Solution$}

The ShowWindow Solution macro command shows the Solutions Window and places it in front of all other windows. The name of a tab in the Solutions window can optionally be provided as the parameter Solution$ in which case the window having this name on its tab will be shown. Setting Solution$ to 'Key' will display the Key Variables Solution Window. Setting Solution$ to 'Main' will display the Main Solution Window. Setting Solution$ to 'Uncertainty Results' will display the results of an uncertainty propagation analysis.

Solve

The Solve macro command initiates the solution process in the same manner as if the Solve command were selected from the Calculate menu.

SolveTable Table$ {Rows = RowStart ... RowEnd}

The SolveTable macro command initiates the solution process for Parametric Table Table$ in the same manner as if the Solve Table command were selected from the Calculate menu. Optionally, the range of rows to be solved can be provided using the Rows = keyword with RowStart and RowEnd values indicating the first and last rows, respectively.

Example:

```
RowStart=1
RowEnd=7
SolveTable 'Table 1' Rows=RowStart..RowEnd
```

Stop

The Stop macro command halts the execution of the macro.

StopCrit {It=ItValue} {Time=TimeValue} {Res=ResValue} {Var=VarValue}

The StopCrit macro command sets the stop criteria properties used by EES in the solution process. These are the same properties that can be set within the Stop Crit tab in the Preference dialog or by using the $StopCriteria directive. The keywords It, Time, Res, and Var must each be followed by an equal sign and then a numerical value. These keywords may be in any order and do not all need to be specified. The value of ItValue is the maximum number of iterations. The value of TimeValue is the maximum value of elapsed time. The value of ResValue is the criteria on the relative residual. The value of VarValue is the stop criterion for the change in variables.

Example:

```
StopCrit It=200          //sets the maximum number of iterations to 200
```

Uncertainty X1 X2 ... Y1=UY1 Y2=UY2 ...

The Uncertainty macro command initiates the uncertainty propagation calculations required to determine the uncertainty in the calculated variables in the list X1 X2 ... resulting from uncertainties specified for several measured variables. The measured variables and their uncertainty are specified in the list Y1=UY1 Y2=UY2 .... The parameters Y1 Y2 ... must be EES variables and the parameters UY1 UY2 ... are the associated uncertainties. The uncertainties must either A (for absolute) or R (for relative) followed by the specified uncertainty value. For example A1 indicates that the uncertainty is 1 on an absolute basis (note that the units of the uncertainty are identical to the units of the variable. An uncertainty value of R0.01 indicates that the uncertainty is 1% of the value of the variable.

Example:

```
EESDirectory$=EESProgramDir$          //get the directory that EES is installed in
//set the file name within the EES directory
File$=Concat$(EESDirectory$,'\USERLIB\EXAMPLES\uncertainty.ees')
Open File$                            //open the file

//calculate the uncertainty in h and Q based on the specified uncertainties for
//T_infinity (1 C), T_s (1 C), and Vel (10%)
Uncertainty h Q T_infinity=A1 T_s=A1 Vel=R0.1
ShowWindow Solution 'Uncertainty Results'       //bring the uncertainty results to the front
```

Units SI Mass (or Mole) Deg (or Rad) kPa (or Pa, bar, MPa) J (or kJ) C (or K)
Units Eng Mass (or Mole) Deg (or Rad) psia (or atm) F (or R)

The Units macro command sets the unit system in EES. The macro command works in the same manner that the $UnitSystem directive works, as discussed in Section 14.3. All of the specifications that could be set in the Unit System tab of the Preferences dialog can also be set using the Units macro command. Note that it is not possible to specify a mixed unit system. For example, if SI is used then specifying F for the temperature units will result in temperature units of C, not F.

Example:

```
Units SI Mass J K Pa Rad
```

UpdateGuesses

The UpdateGuesses macro command sets all variable guess values to their last set of calculated values.

VarInfo Variable {Lower=LowerValue} {Upper=UpperValue} {Guess=GuessValue}

The VarInfo macro command sets the bounds and guess value for the variable Variable in the same way as if these characteristics were specified within the Variable Information Window. The lower and upper bounds are set to the values LowerValue and UpperValue, respectively. The guess value is set to GuessValue. Not all of these parameters must be specified. The numerical values LowerValue, UpperValue, and GuessValue can be variables that have been set in either the Macro Window or the Equations Window.

Example:

```
GuessValue=-1
VarInfo x Upper=0 Guess=GuessValue
```

Word.FileNew

The Word.FileNew macro command starts Microsoft Word and creates a new empty document.

Word.FileOpen(File$)

The Word.FileOpen macro command starts Microsoft Word and opens the file File$. It is necessary to provide the complete file name with directory information. The file name should have an extension of either .doc or .docx.

Example:

```
File$='C:\temp\temp.doc'
Word.FileOpen(File$)
```

Word.FileSaveAs(File$)

The Word.FileSaveAs macro command saves the current Microsoft Word document with the file name File$. It is necessary to provide the complete file name with directory information. The file name should have an extension of either .doc or .docx.

Example:

```
File$='C:\temp\temp.doc'
Word.FileOpen(File$)
Word.FileSaveAs('C:\temp\temp2.doc')
```

Word.Hide

The Word.Hide macro command hides the current Microsoft Word document.

Word.Insert(Text$)

The Word.Insert macro command inserts the text Text$ into the current Microsoft Word document at the current position.

Example:

```
File$='C:\temp\temp.doc'
Word.FileOpen(File$)
Word.Insert('Hello')
```

Word.Paste

The Word.Paste macro command pastes the current contents of the clipboard into the current Microsoft Word document.

Word.PasteSpecial(FormatType)

The Word.PasteSpecial macro command pastes the current contents of the clipboard into the current Microsoft Word document using the format specified by the parameter FormatType. The parameter FormatType can be either Text, Picture, Bitmap, Device Independent Bitmap, or Enhanced Metafile.

Example:

```
Open C:\EES32\USERLIB\Examples\CapvsT.ees
Word.FileNew
Copy PlotWindow 'P-h: R134a'
Word.Show
Word.PasteSpecial(Enhanced Metafile)
```

Word.Quit

The Word.Quit macro command closes the current Microsoft Word document.

Word.Show

The Word.Show macro command makes the current Microsoft Word document visible.

18.3 Interacting with External Programs using Macros

This section discusses methods for using macros to allow EES to interact with other programs such as Excel.

Interacting with Microsoft Excel

The macro command set in EES is rich and it provides the capability to apply nearly every feature in EES. One set of commands allows EES to interact with other programs, such as Microsoft Excel. This section describes a very simple example of this capability.

Start EES and enter the following equations into the Equations Window:

```
$UnitSystem SI C kPa mass
R$='R134a'
h=enthalpy(R$,T=T,P=P)
s=entropy(R$,T=T,P=P)
v=volume(R$,T=T,P=P)
```

These equations cannot be solved unless values are provided for the variables T (in °C) and P (in kPa). In this example, values of T and P will be obtained from Excel and the calculated values of specific enthalpy, specific entropy and specific volume will be exported back to Excel.

Create a Parametric Table using the New Parametric Table command in the Tables menu. Specify 10 rows (the default) and provide columns for variables T, P, h, s, and v (in that order) as shown in Figure 18-9.

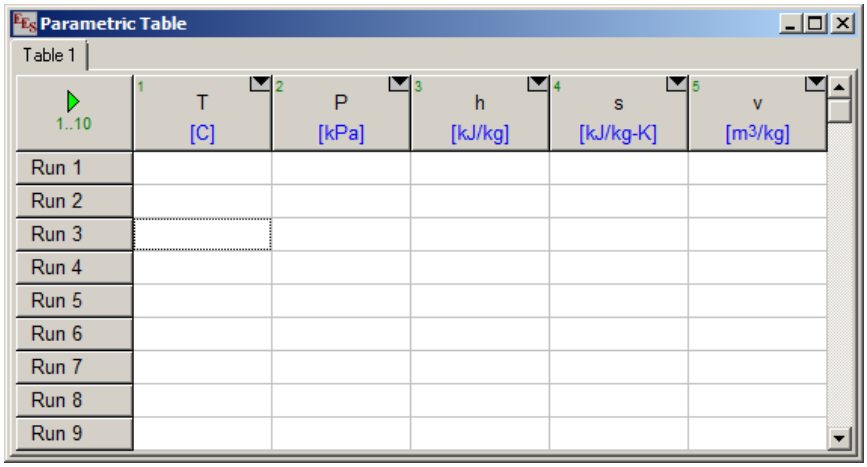


Figure 18-9: Parametric Table.

Select the Variable Information command in the Options menu and enter the units for all variables, as shown in Figure 18-10:

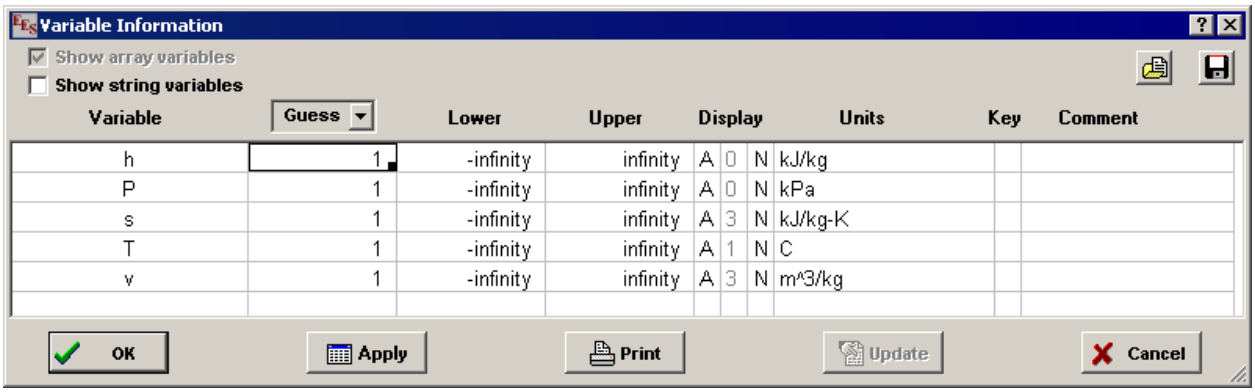


Figure 18-10: Setting units of variables in the Variable Information dialog.

Save the EES file as C:\Temp\TestExcelMacro.ees. Next, open Microsoft Excel and enter the information shown in Figure 18-11 into Sheet 1. Note that the Excel sheet provides values of temperature and pressure that will be copied into the Parametric Table in EES. It also has empty columns for specific enthalpy, specific entropy and specific volume. Save the Excel file. In this example, the file name is C:\Temp\Excel_ws1.xlsx. You can of course use any directory or file name, but it is necessary to use the complete file name in the EES Macro file that we will develop next. Close the Excel file.

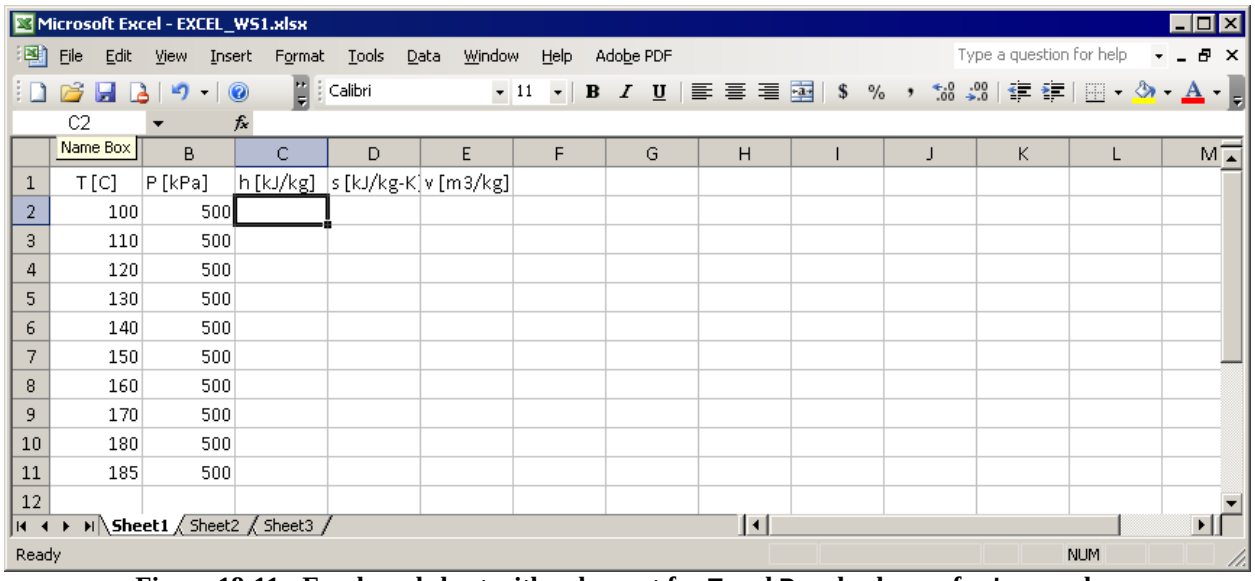


Figure 18-11: Excel worksheet with values set for T and P and columns for h, s, and v.

Return to EES. Next, select the Open or Create Macro command from the File menu and enter a file name. The file name used in this example is C:\Temp\Excel1.emf. The macro will open the Excel file, copy the temperature and pressure data to the Parametric Table, run the Parametric Table, and finally copy the properties calculated in EES back to Excel. The Excel file name is assigned to the string File$. The Excel.Open macro command is used to open the file and the Excel.Show macro command is used to bring the Excel file to the front.

```
File$='C:\Temp\Excel_ws1.xls'          //name of Excel file
Excel.Open(File$)                      //open the Excel file
```

```
Excel.Show                          //make the file visible
```

The Excel.Copy macro command is used to copy the values in the Excel file in the range A2:B11 to the clipboard. The Paste Parametric macro command is used to paste the values in the clipboard into the Parametric Table starting at row 1 and column 1 (R1 C1).

```
Excel.Copy('A2:B11')                //copy the temperature & pressure data to the clipboard
Paste Parametric 'Table 1' R1 C1    //paste the data into the Parametric Table
```

The Parametric Table is solved using the Solve Table macro command and the Copy Parametric macro command is used to copy the values in the table from row 1 column 3 to row 10 column 5 (R1 C3 R10 C5) to the clipboard.

```
SolveTable 'Table 1'                                //solve the Parametric Table
Copy ParametricTable 'Table 1' R1 C3 R10 C5         //copy the results in the Parametric Table to clipboard
```

The insertion point in Excel is set to cell C2 using the Excel.Range macro command and the contents of the clipboard are pasted into Excel using the Excel.Paste macro command.

```
Excel.Range('C2:C2')    //set the insertion point in the Excel document
Excel.Paste             //paste the contents of the clipboard into Excel at the insertion point
```

The Excel file is saved using the Excel.FileSaveAs macro command. Execution is paused for 2 seconds using the Pause macro command and finally the Excel file is closed using the Excel.Quit macro command.

```
Excel.FileSaveAs(File$)     //save the file
Pause 2                     //pause for 2 seconds
Excel.Quit                  //close Excel
```

Type the macro commands listed above into the Macro Window, as shown in Figure 18-12. If you saved the Excel file with a different name or in a different directory, then change the file name accordingly. When you are sure that all of the macro commands have been correctly entered, click the Save button () in the Macro Window tool bar and then click the lock button  so that no additional changes can be made to the Macro window. Click the Play button  to start the calculations.

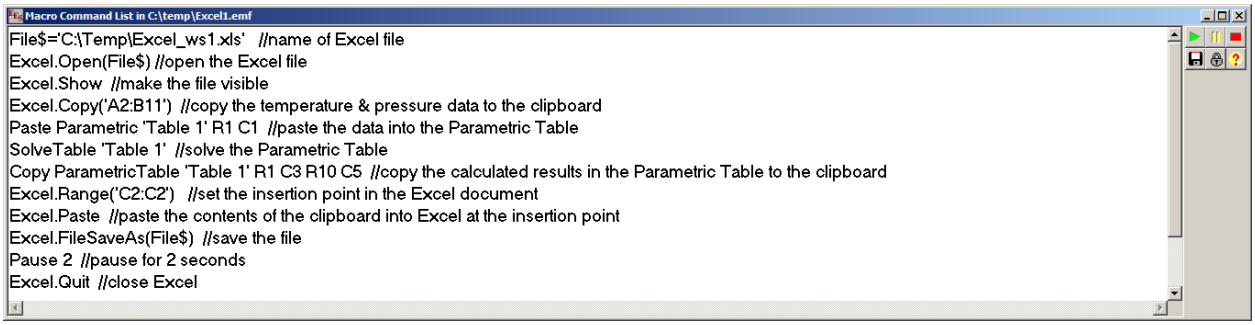


Figure 18-12: EES Macro Window with macro commands to interact with Excel.

The macro will open the Excel file that you created previously and display it. It will then copy the values of T and P from the first two columns of the Excel sheet to the EES Parametric Table. The Solve Table macro command will fill the Parametric Table with values of calculated specific enthalpy, specific entropy and specific volume that are copied to the clipboard and pasted back into Excel. The modified Excel file is saved with the same file name. The Pause 2 macro allows you to see the modified file before Excel is closed.

Interacting with MATLAB

The EES macro command set includes commands specifically to interact with MATLAB. MATLAB provides a powerful computing environment that can be used within an EES program through appropriate macro commands. As a simple example, we will use MATLAB to compute the Airy function for an array of numbers. The array is generated in the Equations Window:

```
$TabStops 0.25 in

Duplicate i=1,10
   X[i]=i/10
End
```

The macro starts by solving the Equations Window and then exporting the X array values to the file C:\Temp\Input.dat that can subsequently be opened by MATLAB.

```
Solve
Export 'C:\Temp\Input.dat' X[1..10]          //export the array to a file
```

An application of MATLAB is opened using the MATLAB.Open macro command. MATLAB.Execute macro commands are used to open the file, execute the Airy command in MATLAB and finally save the results to the file C:\Temp\Output.txt.

```
MATLAB.Open //open MATLAB
MATLAB.Execute('A=importdata('C:\Temp\Input.dat')')
MATLAB.Execute('B=airy(A)')
MATLAB.Execute('save('C:\Temp\Output.txt','B','-ascii')')
```

The Import macro command is used to read the data in the file C:\Temp\Output.txt into the array Y. The Solve macro command is executed in order to have EES rebuild the Arrays Table. Then arrays X and Y are used to create a plot using the NewPlot macro command.

```
Import 'C:\Temp\Output.txt' Y[1..10]
Solve
NewPlot Name='Plot 1' Table=ARR1  X=X[i]  Y=Y[i]  Rows=1..10  Line=1  Symbol=0  Color=0
```

The result of running the macro should be the plot of the Airy function shown in Figure 18-13.

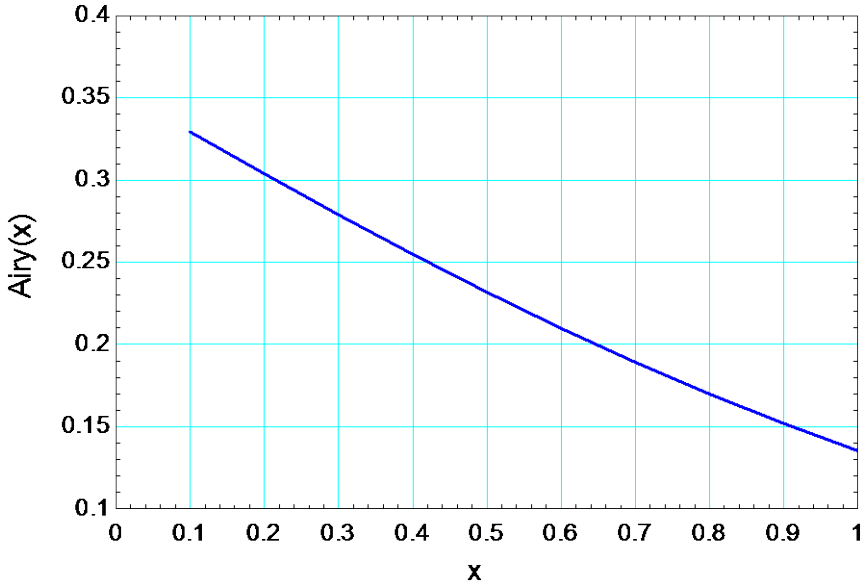


Figure 18-13: Airy function.

18.4 Executing EES Macros from External Programs

Section 18.1 shows how a macro file can be created and played in the EES Macro Window. This capability is very convenient because it allows a complex set of EES commands to be scripted. However, the use of macros is not limited to the EES Macro Window. An EES macro file is a simple text file and it can be created and/or edited in any text editor, for example Notepad. It can also be created or modified programmatically. Once created, a macro command list can be executed by starting EES from an external program, as described in this section. An alternative way to interact with EES is to issue commands to EES with Dynamic Data Exchange (DDE), which is described in Section 18.5.

We will use a very simple set of macro commands to illustrate how EES can be controlled from external programs. In this example, we will use EES to return property information for a specified fluid. The EES program that will be executed contains the following equations.

```
v=volume(R$,T=T,P=P)
h=enthalpy(R$, T=T,P=P)
s=entropy(R$, T=T,P=P)
```

Enter these equations into EES and save the file as C:\Temp\TestMacro.ees. (Change the path name throughout this example as needed to operate on your computer system.) Next, we'll create a macro file that opens the EES file, imports the fluid, temperature and pressure of interest from a file, calculates the properties, exports the results to a file, and closes EES. The macro can be created in the EES Macro Window or using any text editor.

```
Open 'C:\Temp\TestMacro.ees'
Units SI C kPa J Mass
```

```
Import 'C:\Temp\Input.dat' R$ T P
Solve
Export 'C:\Temp\Output.dat' R$ T P v h s
Quit
```

Save the macro file as C:\Temp\aMacro.emf. Note that the file name extension must be .emf to indicate an EES Macro File.

Executing an EES Macro File from the Windows Run dialog

We can now run this simple macro from external programs. We will first run it from the Windows Run dialog which is accessed from the Start menu. Use a text editor to create the file C:\Temp\Input.dat that contains the following line of text:

```
'R134a' 50 200
```

corresponding to the fluid R134a at 50ºC and 200 kPa.

Next, enter EES C:\Temp\TestMacro.emf into the edit space of the Run dialog, as shown in Figure 18-14, and click the OK button.

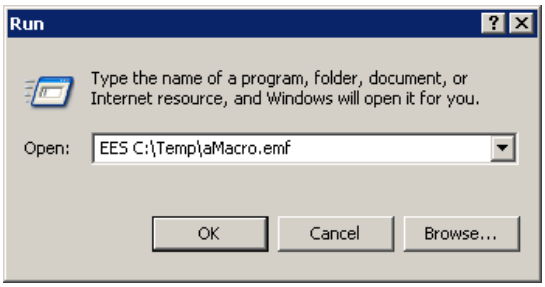


Figure 18-14: Starting EES and a macro command list from the Windows Run dialog.

Note that EES accepts the name of the macro file (in this case C:\Temp\aMacro.emf) as an input parameter as well as an EES file name. When EES is provided with the name of a macro file, it will start EES in a hidden state and execute the macro commands quietly. The only indication that EES is running is the appearance of the EES icon in the application tray at the bottom of the screen. The macro command set opens the EES file that calculates properties and then writes the results to the file C:\Temp\Output.dat. After the command has completed, you should find the Output.dat file in the C:\Temp\ directory. Open the Output.dat file to see the computed properties:

```
'R134a' 5.00000000E+01 2.00000000E+02 1.27661555E-01 2.96696970E+05 1.11640747E+03
```

corresponding to v = 0.1277 m^3/kg, h = 2.9670x10^5 J/kg, and s = 1.1164x10^3 J/kg-K. The EESMacro.log file should also appear in this directory, showing each command that is executed in the macro.

Executing an EES Macro File from MATLAB

EES can be started and directed to run a predefined macro file from most programs. This capability allows other programs to communicate with EES and direct its execution. To illustrate this capability, we will develop a MATLAB function that returns the specific volume, specific enthalpy, and specific entropy given the fluid, temperature, and pressure.

The MATLAB function is called EESProperty and the header is given below:

```
function[v, h, s] = EESProperty(R, T, P)
    %[v,h,s]=EESProperty(R,T,P)
    %
    % Inputs
    % R = an EES recognized fluid
    % T = temperature (C)
    % P = pressure (kPa)
    %
    % Outputs
    % v = specific volume (m^3/kg)
    % h = specific enthalpy (J/kg)
    % s = specific entropy (J/kg-K)


end
```

The file C:\Temp\Input.dat is opened using the fopen command in MATLAB. The 'w' flag indicates that the file is available to write to and any existing contents are flushed. The fprintf command is used to write the inputs R, T, and P to the file. Notice the formatting required to put single quotes around the string R so that it is read appropriately as a string in EES. The file is closed using the fclose command.

```
    fid=fopen('C:\Temp\Input.dat','w');      %open the input file for writing
    fprintf(fid,'''%s'' %f %f',R,T,P);       %write the inputs to the file
    fclose(fid);                             %close the file
```

The system command in MATLAB executes a command in the operating system as if it were entered into the Run Window. We will use the system command to open EES with the macro file name as an input parameter.

```
    system('C:\EES32\EES.exe  C:\Temp\aMacro.emf'); %run the macro
```

EES will open and automatically execute the commands in the macro C:\Temp\aMacro.emf. This causes EES to open the file C:\Temp\Input.dat and assign the contents to the variables R$, T, and P in the EES file. The file is solved and the properties v, h, and s are written to the file C:\Temp\Output.dat. Finally, EES is closed. It is necessary to pause within MATLAB while this process occurs in order to allow EES time to finish these operations. The pause command in MATLAB is used for this purpose. The duration of the pause that is required will be different for different machines; the code below waits for 2 seconds.

```
    pause(2);                                %wait 2 seconds
```

All that remains is to open the file C:\Temp\Output.dat and assign the values to the output parameters v, h, and s in the MATLAB function EESProperty. This is done using the importdata command in MATLAB.

```
TPvhs=importdata('C:\Temp\Output.dat'); %import results from the file
v=TPvhs.data(3);                        %assign the results to outputs
h=TPvhs.data(4);
s=TPvhs.data(5);
```

The MATLAB function EESProperty can now be run like any other MATLAB function in order to provide access to EES' internal property routines. To calculate the properties of R134a at 50ºC and 200 kPa, enter the following lines in the MATLAB Command Window:

```
>> R='R134a';T=50;P=200;
>> [v,h,s]=EESProperty(R,T,P)
v =
   0.1277

h =
  2.9670e+005

s =
  1.1164e+003
```

which leads to v = 0.1277 m^3/kg, h = 2.9670x10^5 J/kg, and s = 1.1164 J/kg-K. Notice that there is no visual indication that EES has started and executed the macro command list. However, you see that the C:\Temp\EESMacro.log file appears in the directory.

Executing an EES Macro File from Excel

It is necessary to create a Macro in Excel in order to run EES. This process is described here using Microsoft Office Excel 2010. The process is slightly different for other versions of Excel. Start Excel and click the select the Developer Tab, as shown in Figure 18-15. Note that it may be necessary to right-click in the menu space and select Customize the Ribbon from the pop-up menu that appears; add the Developer menu to the Main Tabs.

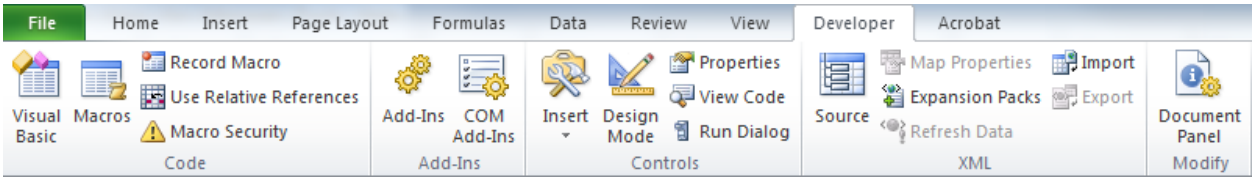


Figure 18-15: Menu bar that provides access to macros in Excel.

Select Visual Basic from the Developer tab in order to access Visual Basic, as shown in Figure 18-16.

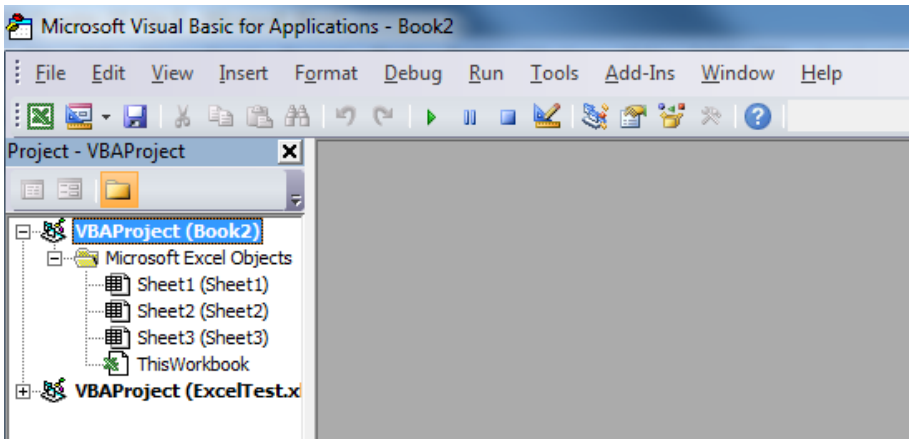


Figure 18-16: Visual Basic window.

Select Module from the Insert menu in order to create a new module. The module will contain a function Volume that returns the specific volume given the fluid name, temperature, and pressure. Declare the function with three arguments (R, T, and P) as shown below. Notice that the type of each argument as well as the type of the value returned by the function are declared.

```
Function Volume(R As String, T As Double, P As Double) As Double

End Function
```

In order to write the string to a file in a name that is recognized by EES it must be surrounded by single quotes. A temporary string, RR is declared and formed.

```
  Dim RR As String

  RR = "'" + R + "'"
```

Note that the single quote is surrounded by double quotes. The file C:\Temp\Input.dat is opened for output using the Open command in Visual Basic. The quantities RR, T, and P are written to the file and it is closed. The macro aMacro.emf is modified so that only specific volume is returned using the Notepad program, as shown in Figure 18-17.

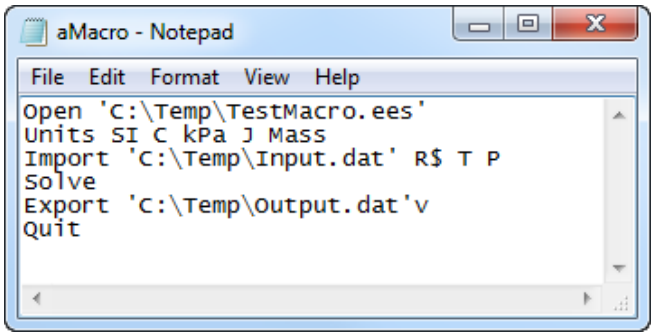


Figure 18-17: Modified macro aMacro.emf that returns only specific volume.

The Shell command in Visual Basic is used to execute system commands. The EES program is opened with the macro name provided as the parameter. The EES program TestMacro.ees is opened and the inputs R$, T, and P are read from the file C:\Temp\Input.dat. The program is solved and the value of the calculated specific volume, v, is written to the file C:\Temp\Output.dat.

```
myShell = Shell("C:\\EES_CAE\EES.exe C:\\Temp\aMacro.emf")
```

Next it is necessary to pause for a sufficient amount of time to allow EES to complete its calculations and write the output file. This may be different for different machines; the code below pauses for 2 seconds.

```
PauseTime = 2
Start = Timer
Do While Timer < Start + PauseTime
  DoEvents
Loop
```

The file C:\Temp\Output.dat is opened by Excel and the value of Volume is read using the Input command.

```
Open "C:\Temp\Output.dat" For Input As #1
Input #1, Volume
Close #1
```

The Visual Basic function Volume is shown in Figure 18-18.

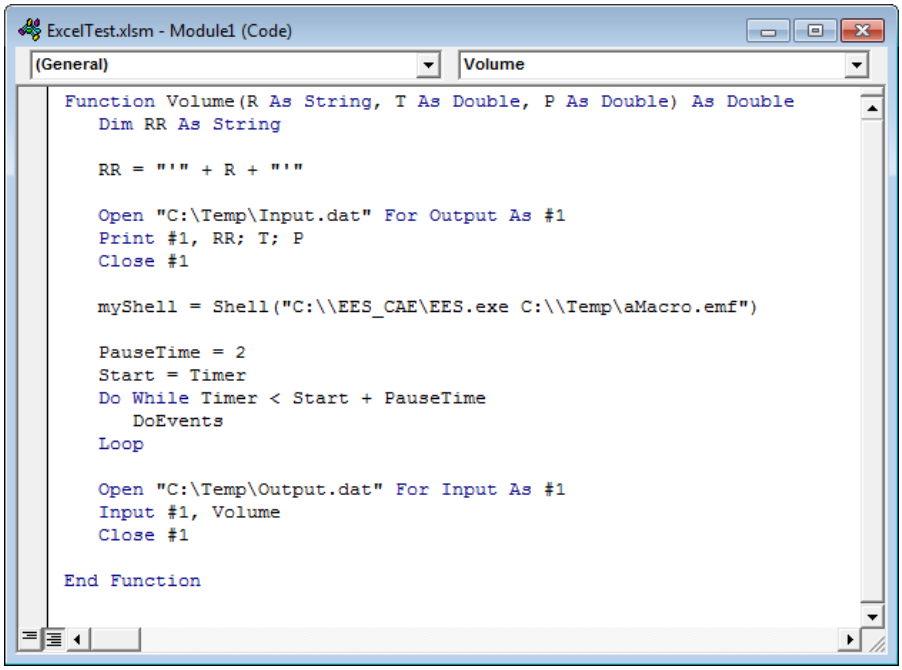
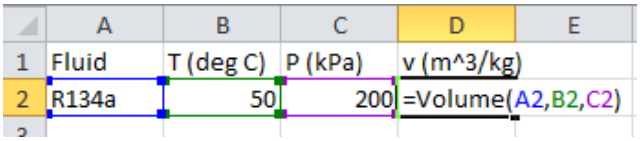


Figure 18-18: Visual Basic function Volume.

The function Volume can be used like any other Excel function. For example, enter the fluid name, temperature, and pressure into sequential cells and then call the function Volume referring to these cells as inputs, as shown in Figure 18-19(a). The value returned by Volume is shown in Figure 18-19(b). Figure 18-19(c) illustrates a range of values filled in using the Volume function.



	A	B	C	D	E
1	Fluid	T (deg C)	P (kPa)	v (m^3/kg)	
2	R134a	50	200	=Volume(A2,B2,C2)	

(a)

	A	B	C	D
1	Fluid	T (deg C)	P (kPa)	v (m^3/kg)
2	R134a	50	200	0.127662

(b)

	A	B	C	D
1	Fluid	T (deg C)	P (kPa)	v (m^3/kg)
2	R134a	50	200	0.127662
3	R134a	60	200	0.132056
4	R134a	70	200	0.136414
5	R134a	80	200	0.140742
6	R134a	90	200	0.145045
7	R134a	100	200	0.149325

(c)

Figure 18-19: (a) Calling the function Volume from the Worksheet, (b) the value returned, and (c) a range of values returned.

Executing an EES Macro File from LabView

In the previous examples it was necessary to open EES each time that we wanted to have the external program execute a command. This process takes some significant time and computational overhead. An alternative is to have EES running a macro continuously in the background. The macro can check the timestamp of a file that is written by the external program. When the timestamp changes, EES will read the file, execute a set of commands and then write an output file. In this way, the external program does not need to restart EES each time a command is executed.

We can write an interface between LabView and EES using this technique. Open EES and write a simple program that computes the specific volume given the fluid, temperature, and pressure.

```
$UnitSystem SI Mass J K Pa
v=volume(R$,T=T,P=P)
```

Save the EES program as TestMacro.ees in a temporary directory. Select Open or Create Macro from the File menu and save the macro file as monitor.emf. The macro file first deactivates writing to the log file using the LogFileName macro command; this is necessary in order to avoid creating an endlessly growing log file as the loop executes continuously.

```
LogFileName                    //Deactivate writing to the log file
```

Next, the input file name is assigned to the string variable File$. The timestamp for this file is obtained using the TimeStamp macro command.

```
F$='C:\Temp\Input.dat'         //Name of the input file
old=TimeStamp(F$)              //get the timestamp of the input file
```

Next we will program the loop.

```
10:
  Pause 0.2                    //pause 0.2 seconds between loops
  new = TimeStamp(File$)       //get the new timestamp of the file
  If (new>old) Then Goto 20    //if the timestamp has changed then execute commands
Goto 10
```

The first line is a label that can be used with Goto statements. The next line pauses 0.2 seconds between each execution of the loop in order to avoid eating up all of the computer resources. The third line obtains the new timestamp of the input file. If the timestamp changes then execution is directed to line 20 where any commands that are to be executed upon detecting a new input file will be placed. Finally, the loop is repeated using the Goto 20 statement.

Commands are entered after line 20 as shown below.

```
20:
Beep 2                         //make a noise
old=new                        //reset the time stamp
Goto 10                        //return to the loop
```

In this case, each time the timestamp of the input file changes, EES will make a noise and then reset the timestamp and re-enter the loop. Using NotePad, prepare a simple input file as shown in Figure 18-20.

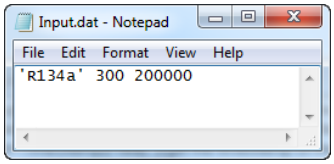


Figure 18-20: Input.dat file.

Save the Input.dat file and then start the EES macro. Each time you save the Input.dat file you should hear a beep, indicating that EES has detected that the timestamp of the file changed at which time the commands between 20: and Goto 10 are executed.

We can use this macro as the basis of an interface between EES and any other program. In this example, we will write a LabView Virtual Instrument (VI) that allows the user to enter the fluid, temperature, and pressure and then interact with EES in order to return the specific volume. When the timestamp of the file Input.dat changes, the EES macro will import the parameters R$, T, and P from the file, solve the equations, and export the resulting value of specific volume to the file Output.dat. The resulting macro is shown below:

```
LogFileName                          //Deactivate writing to the log file
File$='C:\Temp\Input.dat'            //Name of the input file
old=TimeStamp(File$)                 //get the timestamp of the input file
10:
  Pause 0.2                          //pause 0.2 seconds between loops
  new = TimeStamp(File$)             //get the new timestamp of the file
  If (new>old) Then Goto 20          //if the timestamp has changed then execute commands
Goto 10

20:
  Beep 2                             //make a noise
  Import File$ R$ T P                //get the input parameters from the file
  Solve                              //solve the equations to compute the specific volume
  Export 'C:\Temp\Output.dat' v      //export the value of v to the output file
  old=new                            //reset the time stamp
Goto 10                              //return to the loop
```

In LabView, start a new VI and place two numeric controls (for temperature and pressure) and one string control (for fluid) on the front panel. Place a numeric indicator (for specific volume) on the front panel. The result is shown in Figure 18-21.

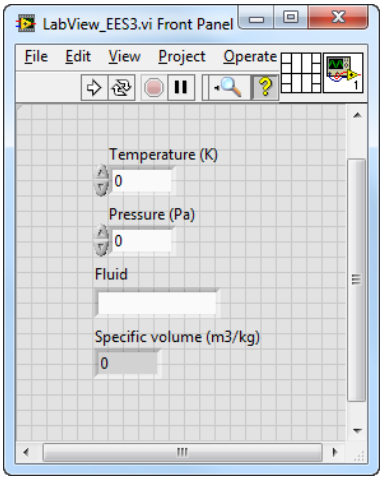


Figure 18-21: Front panel of LabView VI.

On the block diagram, place a flat sequence structure, as shown in Figure 18-22. In the first frame, obtain the current time stamp of the file Output.dat using the File/Directory Info subvi which can be found in the Programming>File I/O>Advanced File Functions> palette. In the second frame, open the file Input.dat, write the inputs to the file, and close the file. The input string is formed by concatenating the strings corresponding to the individual inputs. Note that single quotes were added around the fluid name and spaces were added between inputs. The subvi Number to Fractional String is used to convert the numerical inputs to strings. In the third frame place a while loop that continuously reads the time stamp of the file Output.dat and terminates when the time stamp changes relative to the value read in the first frame. Finally, the fourth frame opens the file Output.dat and reads the specific volume. The string is converted to a number using the Fract/Exp String to Number subvi and displayed using the indicator for specific volume.

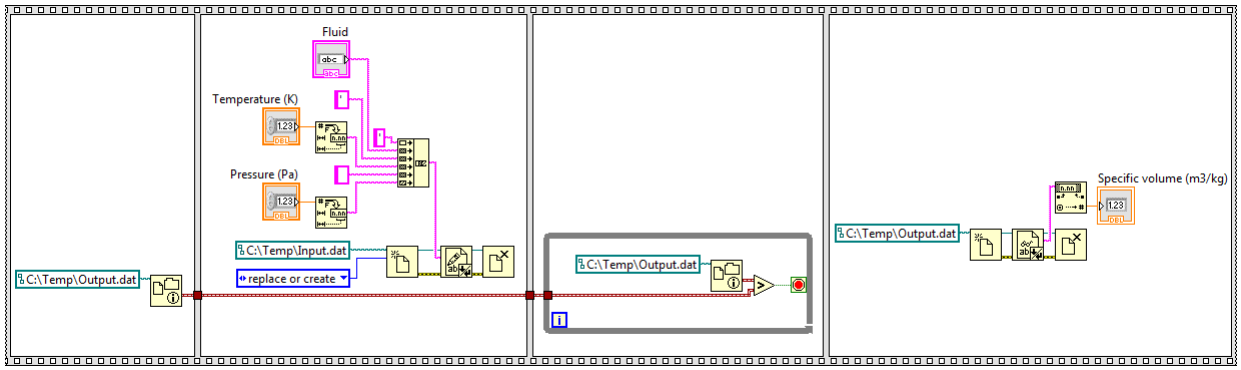


Figure 18-22: Block diagram of LabView VI.

To run the program, start the EES macro and then enter some reasonable inputs into the controls on the LabView VI front panel. Run the VI and you should hear a beep (corresponding to EES running) and then the value of specific volume should be displayed, as shown in Figure 18-23.

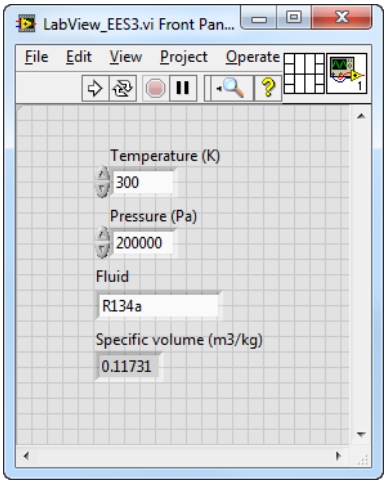


Figure 18-23: LabView front panel during operation.

Executing an EES Macro File from DELPHI

EES can be started and directed to run a macro file from within a compiled program. This capability is illustrated using the DELPHI XE2 compiler, which is the compiler used to develop the EES program itself.

Start DELPHI XE2 and select VCL Forms Application – DELPHI from the New menu item in the File menu. Place a button and an edit box on Form1, as shown in Figure 18-24.

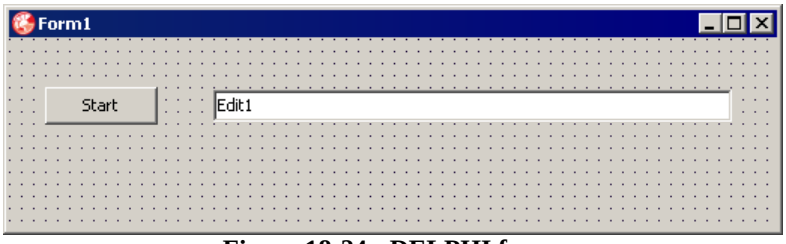


Figure 18-24: DELPHI form.

Double click on the Start button and enter the code shown in Figure 18-25. This code picks up the name of the file entered in the edit box and concatenates it to the string C:\EES32\EES.exe with an intervening space. The code then calls the Windows CreateProcess routine to start the specified program. Run the program and enter C:\EES32\TestMacro\aMacro.emf in the edit box. Then click the Start button. A message will appear to indicate whether or not the process was successful.

```
procedure TForm1.StartClick(Sender: TObject);
  type CharString=array[0..255] of Char;
  var CallProg:CharString;
      Name,S:ShortString;
      SU:TStartupInfo;
      PI:TPROCESSINFORMATION;
      Worked:boolean;
      //Tag,p:integer;
  begin
    Name:='C:\EES32\EES.exe '+Edit1.Text;
    StrPCopy(CallProg,Name);
    ZeroMemory(@SU,sizeof(TStartupInfo));
    ZeroMemory(@PI,sizeof(TPROCESSINFORMATION));
    SU.cb:=Sizeof(TStartUpInfo);
    SU.dwFlags:=STARTF_USESHOWWINDOW;
    SU.wSHOWWINDOW:=SW_ShowNormal;
    Worked:=CreateProcess(
      nil,  // pointer to name of executable module
      CallProg, // pointer to command line string
      nil,  // pointer to process security attributes
      nil,  // pointer to thread security attributes
      false,  // handle inheritance flag
      0 , // creation flags
      nil,  // pointer to new environment block
      nil,  // pointer to current directory name
      SU, // pointer to STARTUPINFO
      PI  // pointer to PROCESS_INFORMATION
        );
      if (Worked) then
        MessageDlg('Executing '+Name, mtInformation, [mbOk], 0)
      else
        MessageDlg('Error encountered while trying to execute '+Name, mtInformation, [mbOk], 0);
end;
```

Figure 18-25: DELPHI code to start EES with a specified file name.

18.5 Interacting with EES using Dynamic Data Exchange

An alternative method that can be used by an external program to communicate with EES is Dynamic Data Exchange (DDE). Dynamic Data Exchange is an information exchange format within the Windows operating systems that allows instructions to be passed between different applications. Most Windows programs support this information exchange system, including Excel and MATLAB, however the communication protocol is slowly being phased out. Most compilers also provide instructions that will allow DDE messages to be passed to another program. All of the EES macro commands listed in Section 18.2 can be initiated with DDE. In addition, the DDE protocol implemented in EES provides Run EESFile and Play MacroFile commands that allow calculations to be initiated by EES after EES is loaded. In this way the overhead associated with loading EES is eliminated and the calculations are faster. This section will demonstrate how EES macro commands can be initiated from Excel, MATLAB and a compiled language.

Dynamic Data Exchange with Excel

This section provides an example of an Excel macro that is written in Visual Basic in order to interact with EES using Dynamic Data Exchange. It should serve as a model of how EES can be integrated with Excel. Note that EES can also communicate with Excel directly using the Excel macro commands that are described in Section 18.2. Also Excel can start EES and execute a macro command file as described in Section 18.3.

The Excel macro can be implemented on a form in Excel with 5 buttons, as shown in Figure 18-26. (See the Excel Visual Basic instructions for creating a form.) The Visual Basic code that accompanies each of these buttons is shown in Figure 18-27.

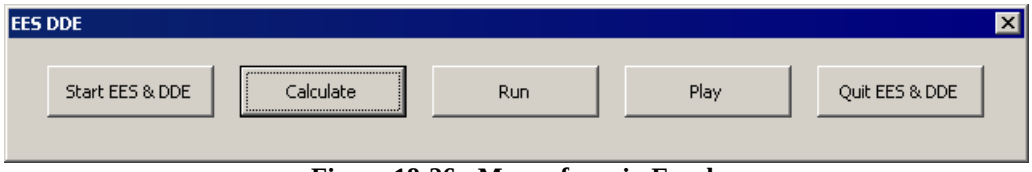


Figure 18-26: Macro form in Excel.

RunButton | Click

```
Dim ChannelNumber As Integer
Private Sub StartEES_Click()
    'Open EES
    Shell_R = Shell("C:\EES32\EES.exe /Hide", 1)
   'Initiate DDE
    ChannelNumber = -1
    ChannelNumber = Application.DDEInitiate(app:="EES", topic:="DDE")
    If ChannelNumber <> -1 Then
            MsgBox "Ready for Calculations", vbExclamation, "EES DDE"
    Else
            MsgBox "Unable to initialize DDE", vbExclamation, "EES DDE"
    End If
End Sub
Private Sub cmdDDE_Click()
    Dim myShell As String
    'Copy selected rows into clipboard which will be pasted into a Parametric Table in EES'
    Range("A11:B20").Select
    Selection.Copy
    Application.DDEExecute ChannelNumber, "[Open C:\EES32\USERLIB\Examples\EXCEL_EES.ees]"
    Application.DDEExecute ChannelNumber, "[Paste Parametric 'Table 1' R1 C1]"
    Application.DDEExecute ChannelNumber, "[SOLVETABLE 'TABLE 1' Rows=1..10]"
    Application.DDEExecute ChannelNumber, "[COPY ParametricTable 'Table 1' R1 C3:R10 C5]"
    ActiveSheet.Paste Destination:=Worksheets("Sheet1").Range("C11:E20")
End Sub
Private Sub RunButton_Click()
    Application.DDEExecute ChannelNumber, "RUN C:\EES32\USERLIB\Examples\CH1EX.ees"
End Sub
Private Sub PlayButton_Click()
    Application.DDEExecute ChannelNumber, "PLAY C:\EES32\TestMacro\aMacro.emf"
End Sub
Private Sub QuitButton_Click()
        'Quit EES and Terminate DDE
        DDEExecute ChannelNumber, "[QUIT]"
        Application.DDETerminate ChannelNumber
        frmEESDDE.Hide
End Sub
```

Figure 18-27: Visual Basic code for implementation of DDE with EES.

The code begins by defining ChannelNumber as a global variable. The Start EES & DDE button runs the StartEES_click subroutine. The subroutine first starts EES in the manner described in Section 18.3 using the Shell command. The location of the EES program is assumed to be in the directory C:\EES32\. You will need to change this path if your version of EES is installed elsewhere. After starting EES, the Excel macro executes the DDEInitiate command which broadcasts a message to all open applications. Any application that can recognize EES and DDE codes responds and Excel stores its address within the variable ChannelNumber.

Clicking the Calculate button runs the Sub_cmdDDE_Click subroutine. The Excel macro begins by copying two columns of numbers that provide values of temperature and pressure. Then it issues a series of macro commands to EES using the DDEExecute command. The macro command is placed inside of square brackets, as shown in Figure 18-27. Any of the macro commands listed in Section 18.2 can be sent to EES in this manner. In this case, EES is instructed to open the file EXCEL_EES.ees, paste the temperature and pressure data from the Excel sheet in the EES Parametric table, run the table, copy the third, fourth and fifth columns to the clipboard and paste them back into Excel. The macro is run each time the Calculate button is clicked. EES remains loaded and it does not have to be restarted each time the calculations are initiated. The Excel sheet should appear as shown in Figure 18-28 after calculations are completed.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	This EXCEL sheet contains a macro that will call EES. It should serve as a model of how EES and EXCEL can interact.												
2													
3	The macro will start EES and open the file C:\ees32\userlib\examples\EXCEL_EES.ees. It will then copy two columns of data representing												
4	temperature and presssure to the clipboard, paste them iinto the first two columns of the EES Parametric table 'Table 1', solve the												
5	table and copy the the calculated results back to the clipboard. In this case, the calculated results are the specific enthalpy, entropy,												
6	and volume for for refrigerant R134a. The EXCEL macro then pastes the result into the EXCEL sheet.												
7													
8	To view the code editor hit Alt+F11 or Tools->Macro->Visual Basic Editor.												
9													
10	T [C]	P [kPa]	h [kJ/kg]	s [kJ/kg-L]	v [m3/kg]								
11	100	500	3.41E+02	1.17E+00	5.81E-02								
12	110	500	3.51E+02	1.20E+00	5.99E-02								
13	120	500	3.61E+02	1.22E+00	6.17E-02								
14	130	500	3.71E+02	1.25E+00	6.35E-02								
15	140	500	3.81E+02	1.27E+00	6.53E-02								
16	150	500	3.92E+02	1.30E+00	6.70E-02								
17	160	500	4.03E+02	1.32E+00	6.88E-02								
18	170	500	4.14E+02	1.35E+00	7.05E-02								
19	180	500	4.25E+02	1.37E+00	7.23E-02								
20	185	500	4.30E+02	1.39E+00	7.31E-02								
21													

Figure 18-28: Excel Sheet after the macro has been run.

Clicking the Run button will execute the DDE RUN command in EES. EES must be running with DDE initiated before this button is clicked. When it receives this command, EES will close any open file and open and solve the specified file, which in this case is the CH1EX.ees example problem in the UserLib examples folder. After solving the file, EES will write the contents of the Solution Window to a file having the same name as the EES file, but with a .txt file name extension. Note that the Run command is not an EES macro command and therefore is not enclosed in square brackets.

Clicking the Play button will play the specified macro file. Note that, as with Run command, the Play command is not an EES macro command and therefore it is not enclosed in square brackets. The Run and Play commands do not restart EES, so they execute relatively fast.

The Quit EES & DDE button quits EES and disconnects the DDE channel.

Dynamic Data Exchange with MATLAB

This section provides an example MATLAB script that will interact with EES using Dynamic Data Exchange. The comments in the script describe the commands. It should serve as a model of how EES can be integrated with MATLAB. Note that EES can communicate with MATLAB directly using MATLAB macro commands described in Section 18.2. MATLAB can also start EES and initiate a macro command list as described in Section 18.3.

Write the simple EES program shown below. The program reads the values of T and P from the clipboard, uses the Volume command to compute the specific volume, and then writes the parameter v to the clipboard.

```
$UnitSystem SI Mass J K Pa

$Import 'ClipBoard' T, P
v=volume('R134a',T=T,P=P)
$Export 'ClipBoard' v
```

We will write a script in MATLAB that communicates with EES via DDE. The temperature and pressure of interest are assigned.

```
T=300;                                  %temperature (K)
P=250000;                               %pressure (Pa)
```

The ddeinit command is used to initiate a DDE conversation with EES. The service is 'EES' and the topic is 'DDE'.

```
channel = ddeinit('EES','DDE');         %open a DDE channel with EES
```

The clipboard command is used to copy the values of T and P to the clipboard. Note that these values are converted to strings using the num2str command and the strings are separated by a comma.

```
clipboard('copy',[num2str(T),', ',num2str(P)]);   %copy the inputs to the clipboard as a string
```

The ddeexec command is used to execute a macro command in EES. In this case, the equations are solved which causes T and P to be read from the clipboard, v to be calculated, and finally v to be written to the clipboard.

```
rc = ddeexec(channel,'[Solve]');        %solve the equations in EES
```

The value of v is read from the clipboard and converted from a string to a number using the str2num command.

```
v=str2num(clipboard('paste'));          %copy the output from the clipboard & convert to number
```

Finally, the DDE channel is closed using the ddeterm command.

```
ddeterm(channel);                       %terminate the DDE channel
```

Start EES and click the minimize button. Run the script and you should see that the variable v is assigned in the Command Window. This approach can be expanded to create a sophisticated interface between MATLAB and EES. Note that MATLAB can send any of the macro commands listed in Section 18.2 to EES in this manner.

Dynamic Data Exchange with DELPHI

This section demonstrates how DDE can be implemented in a DELPHI program. Implementation in C++ and other languages is similar. The Professional version of EES must be running to receive a DDE message. This is not really a problem as EES can be started from a remote application using the Windows CreateProcess command, as shown in Section 18.3. However, when EES starts, it normally displays its splash screen and waits for the user to click the OK button. To avoid having EES display the splash screen and wait for user input, supply /Hide as a parameter on the command line, i.e., start EES with the following command:

```
C:\EES32\EES.exe /Hide
```

After receiving this command, EES will start in a minimized mode. Its icon will be visible in the Windows task bar, but the program will otherwise not be visible on the screen.

The calling program must next send EES a series of messages according to the DDE message protocol. The first message that must be sent is the wm_DDE_initiate message. This message is normally sent using the Windows SendMessage command. The SendMessage command requires four parameters. The first should be a broadcast message to all running applications. The second must be wm_DDE_initiate. The third is the handle of the calling application. EES will use this handle to send a message back to the calling application. The final parameter is a global atom referring to 'EES', so that EES knows to act on this initiate message. Shown below is a code fragment in Delphi XE2 that sends the wm_DDE_initiate message to EES.

```
procedure TForm1.doInitiateClick(Sender: TObject);
  var theApp, theTopic:Atom;
      lParam:longInt;
begin
  theApp:=GlobalAddAtom('EES');
  theTopic:=GlobalAddAtom('DDE');
  lParam:=MakeLong(theApp,theTopic);
  SendMessage(HWND(-1),wm_DDE_initiate,Handle,lParam);
  GlobalDeleteAtom(theApp);
  GlobalDeleteAtom(theTopic);
end;
```

EES will respond to the wm_DDE_initiate message by sending the application a wm_DDE_ack message. Included in this message is the handle to the EES application, here called EESWindowHandle, that the calling program must provide in following messages. A code segment that receives the wm_DDE_ack message may appear as shown below.

```
procedure TForm1.GetACK(var theMessage:TMessage);
  Type charString:array[0..255] of char;
  var AppName,TopicName:charString;
      S:shortString;
      ReturnCode:smallInt;
 begin
 inherited;
   EESWindowHandle:=theMessage.wParam;
   ReturnCode:=LoWord(theMessage.lParam);
   S:='Acknowledgement received.';// with result code = '+inttoStr(ReturnCode);
   MessageDlg(S, mtInformation, [mbOk], 0);
 end;
```

After the wm_DDE_initiate message has been received by EES, the calling application can next send a wm_DDE_execute message with an EES Macro command enclosed in square braces. The code fragment below shows how the macro Solve command can be sent. Any macro command listed in Section 18-2 can be initiated in this manner. The macro commands must be enclosed with in square brackets, as shown for the Solve command.

```
procedure TForm1.ExecuteBtnClick(Sender: TObject);
  var cformat,command:atom;
```

```
    lParam:longint;
begin
  cformat:=GlobalAddAtom(' ');
  Command:=GlobalAddAtom('[Solve]');
  lParam:=MakeLong(cformat,Command);
  PostMessage(EESWindowHandle,wm_DDE_REQUEST,Handle,lParam);
end;
```

The Quit command terminates EES. The code below shows how it can be sent:

```
procedure TForm1.QuitClick(Sender: TObject);
var cformat,command:atom;
    lParam:longint;
begin
  cformat:=GlobalAddAtom('');
  Command:=GlobalAddAtom('QUIT');
  lParam:=MakeLong(cformat,Command);
  PostMessage(EESWindowHandle,wm_DDE_REQUEST,Handle,lParam);
end;
```

18.6 Useful Macro Examples

This section provides some useful example macro files. This first macro is used to create a Lookup Table that will be used for a complete factorial design study with N variables. The table contains 2^N rows and N columns. The table is populated with values of 0 and 1 for all combinations of the N variables. The macro listing for a 4-factor design follows:

```
N=4     //Number of variables
T$='Factorial Table'
Rows=2^N
NewLookup T$ Rows=Rows Cols=N  //Create the table
Row=-1
repeat
        Row=Row+1
        Rowp1=Row+1
        col=0
        repeat
                col=col+1
                V=Mod(trunc(Row/2^(N-Col)),2)
                Lookup[T$,Rowp1,Col]=V
        until (Col>=N)
until (Rowp1>=Rows)
```

After playing this macro file, a Lookup table named 'Factorial Design' will be created, as shown in Figure 18- 29. It would be possible to fill the Parametric table instead of the Lookup table using the Parametric[...] macro command. It would also be easy to modify this macro so that it fills the table with desired low and high values for each factor, rather than 0 and 1.

Lookup Table

Factorial Table

Paste Special	1 Column1	2 Column2	3 Column3	4 Column4
Row 1	0	0	0	0
Row 2	0	0	0	1
Row 3	0	0	1	0
Row 4	0	0	1	1
Row 5	0	1	0	0
Row 6	0	1	0	1
Row 7	0	1	1	0
Row 8	0	1	1	1
Row 9	1	0	0	0
Row 10	1	0	0	1
Row 11	1	0	1	0
Row 12	1	0	1	1
Row 13	1	1	0	0
Row 14	1	1	0	1
Row 15	1	1	1	0
Row 16	1	1	1	1

Figure 18- 29: Lookup table created for use in a factorial design analysis

19 EXTERNAL FUNCTIONS AND PROCEDURES

A function is a code segment that accepts one or more inputs and returns a single result associated with the function name. A procedure is similar to a function, but it can return one or more results and it is accessed by a Call statement. Functions and procedures that are entered directly in the Equations Window, as described in Chapter 3, are referred to as internal functions and procedures. Although they are originally entered in the Equations Window, they can subsequently be saved as library files having a .lib file name extension, as described in Chapter 11. A very powerful feature of EES is that functions and procedures can be written in any compiled language, such as Pascal, C, C++ or Fortran. These code segments, referred to as external functions and procedures, are used in exactly the same manner as internal EES functions and procedures that have been saved in library (.lib) files, as described in Chapter 11. They will be automatically loaded when EES starts if they are placed in the Userlib directory. Alternatively, they can be manually loaded using the Load Library command from File menu or the $Include directive. Help files can be provided for documenting external functions and procedures in the same manner as described in Chapter 11 for library files. External functions and procedures offer some advantages. Because they are written in a compiled language they will execute much faster than their internal counterparts. Also, the equations that are employed in external functions or procedures are not visible to the user. External functions and procedures are written as dynamic link library routines in the Windows operating system, as explained in this chapter.

19.1 EES External Functions (.dlf files)

External functions can be written in Pascal, C, C++, or any language that implements pointers and can produce a dynamic link library (DLL) that is compatible with the Windows operating system. The function statement header must have the following format:

```
Function MYFUNC(S, Mode, Inputs)
```

where the parameter S is a 256 ANSI character string, Mode is a 4-byte integer, and Inputs is a pointer to the head of a linked list of input values. The linked list structure consists of a double precision value and a pointer to the next input. The last input points to nil. Fortran 77 does not support pointers, therefore .dlf external functions cannot be written in Fortran 77. Pointers and linked lists are possible with Fortran 95, so external functions can be written with newer Fortran compilers. However, doing so requires conversion from Fortran pointers to C pointers. It is easier to write Fortran external procedures using the .fdl format that does not require use of pointers, as described in Section 19.3.

An external function must be written to accept one or more input arguments and use them to calculate a result that is assigned to the function name. An external function should also internally provide an example of it how it is to be used and check that the number of inputs supplied in the linked list is equal to the number of inputs that the function expects. To demonstrate the process, a simple function will be written in Pascal and C++. The function will

be called RMS and it will return the root mean square of up to 200 values that are supplied to it as arguments. The root mean square is defined in Eq. (19-14)

$$RMS = \sqrt{x_1^2 + x_2^2 + ...x_N^2} \tag{19-14}$$

External Functions written in Pascal

A skeleton listing of an external function written in Embarcadero's Delphi XE2 compiler is shown in Figure 19-1:

```
library MyFunc;
{$E .DLF}
type
   charstring=array[0..255] of ansichar;
   ParamRecPtr = ^ParamRec;
   ParamRec = record  { defines structure of the linked list of inputs }
     Value: double;
     next: ParamRecPtr;
   end;

function F(var S:charstring; Mode:integer; Inputs:ParamRecPtr):double;
export; stdCall;
  begin
     ...
     F:=Value; { F must be double precision }
  end;

exports F name 'MyFunc';

begin
end.
```

Figure 19-1: Skeleton listing of an external function for EES written in Delphi 2010.

The code begins with the Delphi keyword 'library', which instructs the compiler to compile the code as a dynamic link library (.dll) file. The project file name (MyFunc in Figure 19-1) follows the library keyword; the project has a .dproj file name extension. The {$E .DLF) directive instructs the compiler to save the compiler library file with the project file name and a .dlf file name extension, i.e., MyFunc.DLF. This is the name that EES will use to access the function

A string type (charstring) is defined as an array of 256 ANSI characters. The string can be used for input or output. The string is used to output an example of the proper use of the function and any error messages. A pointer to a data structure called ParamRec is defined. This pointer provides the linked list of inputs that are passed to the external function from EES when it is called. Each item in the linked list consists of a double precision value and a pointer to the next link. The last link must point to a null record (nil). EES will automatically construct this linked list when the function is called and pass it as the argument to the function.

The function (named F in Figure 19-1) must have the three arguments, as shown, and it must return a double precision value. The 'export' keyword directs the Delphi linker to make the function accessible to other programs. The 'stdcall' keyword is required to ensure that the

arguments are passed to and from EES in the order that EES expects. Note that the exported function is renamed from F to 'MyFunc' using the 'exports' keyword. This renaming is necessary as Delphi does not allow a function to have the same name as the project, and it is the project name that is assigned to the compiled function.

The parameter S is a 256-character ANSI C string terminated with an ASCII 0 (null character). S can be used for both input and output. If the first parameter provided in the EES function is a string constant (within single quotes) or string variable then EES will pass this string to the external routine. If an error is encountered, S should be set in the external routine to an appropriate error message. If the length of S is not zero, EES will terminate calculations and display S in an error message.

The parameter Mode is an integer set by EES. If Mode = –1, then EES is requesting that the function return in S an example of the function call. If Mode >= 0, then the function should simply return the function value. Currently, EES does not use the return value of Mode.

The parameter Inputs is a pointer to the head of a linked list of input values supplied by EES. The function should count the inputs to ensure that the number provided is as expected and issue an error message in S if this is not the case.

Figure 19-2 provides a complete listing of an external function named RMS written in Delphi XE2 that returns the root mean square of an array of values. Use the Build command from the Project menu in Delphi XE2 to create the file RMS.dlf. Copy this file into the EES Userlib folder and start EES to test it.

```
library RMS;
{$E .DLF}

const doExample = -1;
type
  CharString = array[0..255] of ansichar;
  ParamRecPtr=^ParamRec;
  ParamRec=record {structure of linked list}
    Value:double;
    next:ParamRecPtr;
  end;

 function CountValues (P: ParamRecPtr): integer;
  var N: integer;
  begin
    N := 0;
    while (P <> nil) do begin
      N := N + 1;
      P := P^.next
     end;
    CountValues := N;
  end; {CountValues}

function RMS_p(var S:CharString; Mode:integer; Inputs:ParamRecPtr):double;
    export; stdcall;
  var N:integer;
```

```
function RMSCalc: double;
 var NArgs: integer;
     SumSqrs: double;
     P:ParamRecPtr;
 begin
   RMSCalc:=0; {in case of error exit}
   S := '';
   P := Inputs; {first input in linked list}
   SumSqrs:=0;
   while (P<>nil) do begin
     SumSqrs := SumSqrs + sqr(P^.value);
     P := P^.next;
   end;
   RMSCalc := sqrt(SumSqrs);
 end; {RMSCalc}

begin
  RMS_p:=1;
  if (Mode = doExample) then begin
    S := 'RMS(X[1..N])';
    exit;
  end;
  N:=CountValues(Inputs);
  if (N<1) or (N>200) then
    S := 'The number of values must be between 1 and 200.'
  else
    RMS_p:=RMSCalc;
end; {RMS_p}

exports
  RMS_p name 'RMS';

begin
  {no initiation code needed}
end.
```

Figure 19-2: Complete listing of the RMS function written in Delphi XE2.

The function RMS will automatically load when EES is started, provided that it is found in the Userlib directory. You can confirm that it has been loaded using the Function Information dialog (Options menu) by clicking the External routines button, as shown in Figure 19-3. Select the RMS item that should appear in the list. EES will call the function with the request to return an example of how the function should be used, which is displayed in the Example box at the bottom of the dialog.

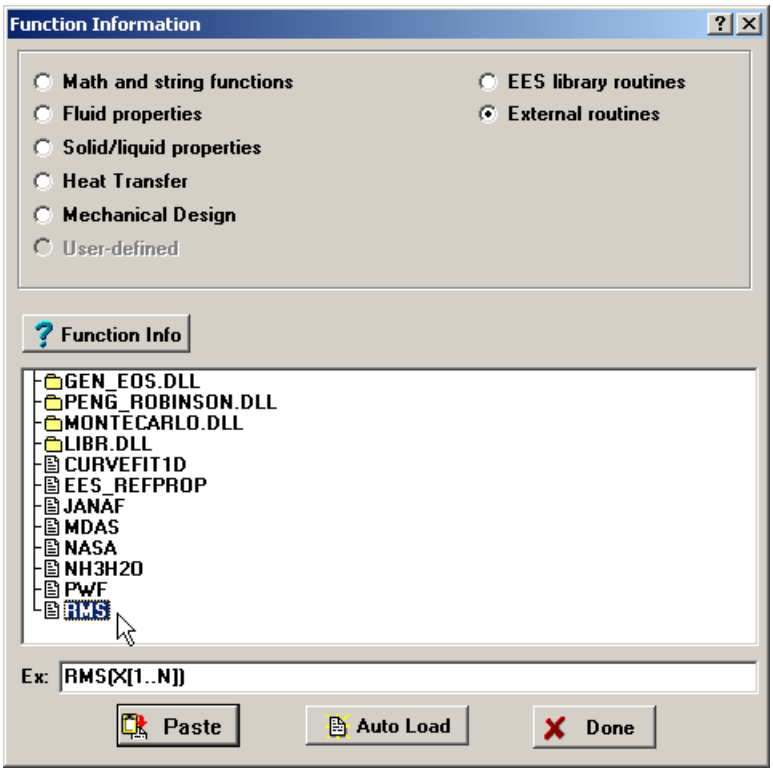


Figure 19-3: Function Information dialog showing RMS function and an example call.

Figure 19-4 provides a listing of an EES program that calls the RMS function.

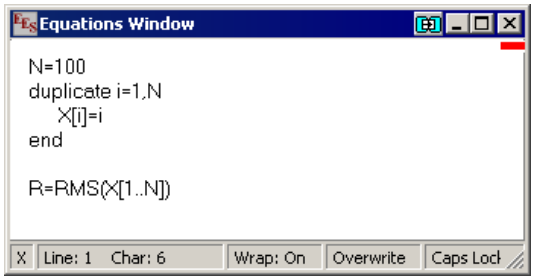


Figure 19-4: EES Equations window showing an example use of the RMS function.

The function returns 581.7. If you try to set N to be greater than 200 then the function should return an error message in string S that EES will display.

External Functions written in C++

A skeleton listing of an external function written in Microsoft's Visual C++ compiler is shown in Figure 19-5.

```
#include <windows.h>
#include <stdlib.h>
#include <math.h>

const int doExample=-1;
struct ParamRec {  // Structure for handling EES calling syntax
  double value;
  struct ParamRec *next;};

extern "C" // Use the "C" style calling convention to avoid C++ mangled names

{__declspec(dllexport) double RMS(char s[256],int mode,struct ParamRec
      *Inputs)
  {
   double Result;
   int NInputs;
   …
   …
   return Result;
  }
};
```

Figure 19-5: Skeleton listing of an external function for EES written in C++.

The project file starts by including the required libraries and then defining the constant doExample and the linked list ParamRec structure that will be used to pass values from EES. The structure consists of a value and a pointer to the next value. The last value is the null pointer, or 0.

The `extern "C"` statement is required for every function declaration having a C header in order to ensure that the function header is not "mangled", i.e., modified by the compiler. The `__declspec(dllexport)` keyword instructs the compiler to export the function with the required STDCALL protocol and eliminates the need for a .def file. Note that the function must accept three arguments. The first argument (S) is a 256 array of ANSI characters that will be primarily used to output an example of the function call or for error messages. The second argument (mode) is an integer. If mode = -1 then the function is expected to return a string with an example call. All other values of mode indicate that the function should return a calculated result. The third argument, Inputs, is a pointer to the first link of the linked list of inputs.

A complete listing of the RMS function written in Microsoft Visual C++ with Visual Studio 2010 is shown in Figure 19-6. This function should be compiled and linked as a Win32 Dynamic-Link Library with name RMS.dlf.

```
#include <windows.h>
#include <stdlib.h>
#include <math.h>

const int doExample=-1;
struct ParamRec {  // Structure for handling EES calling syntax
  double value;
  struct ParamRec *next;};

extern "C"// Use the "C" style calling convention to avoid C++ mangled names
int CountInputs(struct ParamRec *Inputs)
        {
        int NInputs=0;
        ParamRec *anInput=Inputs;
        while (anInput != 0) //Count Inputs
          {
                  anInput=anInput->next;
                  NInputs++;
          };
        return NInputs;
}

extern "C"
{__declspec(dllexport) double RMS(char s[256], int mode, struct ParamRec *Inputs)
        {double Result;
        int NInputs;
        ParamRec * anInput;
        if (mode==doExample) {
                strcpy(s,"RMS(X[1..N])");
                return 0;
        }
        NInputs= CountInputs(Inputs);
        if (NInputs<1 || NInputs>200) {
                strcpy(s,"The number of values provided to RMS must be between 1 and 200");
                return 0;}
        anInput = Inputs;
        Result=0;
        while (anInput!= 0)
        {
                Result = Result+(anInput->value)*(anInput->value);
                anInput = anInput->next;
        };
        Result=sqrt(Result);
        return Result;}
};
```

Figure 19-6: Complete listing of the RMS function written in Microsoft Visual C++.

Copy the file RMS.dlf file into the EES Userlib folder and start EES. The RMS function should automatically load. You can confirm that the function has been loaded by using the Function Information command from the Options menu and clicking on the External Routines button. You should see RMS in the list of routines, as shown in Figure 19-3. Click on RMS and the example of the call to the function should appear in the Example box at the bottom of the dialog. The function is called from EES as shown in Figure 19-4.

19.2 EES External Procedures - Type 1 (.dlp files)

External procedures are very similar to external functions. The only difference is that procedures can return one or more calculated values whereas functions return a single calculated result. External procedures are accessed from EES with the Call statement which has the following format:

```
Call ProcName ('text', A, B, … : X, Y, ...)
```

where ProcName is the name of the procedure, 'text' is an (optional) text string that will be passed to the procedure. This text can either be a string constant enclosed in single quote marks or a string variable. The input text is the only string input allowed in an external procedure. The input list A, B, ... can consist of one or more numerical inputs separated by a list separator (comma for the U.S. system, semicolon for the European system) appearing to the left of the colon. Inputs may be numerical constants, EES variable names, or algebraic expressions. The output list X, Y, ... are outputs determined by the procedure. There should be one or more outputs to the right of the colon, separated by list separators. Outputs must be EES numerical variable names. String variables are not allowed. Note that the Call statement used to access external functions is similar in format to the Call statement used for internal EES procedures, which are described in Chapter 3.

There are two formats for external procedures, referred to as Type 1 and Type2. The Type 1 format is recognized by having a .dlp file name extension. It uses a linked list for the input arguments and a second linked list for calculated outputs with the following format.

```
Procedure MyProc(S, Mode, Inputs, Outputs)
```

where S is a 256 ANSI character string, Mode is a 4-byte integer that is used for both input and output, as described below. The parameter Inputs is a pointer to the head of a linked list of input values and the parameter Outputs is a pointer to the head of a linked list of output values.

The linked lists are most conveniently programmed in Pascal or C++. The Type 2 format, identified with a .fdl file name extension, uses arrays rather than linked lists and therefore it can be written in Fortran, Pascal, C++, or any compiled language that can write a dynamic linked library file. This section describes the Type 1 procedure format and shows examples in Pascal and C++. Section 19.3 describes the Type 2 procedure format.

An external procedure must be written so that it accepts one or more input arguments that are used to calculate one or more output arguments. The parameters are passed by reference with the STDCALL protocol in the Windows Operating System. An external procedure should also internally provide an example of it how it is to be called and check that the number of inputs and outputs supplied in the linked lists are equal to the number that the procedure expects. To demonstrate the process, a simple procedure will be written in Pascal and C++. This procedure will be called MDAS (for Multiply-Divide-Add-Subtract). The procedure will accept two inputs (X and Y) and it will return four outputs (M = X*Y, D = X/Y, A = X+Y, and S = X-Y).

Type 1 External Procedures written in Pascal

A skeleton listing of an external procedure written in Embarcadero's Delphi XE2 compiler is shown in Figure 19-7.

```
library MyProc;
{$E .DLP}
type
   charstring=array[0..255] of ansichar;
   ParamRecPtr = ^ParamRec;
   ParamRec = record  { defines structure of the linked list of inputs }
     Value: double;
     next: ParamRecPtr;
   end;

procedure P(var S:charstring; Mode:integer; Inputs, Outputs :ParamRecPtr); export;
stdCall;
  begin
     Mode=0;
     {Get input values from Input linked list}
     {Set output values in Output linked list}
  end;

exports P name 'MyProc';

begin
end.
```

Figure 19-7: Skeleton listing of an external procedure for EES written in Delphi 2010.

The code begins with the Delphi keyword 'library', which instructs the compiler to compile the code as a dynamic link library (DLL) file. The project file name (MyProc in Figure 19-7) follows the library keyword; the project has a .dproj file name extension. The {$E .DLP} directive instructs the compiler to save the compiler library file with the project file name and a .dlp file name extension, i.e., MyProc.dlp. This is the name that EES will use to access the function

A pointer to a data structure called ParamRec is defined. This structure provides the linked list of inputs and outputs that are passed between the external procedure and EES. Each item in the linked list consists of a double precision value and a pointer to the next link. The last link must point to a null record (nil). EES will construct linked lists for the inputs and outputs when the procedure is called and pass the addresses of the pointers.

The procedure (named P in Figure 19-7) must have the four arguments provided in the order that is shown. The 'export' keyword directs the Delphi linker to make the function accessible to other programs. The 'stdcall' keyword is required to ensure that the arguments are passed to and from EES in the order that EES expects. Note that the exported procedure is renamed to be 'MyProc' using the 'exports' keyword. This renaming is necessary as Delphi does not allow a procedure to have the same name as the project, and it is the project name that is assigned to the compiled function.

The parameter S is a 256-character ANSI C string terminated with an ASCII 0; the parameter S can be used for both input and output. If the first parameter provided in the EES function is a string (within single quotes), EES will pass this string to the external routine. If an error is

encountered, S should be set in the external routine to an appropriate error message. If the length of S is not zero then EES will terminate calculations and display S in an error message.

The parameter Mode is an integer set by EES for input and it is set by the procedure for output. If Mode = –1, during input, then EES is requesting that the procedure return in S an example of the function call. Otherwise, the function should simply return the calculated output values. For output, the procedure should return Mode = 0 for normal operation. If Mode is set to a value other than zero (and S is the null string) then EES will terminate calculations and present an error message indicating the value of Mode.

The parameter Inputs is a pointer to the head of a linked list of input values supplied by EES. The procedure should count the inputs to be sure that the number supplied is as expected and issue an error message in S if this is not the case. Outputs is a pointer to the head of linked list of output values that are calculated and returned by the procedure.

Figure 19-8 provides a complete listing of a external procedure named MDAS written in Delphi XE2 that implements the MDAS (Multiply-Divide-Add-Subtract) procedure. Use the Build command from the Project menu in Delphi XE2 to create file MDAS.dlp. Copy this file into the EES Userlib folder and start EES to test it. Alternatively you can load the external procedure using the Load Library command from the Files menu.

```
library MDAS;
uses  SysUtils, Classes;
{$E .DLP}

const doExample = -1;
type
  CharString = array[0..255] of ansichar;
  ParamRecPtr=^ParamRec;
  ParamRec=record {structure of linked list}
    Value:double;
    next:ParamRecPtr;
  end;

 function CountValues (P: ParamRecPtr): integer;
  var N: integer;
  begin
    N := 0;
    while (P <> nil) do begin
      N := N + 1;
      P := P^.next
     end;
    CountValues := N;
  end; {CountValues}

procedure MDAS_p(var S:CharString; Mode:integer; Inputs,
                   Outputs:ParamRecPtr); export; stdcall;
  var N:integer;
      P:ParamRecPtr;
      X,Y:double;
  begin
    if (Mode = doExample) then begin
      S := 'Call MDAS(X, Y: M, D, A, S)';
      exit;
    end;
```

```
    N:=CountValues(Inputs);
    if (N<>2) then begin
      S := 'The number of inputs must be 2 for MDAS.';
      exit;
    end;
    N:=CountValues(Outputs);
    if (N<>4) then begin
      S := 'The number of outputs must be 4 for MDAS.';
      exit;
    end;
    P:=Inputs;
    X:=P^.Value;    P:=P^.next;
    Y:=P^.Value;
    if (Y<>0) then begin
      S:='Attempt to divide by zero in MDAS';
      exit;
    end;
    P:=Outputs;
    P^.Value:=X*Y;    P:=P^.next;
    P^.Value:=X/Y;    P:=P^.next;
    P^.value:=X+Y;    P:=P^.next;
    P^.Value:=X-Y;
  end; {MDAS_P}

exports
   MDAS_p name 'MDAS';
begin
  {no initiation code needed}
end.
```

Figure 19-8: Complete listing of the MDAS procedure written in Delphi XE2.

Figure 19-9 shows a short EES program that calls the procedure MDAS and the resulting Solution Window.

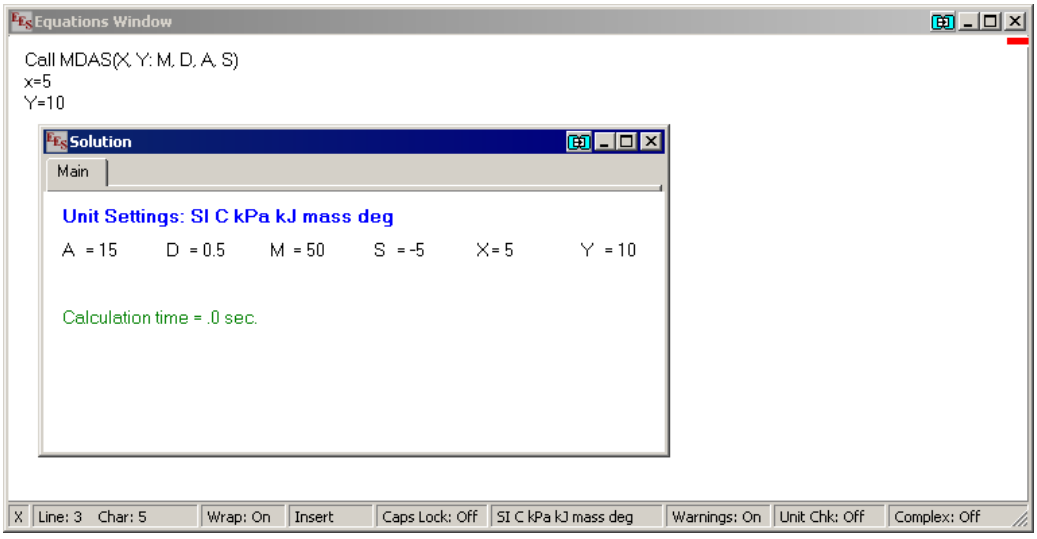


Figure 19-9: Example EES program that calls external procedure MDAS and the resulting Solution Window.

Type 2 External Procedures written in C++

A skeleton listing of an external procedure written in Microsoft's Visual C++ compiler, is shown in Figure 19-10.

```
#include <windows.h>
#include <stdlib.h>
#include <math.h>

BOOL APIENTRY DllMain(HANDLE hModule,DWORD ul_reason_for_call,LPVOID lpReserved)
    {return TRUE; }

const int doExample=-1;
struct ParamRec {  // Structure for handling EES calling syntax
  double value;
  struct ParamRec *next;};

extern "C" // Use the "C" style calling convention to avoid C++ mangled names

{__declspec(dllexport) void RMS(char s[256],int mode, struct ParamRec *Inputs, struct
            ParamRec *Outputs)
  {
   P=Inputs;
   Input1=P->value;  P=P->next; //Get Inputs…
   …
   P=Outputs;
   P->value=Output1; //Set Outputs;
   …
  }
};
```

Figure 19-10: Skeleton listing of an external function for EES written in C++.

The project file starts by including the required libraries. The APIENTRY system call is needed to specify the entry point for the dynamic link library file. The constant doExample is set and the linked list ParamRec structure that will be used to pass inputs and outputs between EES and the library file is defined. The structure consists of a value and a pointer to the next value. The last value is the null pointer, or 0.

The `extern "C"` statement is required for every function declaration having a C header to ensure that the function header is not "mangled", i.e., modified by the compiler. The `__declspec(dllexport)` keyword instructs the compiler to export the procedure with the required STDCALL protocol and eliminates the need for a .def file. Note that the procedure must have four arguments. The first argument (S) is a 256 array of ANSI characters that will be primarily used for output of a example format or error messages. The second argument (mode) is an integer. If mode = -1, the procedure is expected to return a string with an example call. All other values of mode indicate that the procedure should return calculated output values. The third argument, Inputs, is a pointer to the first link of the linked list of inputs. The procedure is responsible for calculating outputs, which are placed in the Outputs linked list

A complete listing of the MDAS procedure written in Microsoft Visual C++ is shown in Figure 19-11. This program should be compiled and linked as a Win32 Dynamic-Link Library with name MDAS.dlp.

```
#include <windows.h>
#include <stdlib.h>

// Defines a stdcall entry point for the dll
BOOL WINAPI DllMain(HANDLE hModule,DWORD  ul_reason_for_call,LPVOID lpReserved)
{ return TRUE; }

const int doExample=-1;
struct ParamRec {  // Structure for handling EES calling syntax
  double value;
  struct ParamRec *next;};

extern "C"// Use the "C" style calling convention to avoid C++ mangled names
int Count(struct ParamRec *Inputs)
{
       int N=0;
       ParamRec *P=Inputs;
       while (P  = 0) //Count Inputs
         {
                P=P->next;
                N++;
         };
       return N;
}

extern "C"
{__declspec(dllexport) void MDAS(char s[256], int mode, struct ParamRec *Inputs,
              struct ParamRec *Outputs)
       {double X,Y;
       int NInputs, NOutputs;
       ParamRec *P;
       if (mode==doExample) {
              strcpy(s,"Call MDAS(X, Y : M, D, A, S)");
              return;
       }
       NInputs= Count(Inputs);
       if (NInputs =2) {
              strcpy(s,"The number of inputs provided to MDAS must be 2");
              return;}
       P = Inputs;
       X=P->value; P=P->next;
       Y=P->value;
       NOutputs= Count(Outputs);
       if (NOutputs =4) {
              strcpy(s,"The number of outputs provided to MDAS must be 4");
              return;}
       P = Outputs;
       mode=0;
       P->value=X*Y; P=P->next;
       P->value=X/Y; P=P->next;
       P->value=X+Y; P=P->next;
       P->value=X-Y;
       }
};
```

Figure 19-11: Complete listing of the MDAS procedure written in Microsoft Visual C++.

Figure 19-9 shows a short EES program that calls procedure MDAS and the resulting Solution Window.

19.3 EES External Procedures - Type 2 (.fdl files)

External procedures can also be written using arrays for the inputs and outputs instead of linked lists. These external procedures are referred to as Type 2 procedures and they are identified with an .fdl file name extension. Type 1 and Type 2 external procedures are called from EES in exactly the same manner.

Type 2 procedures can be written with any compiler that can write a dynamic link library for the Windows Operating System with the STDCALL protocol, including Fortran 77 (or later). However, because the procedure is called from a program (EES) that is written in a different language, it is necessary to provide the compiler and linker with instructions about how to pass the arguments of the procedure. The required instructions vary depending on the Fortran compiler that is used. Instructions are provided here for the Intel Visual Fortran and the MinGW Fortran compilers.

Type 2 (.fdl procedure) with the Intel Visual Fortran Compiler

Shown in Figure 19-12 is a listing of the MDAS routine written as a Type 2 external procedure with the Intel Visual Fortran compiler.

```
    SUBROUTINE MDAS(S,MODE,NINPUTS,INPUTS,NOUTPUTS,OUTPUTS)
          DEC$ATTRIBUTES ALIAS:'MDAS' :: MDAS
          DEC$ATTRIBUTES DLLEXPORT :: MDAS
       IMPLICIT NONE
       INTEGER(4) MODE, NINPUTS, NOUTPUTS
       REAL(8) INPUTS(50), OUTPUTS(50), X, Y
       CHARACTER(255) S
       IF (MODE.EQ.-1) THEN
          S='CALL MDAS(X,Y:A,B,C,D)'C
          RETURN
       ENDIF
       IF (NINPUTS.NE.2) THEN
          S='MDAS requires two input.'C
          RETURN
       ENDIF
       IF (NOUTPUTS.NE.4) THEN
          S='MDAS EXPECTS TO PROVIDE 4 OUTPUTS'C
          RETURN
       ENDIF
       X=INPUTS(1)
       Y=INPUTS(2)
       IF (ABS(Y).LE.1E-9) THEN
         S='DIVISION BY ZERO IN MDAS'C
         RETURN
       ENDIF
       OUTPUTS(1)=X*Y
       OUTPUTS(2)=X/Y
       OUTPUTS(3)=X+Y
       OUTPUTS(4)=X-Y
       S=''C
       RETURN
     END
```

Figure 19-12: Listing of the MDAS procedure written with the Intel Visual Fortran compiler.

The Subroutine header statement must have the six arguments shown in Figure 19-12. Parameter S is a null-terminated C-style character string containing 255 characters. If the first parameter in

the EES Call statement is a text constant (within single quotes) or text variable (ending with a $) then EES will pass this string to the external program in the parameter S. Any error messages generated by the procedure are returned to EES in the parameter S. Note the use of the terminating character C in the error message strings shown in Figure 19-12, which ensures that the strings are terminated by an ANSI null character. The parameter S should be set to ''C for normal operation.

The parameter Mode is a 4-byte integer. When EES calls the subroutine with Mode = -1, it is instructing the external procedure to place an example of the calling sequence for this procedure in the parameter S so that it can be displayed in the Function Info Dialog window. The parameter S is also used to return user-supplied error messages, if necessary. If an error is detected in the subroutine, then Mode should be set to a value that is greater than 0 to signal that EES should terminate calculations. If S is defined, it will be displayed in the EES error message. During normal operation, Mode is zero and should not be redefined.

The parameters NINPUTS and NOUTPUTS are 4-byte integers that hold the number of inputs and outputs, respectively. Values for the inputs are provided by EES based on information in the Call statement issued from EES. The procedure should check to see if NINPUTS and NOUTPUTS agree with the expected number of inputs and outputs and return an error condition, if needed, by setting Mode > 0 or by setting S to an error message, as shown in Figure 19-12. If Mode > 0 or if S is not the null string then EES will display an error message and then terminate.

The parameters INPUTS and OUTPUTS are arrays of double precision (REAL(8)) values. EES will allocate memory for these arrays as needed. Up to 1000 inputs and 1000 outputs can be passed in the argument list of external functions and procedures using array element notation, i.e., X[1..1000]. EES will supply the values in the INPUTS array. Results calculated by the external procedure are placed in the parameter OUTPUTS and returned to EES.

The external program must be compiled and linked as a Dynamic Link Library (DLL) routine. The compiling procedure differs for different languages and compilers. It is essential to set the compiler and linker options properly in order for EES and the operating system to recognize and use the library file. The Intel Visual Fortran compiler requires the /iface:mixed_str_len_arg and /iface:cvf compiler options, as shown in Figure 19-13. The /iface:mixed_str_len_arg compiler option instructs the compiler to include the string length (for string S) as a hidden argument directly after the string argument. The /iface:cvf option instructs the compiler to use the STDCALL format and to pass the arguments by reference (rather than by value); it further instructs the Fortran compiler to export the name of the subroutine in uppercase letters. On a 32-bit system, this option also results in @N being appended to the procedure name, where N = 32 is the number of bytes for the pointers to the arguments. EES will not be able to identify the procedure with the @N appended. Consequently, the first compiler directive beginning with DEC$ in Figure 19-12 is required to force the compiler to rename the procedure according to the name provided in quotes, which will not have the @N appended. The second compiler directive in Figure 19-12 instructs the compiler to make the procedure visible as a dynamic link library.

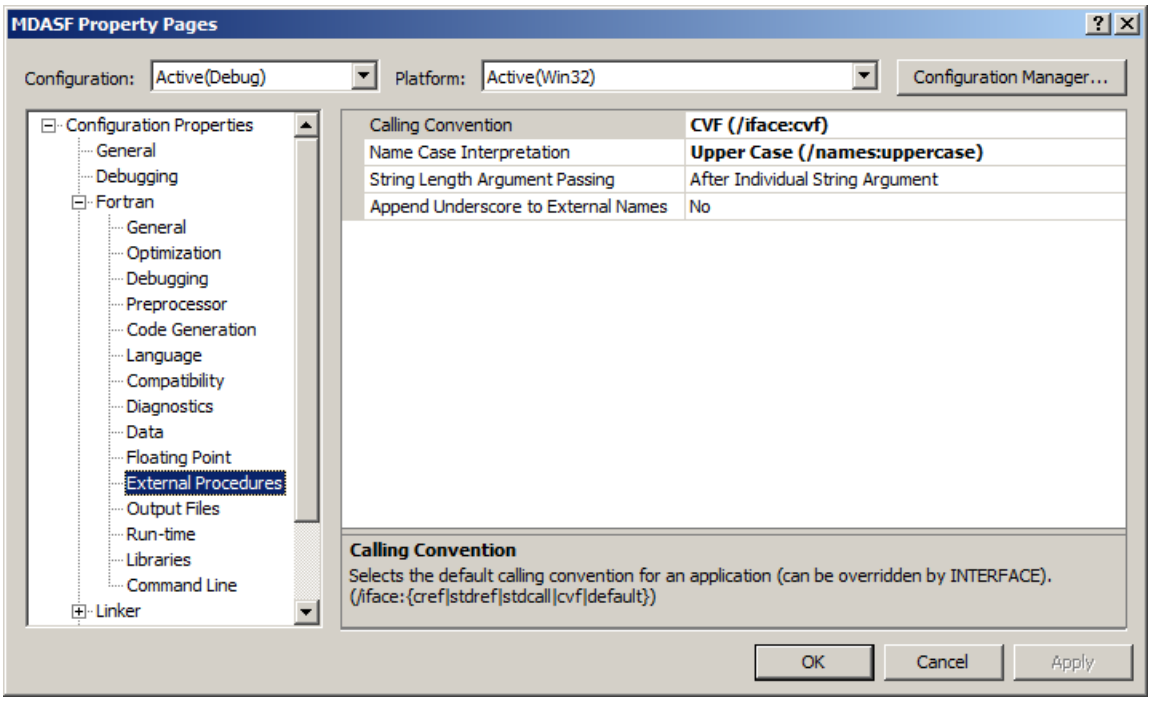


Figure 19-13: External Procedures Properties for the Intel Visual Fortran Compiler.

It is also necessary to specify the runtime library that will be included with the compiled library file. Even though the compiler is creating a DLL, you should not select the Multithread DLL runtime library. Instead, select the Multithreaded Runtime library, as shown in Figure 19-14. The Multithread DLL library will not work on some systems and it is not needed for the DLLs that are built for use with EES.

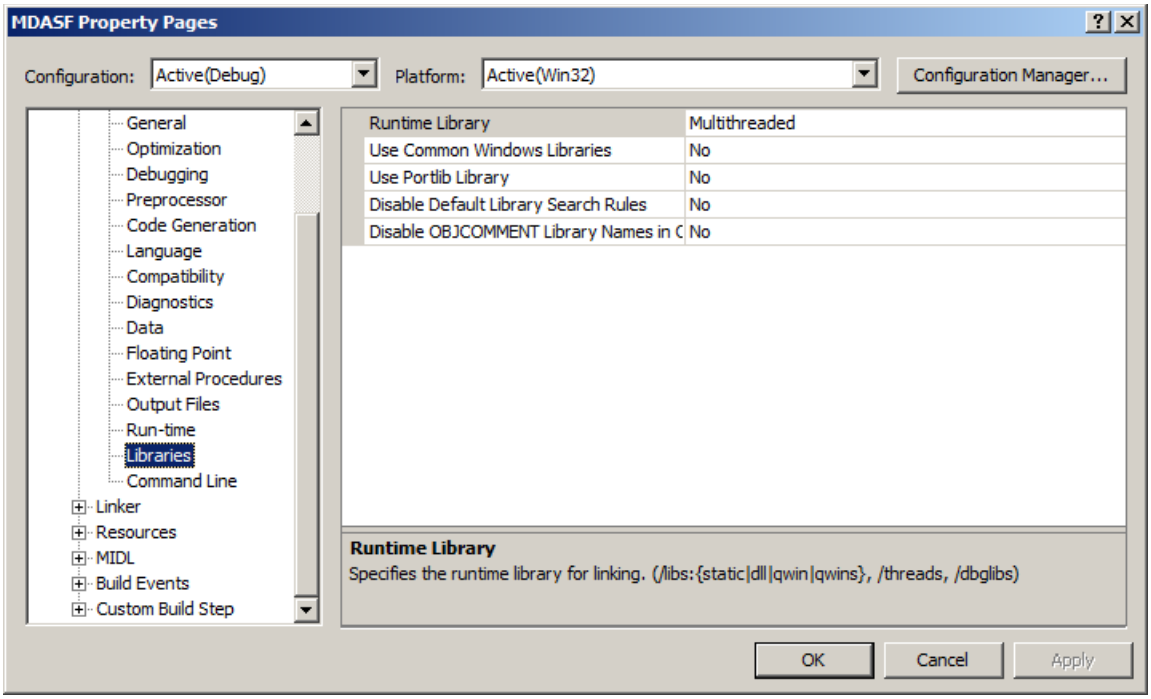


Figure 19-14: Runtime Library Properties for the Intel Visual Fortran Compiler.

By default, the file name extension of a dynamic link library is .dll. This extension must be changed to .fdl in order for EES to recognize the Type 2 procedure format. The change can be made by changing the output file name template provided for the linker.

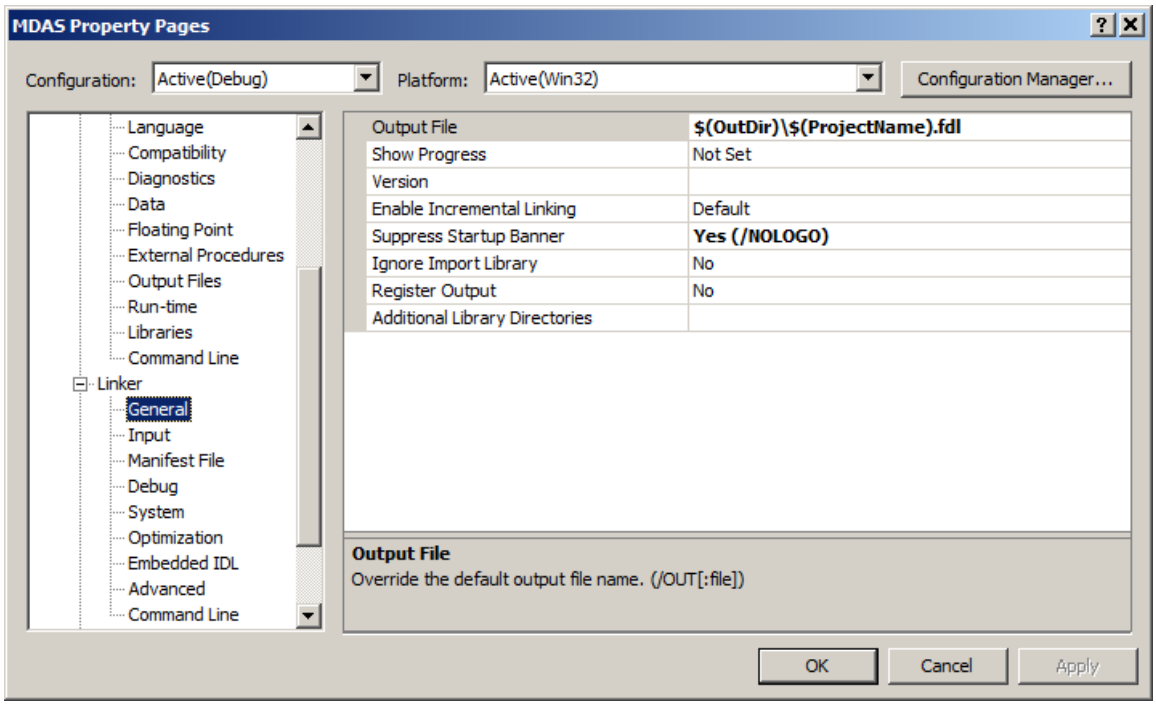


Figure 19- 15: Linker Properties showing the output file name extention set to .fdl.

Type 2 (.fdl procedure) with the MinGW Open Source GCC Fortran Compiler

MinGW is a development environment for Microsoft Windows applications that provides an Open Source programming tool set, including the GNU Compiler Collection (GCC). This compiler collection includes a Fortran compiler that can be used to generate Type 2 (.fdl) external procedures. This section provides summary data for downloading, installing and using the GCC Fortran compiler for creating .fdl external procedures. For more information please visit http://www.mingw.org.

To install MinGW (Minimalist GNU for Windows), download the latest version by clicking on Download file mingw-get-inst link-20110802 from http://sourceforge.net/projects/mingw/files/. Double-click on the file icon to start the installer. It is recommended that you select the option to download the latest repository catalogues and that you do not change the default path (C:MinGW) for the compiler installation. It is necessary to select both the Fortran Compiler and the MSYS Basic System as shown in Figure 19-16. These options are not selected by default. Click Next to complete the installation process.

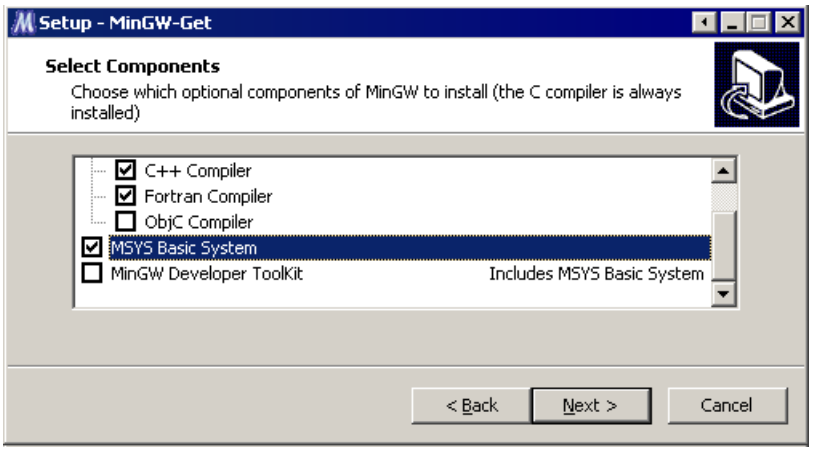


Figure 19-16: Screen shot from the MinGW Installation program showing selected options.

The Fortran compiler is accessed with the MinGW Shell command line interpreter, which is similar to the Microsoft DOS command line interpreter. Start the command shell by selecting MinGW Shell from the Start menu in the MinGW folder, as shown in Figure 19-17.

Figure 19-17: Starting the MinGW Shell from the Start menu.

The most important MinGW Shell commands to know are:

ls	list files in the current directory (comparable to dir)
pwd	print the current directory
cd	change directory, note that the MinGW shell uses forward (rather than back) slashes as delimiters.

The MinGW Shell opens in a fake home directory. Navigate to the directory containing your source code using the cd command. Surround your directory name with single quote marks if it contains spaces. For example, to change to the "my FDL" folder on drive C, enter:

```
cd '/c/my FDL/'
```

The Fortran source file is a text file that can be generated in any text editor, such as NotePad. A listing of the MDAS Fortran source code, as modified for MinGW Fortran, appears in Figure 19-18.

```
    SUBROUTINE MDAS(S,LS,MODE,NINPUTS,INPUTS,NOUTPUTS,OUTPUTS)
     USE iso_c_binding, only: c_null_char !necessary to append a null character
       IMPLICIT NONE
        INTEGER(4) MODE, NINPUTS, NOUTPUTS, LS
       REAL(8) INPUTS(50), OUTPUTS(50), X, Y
       CHARACTER(255) S
       IF (MODE.EQ.-1) THEN
          S='CALL MDAS(X,Y:A,B,C,D)'//C_null_char
          RETURN
       ENDIF
       IF (NINPUTS.NE.2) THEN
          S='MDAS requires two input.'//C_null_char
          RETURN
       ENDIF
       IF (NOUTPUTS.NE.4) THEN
          S='MDAS EXPECTS TO PROVIDE 4 OUTPUTS'//C_null_char
          RETURN
       ENDIF
! Do calculations
       X=INPUTS(1)
       Y=INPUTS(2)
       IF (ABS(Y).LE.1E-9) THEN
         S='DIVISION BY ZERO IN MDAS'//C_null_char
         RETURN
       ENDIF
       OUTPUTS(1)=X*Y
       OUTPUTS(2)=X/Y
       OUTPUTS(3)=X+Y
       OUTPUTS(4)=X-Y
       S=''//C_null_char
       RETURN
     END
```

Figure 19-18: MDAS Fortran source code (MDAS.f90) for the MinGW Fortran compiler.

The MinGW Fortran compiler handles strings differently than the Intel Visual Fortran compiler, and this difference requires a minor change in the Fortran source code. Specifically, an additional integer variable (LS) must be added after the string argument (S) in the Subroutine statement. In addition, all strings must be terminated with the C_null_char, which is defined in the iso_c_binding library. A Use command is required to access this library.

A definition file (here called mdas.def) is required by the linker in order for EES and the compiled .FDL procedure to communicate successfully. The contents of the definition file are:

```
EXPORTS
mdas_@32 @ 1
MDAS = mdas_@32
```

The .def file should be placed in the same directory as the .f90 source code. Note that mdas on the second and third lines of the .def file is the name of the subroutine that appears in the source code; however, the name should be all lower case in the .def file. If the subroutine name is changed then these lines should be changed accordingly. (The _@32 characters that are appended to the subroutine name are used by the compiler to provide information relating to the type and number of parameters.)

To create the .fdl file, enter the following command on one line into the MinGW Shell while the directory is set with the cd command to the directory containing the source code and definition file:

```
gfortran -ffree-line-length-0 -shared -mrtd -Wl,--enable-stdcall-fixup -
static –o mdas.FDL mdas.f90 mdas.def
```

Note that the -Wl,--enable-stdcall-fixup parameter is required in Windows 7 (capital W is followed by lowercase l and there are two dashes in front of enable). You can move the .fdl file to the EES Userlib folder from within the MinGW shell so that it is automatically loaded when EES is started using the following command. (This command assumes that EES is located in the C:\EES32\ directory.)

```
mv MDAS.FDL /c/EES32/Userlib/
```

19.4 Multiple Files in a Single Library File (.dll)

EES recognizes three different types of externally compiled files. The three types are:

.dlf - dynamically-linked function
.dlp - dynamically-linked procedure, Type 1
.fdl - dynamically-linked procedure, Type 2

Only one external function or procedure will be recognized by EES in these file types. The filename extension (.dlf, .dlp, or .fdl) identifies the type of externally compiled file and the name of the external routine must be the same name as filename (without the extension).

However, it is also possible to place one or more external routines in a single file and this single file can contain one or more external programs of all three types of external routines. The external file can have any name but it must have a .dll filename extension. If this file is placed in the UserLib subdirectory then EES will automatically load all the external routines in the file at startup, provided that the Library Manager (See Chapter 11) does not direct otherwise.

EES must know the names of all of the external routines in the .dll file and their type (.dlf, .dlp, or .fdl) because the calling arguments differ for each type. The mechanism for telling EES the names and types of the external routines is to provide three short routines in the DLL file with names DLFNames, DLPNames, and FDLNames. These routines do nothing but return the calling names of each routine type in the DLL file. DLFName, DLPNames, and FDLNames must be exported in the DLL. They have one argument, which is a character string. The character string is filled with the names of the routines of each type that are contained in the .dll file. A comma separates each file name. A zero length string is used to indicate that there are no files of that type. A skeleton example is provided in Figure 19-19 for DELPHI XE2 code and in Figure 19-20 for Visual C++ code.

```
library MYEXTRNLS  {This DLL file contains two DLF functions and one DLP procedure}
uses  SysUtils;
const  doExample = -1;
{*****************************************************************}
```

```
type
  CharString = array[0..255] of ansichar;
  ParamRecPtr = ^ParamRec;
  ParamRec = record
               Value : Double;
               Next  : ParamRecPtr;
  end;
{*********************************************************************}
{There are two .DLF functions; names are separated with commas}

procedure DLFNames(Names : PAnsiChar); export; stdcall;
  begin
    StrCopy(Names,'myFunc1, myFunc2');
  end;

{*********************************************************************}
{There is one .DLP procedure}
procedure DLPNames(Names : PAnsiChar); export; stdcall;
  begin
    StrCopy(Names,'myDLP');
  end;

{*********************************************************************}
{no FDL procedures so return a null string}
procedure FDLNames(Names : PAnsiChar); export; stdcall;
  begin
    StrCopy(Names,'');
  end;

{*********************************************************************}
function myFunc1 (var S: CharString; Mode: integer;
       Inputs: ParamRecPtr): double; export; stdCall;
  begin
      {Code for myFunc1}
      ...
  end; {myFunc1}

{*********************************************************************}
function myFunc2 (var S: CharString; Mode: integer;
       Inputs: ParamRecPtr): double; export; stdCall;
  begin
      {Code for myFunc2}
      ...
  end; {myFunc2}

procedure myDLP(var S:CharString; Mode:integer;
        Inputs,Outputs:ParamRecPtr); export; stdCall;
  begin
      {Code for myDLP}
      ...
  end; {myDLP}

{*********************************************************************}
 exports
   DLFNames,
   DLPNames,
   FDLNames,
   myFunc1,
   myFunc2,
   myDLP;

begin
end.
```

Figure 19-19: Skeleton listing of a .DLL file in Delphi 2010 code that holds several EES external library files.

```
#include <windows.h>
#include <math.h>

// Defines the entry point for the dll
BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved )
{ return TRUE;}

// Tell EES which functions and procedures are exported in the library :
// List of DLF format functions
__declspec(dllexport) void DLFNames(char* Names)
{ strcpy(Names,"myFunc1,myFunc2"); }

// List of DLP format procedures
__declspec(dllexport) void DLPNames(char* Names)
{ strcpy(Names,"myDLP"); }

// List of FDL format procedures
__declspec(dllexport) void FDLNames(char* Names)
{ strcpy(Names,"myFDL");}

// DLF functions implementation
__declspec(dllexport) double myFunc1(char s[256], int mode, EES_PARAM_REC *rec_in)
{Code for myFunc1}
    ...
          return;
}
__declspec(dllexport) double myFunc2(char s[256], int mode, EES_PARAM_REC *rec_in)
{Code for myFunc2}
    ...
          return;
}
// DLP procedure implementation
__declspec(dllexport) void myDLP(char s[256], int mode, EES_PARAM_REC *rec_in,
EES_PARAM_REC *rec_out)
{Code for myDLP}
    ...
           return;
          }
// FDL procedures implementation (FORTRAN style)
// Note: Fortran always passes the length of the string after a string parameter.
// FORTRAN is expects parameters to be passed by reference
//
__declspec(dllexport) void myFDL(char s[256], int& clen, int& mode,
          int& NInputs, double inputs[25], int& NOutputs, double outputs[25])
{
{Code for myFDL}
    ...
return;
}
```

Figure 19-20: Skeleton listing of a .DLL file in Visual C++ code that holds several EES external library files.

19.5 Managing External Library Files

External library files are handled by EES in the same manner as the internal library files described in Chapter 11. The only significant difference is that EES cannot provide unit checking for external library files, as it does for the internal library files that are written in EES code. A summary of topics related to managing the library files is provided here.

Loading External Library Files

External library files can have a .dlf, .dlp, .fdl or .dll file name extension. These files will be automatically loaded when EES is started if they are placed anywhere within the Userlib folder.

Note that the automatic loading of both external and internal library files can be controlled in the Professional version using the Library Manager, which is discussed in Section 11.6.

Library files can be manually loaded using the Load Library command from the File menu or by placing a $Include "FileName" directive in the EES equations window. The filename should be enclosed within quotes, as shown, and it must include the file name extension. It is not normally necessary to include the directory name with the file name, as EES will look in the directory that the EES file was loaded from as well as the EES directory if directory information is not provided.

Providing Help for External Library Files

All external library files that have been loaded are displayed in the Function Information dialog, as shown in Figure 19-3. Clicking the Function Info button in this dialog will display help from an external file for the selected library file. The developer of an external function should provide a help file for each external library file that has the same parent name as the library file. The help file can be a .pdf, .htm, or .chm file. The help files for external library files are prepared in the same manner as those for internal library files, which are described in Section 11.3.

20 THE REPORT WINDOW

The Report Window operates just like a typical word processor, offering the usual choice of fonts, font sizes, text style, color, highlighting, tabs, indentation, etc. Graphics can be pasted into the window at the cursor position. The graphics can be equations copied from the Formatted Equations window or from MathType, a plot from the Plot Window, or any graphics in the Diagram Window. Graphics can also be pasted from other applications. The important difference between the Report Window and a standard word processor application, such as Microsoft Word, is that the Report Window can include EES variables that are automatically updated as their values change. The Report Window can therefore generate a nicely formatted document that automatically contains the calculated quantities from your EES program. This chapter demonstrates how the Report Window operates.

20.1 Creating the Report

It is necessary to have a working EES program in order to demonstrate the capabilities of the Report Window. For this purpose, we will use the Refrig.ees example program. This program file can be found in the UserLib\Examples folder within the folder with the EES application. You can also open this file using the Examples menu at the right of the menu bar by selecting Parametric Table and then clicking on the Refrigeration COP vs evaporator temperature line, as shown in Figure 20-1.

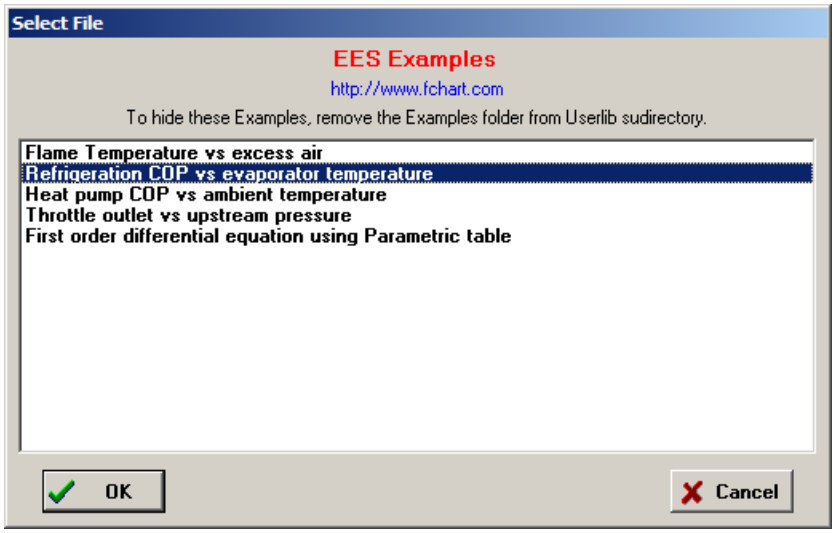


Figure 20-1: Selecting the Refrig.ees example from the Examples menu.

The file includes a Diagram Window with a Calculate button, as shown in Figure 20-2. Depending on the setting of the radio button control, clicking the Calculate button will either direct EES to solve the refrigeration cycle for an evaporator temperature of 10°C or solve the Parametric Table for a range of evaporator temperatures. Click the Calculate button for the evaporator temperature set to 10°C. The program will solve and update the Diagram Window.

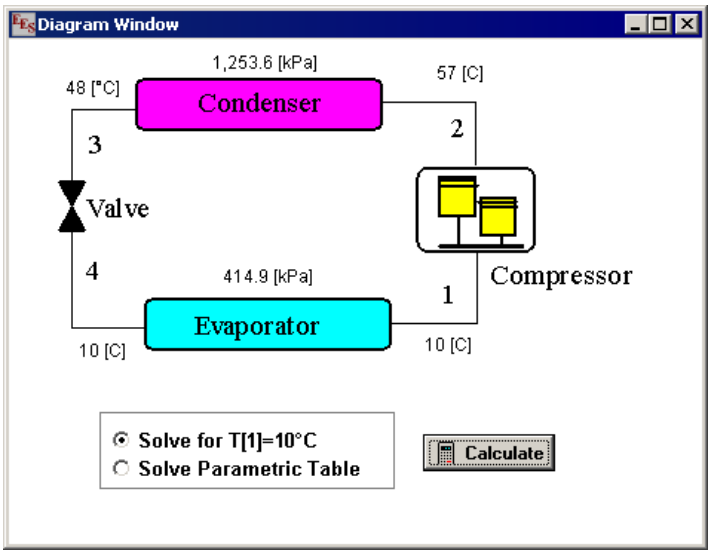


Figure 20-2: Diagram Window from example Refrig.ees.

Now we are ready to construct a Report Window. Select Report Window from the Windows menu or click the Report Window speed button ( ). Either action will bring the Report Window to the front. The Report Window appears as shown in Figure 20-3.

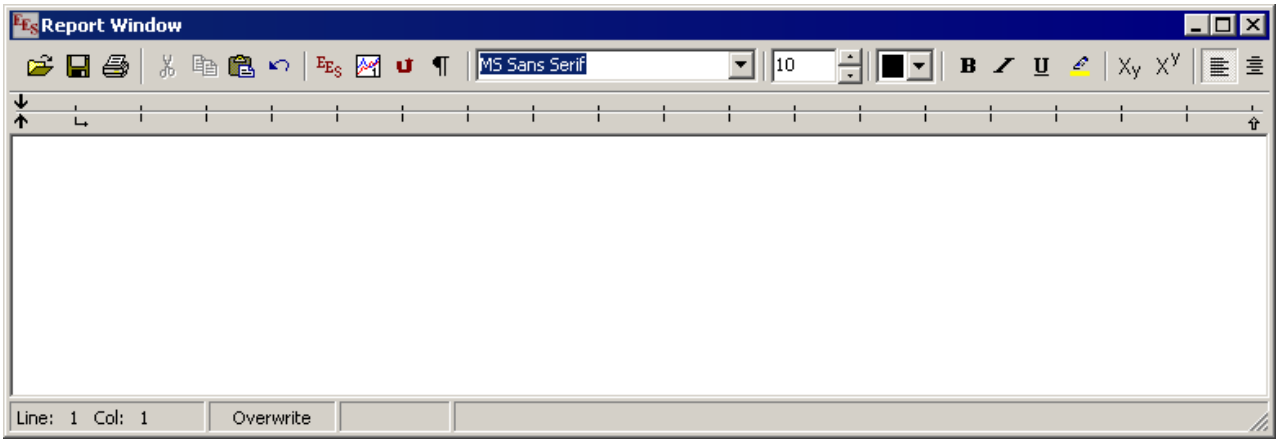


Figure 20-3: Report Window.

The Report Window operates in a manner that is similar to most word processors. Most of the controls will be familiar to you, but it is perhaps useful to formally identify them.

- Import (load) text into the Report Window from a .txt or .rtf file.
- Export (save) the contents of the Report Window to a file. If a .rtf file is selected, then the fonts, colors and graphics appearing in the Report Window will appear in the word processor application just as they appeared in the Report Window. However, the EES variables and plots will not automatically update in the word processor.
- Print the contents of the Report Window to the selected printer.
- Cut the selected text from the Report Window.

- Copy the selected text to the clipboard.
- Paste the contents of the clipboard into the Report Window at the current selection point.
- Undo the last operation in the Report Window.
- Insert an EES variable in the Report Window. See Section 20.2 for details.
- Insert an EES plot in the Report Window. See Section 20.3 for details.
- Update the Report Window so that all EES variables and plots are showing current results.
- Show hidden codes for embedded EES variables and plots. See Section 20.2.
- Arial Set the font for the selected text.
- 10 Set the font size for the selected text.
- Set the color of the selected text
- Set the selected text to have bold font style.
- Set the selected text to have italic font style.
- Set the selected text to have underline font style.
- Highlight or remove highlight from the selected text.
- Set the selected text to be a subscript.
- Set the selected text to be a superscript.
- Left justify the text in the current paragraph.
- Center justify the text in the current paragraph.
- Right justify the text in the current paragraph.

Controls on the ruler allow specification of the left and right margins and a hanging indent for each paragraph, as shown in Figure 20-4. A single tab stop can also be set. Additional tab stops can be specified by clicking the right mouse button anywhere in the Report Window. A pop-up menu will appear, as shown in Figure 20-5(a). The Tabs menu item will bring up a dialog in which additional tabs can be set, as shown in Figure 20-5(b).

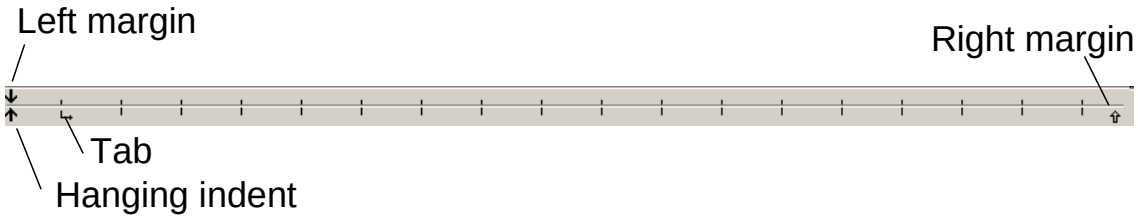
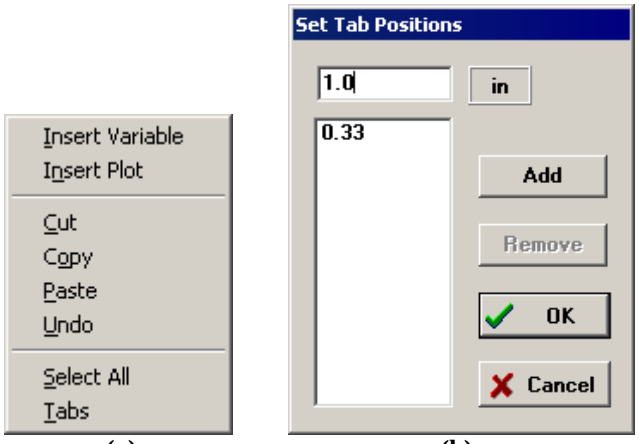


Figure 20-4: The Report Window ruler.



(a) **(b)**

Figure 20-5: (a) Pop-up menu appearing after right click, and (b) the Set Tabs dialog window.

Enter text into the Report Window as you would with any word processor. You can change the font, size, color and style of the text that you enter as you wish. You can also paste graphic items from other applications, such as Power Point or MathType, into the Report Window. Graphics objects can be copied from the EES Diagram Window, as shown in the prototype Report Window in Figure 20-6. Active hypertext can be entered into the Report Window. EES will automatically identify hypertext links that begin with http:\\, https:\\ or file:. In addition, a hypertext link that begins with \\EES_ following by the name of an EES window is understood to be a link to that window. Clicking on a link will move the focus to that location, opening another program if necessary. A summary of the recognized links is shown in Table 20-1. The Report Window shown in Figure 20-6 includes live EES variables and a live EES plot. The following sections describe how this information is inserted in the Report window.

Table 20-1: Recognized hypertext links in the Report Window.

Hyperlink	Description
http:\\fchart.com	Open the default browser and point it at the web page that follows http:
https:\\fchart.com	Open the default browser and point it at the web page that follows https:
file:C:\ees32\ees_manual.pdf	Open the file with the file name that follows file: and start the appropriate application.
\\EES_Solution	Open the Solution Window and bring it to the front
\\EES_Format	Open the Formatted Equations Window and bring it to the front
\\EES_Plot	Open the Plot Window and bring it to the front
\\EES_Parametric	Open the Parametric Table and bring it to the front
\\EES_Lookup	Open the Lookup Table and bring it to the front
\\EES_Array	Open the Array Table and bring it to the front
\\EES_Integral	Open the Integral Table and bring it to the front
\\EES_Equations	Open the Equations Window and bring it to the front
\\EES_Diagram	Open the Diagram Window and bring it to the front
\\EES_Residual	Open the Residuals Window and bring it to the front
\\EES_Calculator	Open the Calculator Window and bring it to the front
\\EES_Solve	Solve the set of equations in the Equations Window
\\EES_SolveTable	Apply the Solve Table command in the Calculate menu
\\EES_MinMax	Apply the Min/Max command in the Calculate menu
\\EES_MinMaxTable	Apply the Min/Max Table command in the Calculate menu

Report Window

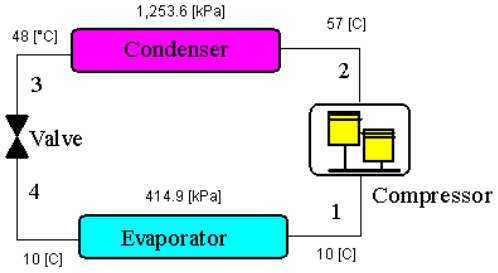
Office Memo:

This window shows some of capabilities of the Report Window that are provided in the Professional version.

You can paste in a letterhead or whatever graphics you wish. Equations can be copied from the Formatted Equations window or from MathType. EES variables and plots can be inserted into the Report in a manner in which they will be updated automatically. Most word processing capabilities are provided.

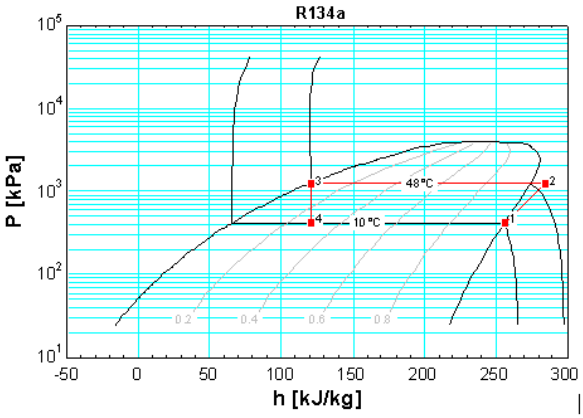
The Report Window is saved with other information in the EES file. It can also be saved separately using the Export button in the tool bar for this window. The Report Window can be printed with other EES information or independently by clicking on the print button in the tool bar.

Our team completed an analysis of a residential refrigeration system. A schematic of the system is shown below with some operating information. *(This figure was copied from the Diagram Window as a bitmap. It can be resized by selecting the figure and dragging the handles.)*



The COP is defined as the ratio of the refrigeration capacity to the compressor power input. When the evaporator temperature is 10 [C] and the condensing saturation temperature is 48 [C], then COP=4.742. *(Note that the COP, pressure and temperature information are from EES variables and they are changed as the variables change. To test this, change T[1] from 10 [C] to -10 [C] and solve. Then return to this window to see the results.)*

A pressure-enthalpy diagram for this system is shown below. *(The plot was inserted usign the Insert EES plot button on the menu tool bar. The plot in this window will automatically be updated when the contents of the plot window are changed.)*



Line: 28 Col: 22 Overwrite

Figure 20-6: Prototype Report window showing text fonts, colors, sizes and inserted EES variables and plot.

20.2 Inserting EES Variables

The major advantage of the Report Window is that EES variables and plots can be inserted into the report so that they always display the current calculated values. Click the mouse at the location where you wish to insert a variable. Then either click the Insert EES variable button ( ) on the menu bar or click the right mouse button and select the Insert Variable from the pop-up menu shown in Figure 20-5(a). Either action will bring up the Insert Variable dialog shown in Figure 20-7.

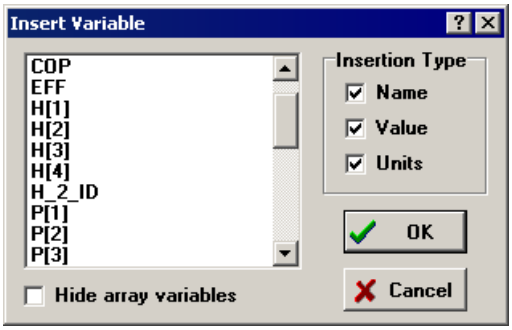


Figure 20-7: Insert Variable dialog.

The list in the Insert Variable dialog shows all of the EES variables that have been defined in the main part of the Equations Window. Click on the variable that you wish to insert and then use the check boxes to select whether you wish to show the name of the variable, its value, and its units. Click OK to insert the variable. For example, the text in the Report Window shown in Figure 20-6 contains a sentence that reads:

When the evaporator temperature is 10 [C] and the condensing saturation temperature is 48 [C], then COP=4.742.

The portions of the sentence: 10 [C], 48 [C] and COP=4.742 were inserted as EES variables and they will change as the values in the analysis change. To insert the 10 [C], select variable T[1] from the list and uncheck the Name check box. To insert COP=4.742, select the variable COP, check the Name and Value boxes, but uncheck the Units box.

To test the automatic update capability, go to the Equations Window and change the value of T[1] from 10 C to -10 C. Solve the problem and return to the Report Window by selecting Report Window from the Windows menu or by clicking the Report Window speed button ( ). The Report Window will now display:

When the evaporator temperature is -10 [C] and the condensing saturation temperature is 48 [C] COP=2.598.

Clicking on the inserted text will provide information in the status bar located at the bottom of the Report Window. For example, if you move the mouse to the location of the 48 [C] and click, the status bar will appear as shown in Figure 20-8. The 48 [C] is identified as being the value of variable T[3].

Figure 20-8: Report Window status bar.

EES uses hidden codes in the text to identify the inserted variables. The hidden codes can be made visible by clicking the Show Hidden Text button (¶) in the Report Window menu bar. When this button is clicked, the sentence shown above will appear as:

When the evaporator temperature is ##02_25_-10 [C]##99 and the condensing saturation temperature is ##02_32_48 [C]##99, then ##06_22_COP=2.598##99.

The code for an embedded EES variable begins with characters ##. Information that identifies the EES variables follows these characters. The code is terminated with the characters ##99. It is not necessary to understand the code in order to effectively use the Report Window, but the following information may be useful should you wish to change the display format quickly. The two digits following the leading ## characters provide the information about how EES should display the variable information. There are six alternatives, as listed in Table 20-2. For example, the evaporator temperature in variable T[1] is displayed with code 02 so only its value and units are visible. To change the display format for an existing EES variable, you need to change the code and then click the Show Hidden Text button (¶) to hide the codes. Click the Update button ( ) to refresh the display.

Table 20-2: Display code used for EES variables in the Report Window.

Code	Show Name	Show Value	Show Units
01	no	yes	no
02	no	yes	yes
03	yes	yes	yes
04	no	no	yes
05	yes	no	no
06	yes	yes	no

Following the code is an underscore character and then the internal number identifying the EES variable. For example, T[1] is identified as EES variable 25. This value cannot be changed.

If you wish to delete an inserted EES variable, it is best to do so while the codes are visible. Delete the codes as well as the variable information. Click the Show Hidden Text button again to return the normal display mode.

20.3 Inserting EES Plots

EES plots from the Plot Window can be copied and pasted into the Report Window. The plots displayed in the Report Window can optionally be set to change whenever the information in the Plot Window changes.

To insert a plot, click the mouse when the cursor is at the position that the plot should appear. Then either click the Insert Plot button ( ) or click the right mouse button to bring up the menu in Figure 20-5(a) and select the Insert Plot menu item. Either action will bring up the Select Plot dialog shown in Figure 20-9. Select the plot you wish to appear in the Report Window. The

default is Automatic Update, which will result in the plot shown in the Report Window changing whenever it is changed in the Plot Window. If you do not wish the plot to change, unselect the Automatic Update option. Click the OK button and the plot should appear in the Report Window.

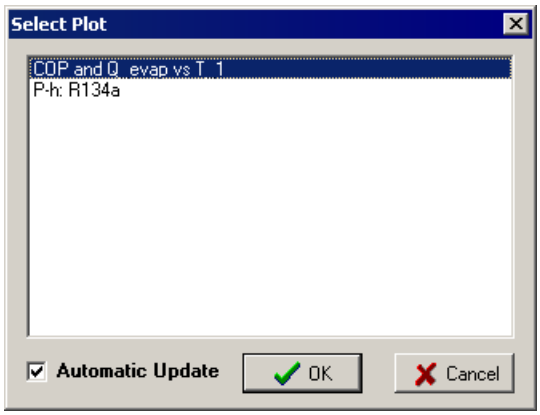


Figure 20-9: The Select Plot dialog

Note that the size of the plot that will be pasted into the Report Window depends on the size of the plot in the Plot Window. You can change the size of a plot or any other graphic in the Report Window by selecting it and using the handles that appear to resize it. However, the plot will return to its original size when it is automatically updated. You can control the size by setting the size of the plot in the Plot Window.

Hidden characters are used to identify a plot that automatically updates. These characters can be viewed by clicking the Show Hidden Text button (¶) in the Report Window menu bar. When this button is clicked, the following code will be visible just before the plot.

~~##97_P-h: R134a_##98~~

If the code is changed or deleted then the plot will no longer automatically update. If you delete the plot, you can also delete this code. These hidden characters are used to identify a plot that automatically updates.

Appendix A: Built-In Mathematical Functions

The mathematic functions that are built into EES can be grouped into the following categories. The functions are described in the tables indicated for each category.

- Basic Math functions (Table A-1)
- Bessel functions (Table A-2)
- Complex Math functions (
-
- Table **A-3**)
- Conditional or Logic functions (

Table A-4)
- Diagram Window Support functions (Table A-5)
- Hyperbolic functions (Table A-6)
- Integral functions (Table A-7)

Statistical functions (

- Table A-8)
- Table Data functions (

Table A-9)
- Trigonometric functions (Table A-13)
- Unit functions (Table A-14)

Table A-1: Basic Math functions.

Function	Description
Abs(X)	Returns the absolute value of the argument X. In complex mode, Abs returns the magnitude of complex variable X.
Ceil(X)	Returns the lowest integer greater than or equal to X. For example: Ceil(2.8)=3
Exp(X)	Returns the exponential of the value, i..e., the value of e raised to the power of the argument, X.
Factorial(X)	Returns the factorial of the value of the argument X rounded to the nearest integer. The factorial of zero or a negative number is returned as 1.
Floor(X)	Returns a value equal to the largest integer value that is less than or equal to the value of X. Note that Floor differs from Trunc when X is less than zero in that Floor rounds down rather than up. For example Trunc(-2.5) = -2 whereas Floor(-2.5) = -3.
Ln(X)	Returns the natural logarithm of argument X.
Log10(X)	Returns the base-10 logarithm of the argument X.
Mod(X1,X2)	Returns the modulus of X1 with respect to X2. The modulus is the remainder when X1 is divided by X2. If X2 is 0, an error will result.
Pi	This function returns the value of pi. It has no arguments.
Round(X)	Return a value equal to the nearest integer value of the argument X.
Sqrt(X)	Returns the square root of argument X, which must be ≥ 0.
Trunc(X)	Returns a value equal to the integer value corresponding to the argument rounded toward zero. See Floor(X).
UncertaintyOf(X)	Returns the assigned or calculated uncertainty of the variable X that is provided as the argument to the function. The returned value will be zero if uncertainty information has not been specified for the specified variable or if the calculations are not initiated with the Uncertainty Propagation or Uncertainty Propagation Table commands. See Chapter 8 for more details.

Table A-2: Bessel functions.

Function	Description
Besseli(j,X)	Returns the value of j^{th}-order modified Bessel function of the first kind for argument value X where X > 0. If j = 0, this function automatically calls the Bessel_i0 function.
Besselj(j,X)	Returns the value of j^{th}-order Bessel function of the first kind for argument value X where X > 0. If j = 0, this function automatically calls the Bessel_j0 function.
Besselk(j,X)	Returns the value of j^{th}-order modified Bessel function of the second kind for argument value X where X > 0. If j = 0, this function automatically calls the Bessel_k0 function.
Bessely(j,X)	Returns the value of j^{th}-order Bessel function of the second kind for argument value X where X > 0. If j = 0, this function automatically calls the Bessel_y0 function.
Bessel_i0(X)	Returns the value of zeroth-order modified Bessel function of the first kind for argument value X where -3.75 < X < infinity.
Bessel_i1(X)	Returns the value of first-order modified Bessel function of the first kind for argument value X where -3.75 < X <infinity.
Bessel_j0(X)	Returns the value of zeroth-order Bessel function of the first kind for argument value X where -3 < X < infinity.
Bessel_j1(X)	Returns the value of first-order Bessel function of the first kind for argument value X where -3 < X < infinity.
Bessel_k0(X)	Returns the value of zeroth-order modified Bessel function of the second kind for argument value X where 0 < X <infinity.
Bessel_k1(X)	Returns the value of first-order modified Bessel function of the second kind for argument value X where 0 < X < infinity.
Bessel_y0(X)	Returns the value of zeroth-order Bessel function of the second kind for argument value X where 0 < X <infinity.
Bessel_y0(X)	Returns the value of first-order Bessel function of the second kind for argument value X where 0 < X <infinity.

Table A-3: Complex Math functions[1]

Function	Description
Abs(X)	Returns the magnitude of complex variable X. See also Magnitude(X).
Angle(X) AngleDeg(X) AngleRad(X)	Returns the angle (also called amplitude or argument) of complex variable X. Representing X as X_r + i*X_i, this function returns ArcTan(X_i/X_r). The function Angle will return the angle in either degrees or radians depending on the unit setting made with the $UnitSystem directive or the Unit System dialog.. The function AngleDeg will always return the angle in degrees and the function AngleRad will always return the angle in radians. All three functions return the angle in the correct quadrant of the complex plane. Note that the Angle functions are used to extract the angle of a complex number variable or expression, but they cannot be used for assigning the angle of a complex number.
Cis(X)	Returns complex number that is equal to Cos(X)+i*Sin(X).
Conj(X)	Returns the complex conjugate of a complex variable X. Representing X as X_r + i*X_i, this function returns X_r − i*X_i.
Imag(X)	Returns the imaginary part of a complex variable X. Representing X as X_r + i*X_i, this function returns X_i.
MagnitudeX)	Returns the magnitude (also called modulus or absolute value) of a complex variable X. In complex mode, the Abs function also returns the magnitude.
Real(X)	Returns the real part of a complex variable X. Representing X as X_r + i*X_i, this function returns X_r.
Sqrt(X)	Returns the complex result of the square root of complex variable X.

1. The functions listed in this table will only work in Complex mode. Note that cos, sin, tan, cosh, sinh, tanh, exp, sqrt, ln, and log10 will operate properly in both Complex and Real modes.

Table A-4: Conditional Assignment functions.

Function	Description
If(A, B, X, Y, Z)	If A < B; the function will return the value supplied for X; if A = B, the function will return the value of Y; if A > B, the function will return the value of Z. The If function is used to provide conditional assignments, or logic However, conditional assignments are more easily and clearly implemented with the If-Then-Else statement in functions and procedures, as described in Chapter 3.
Sign(X)	Returns 1 if the arguments is greater than zero and -1 otherwise.
Step(X)	Returns a value of 1 if X is greater than equal to zero; otherwise it will return zero. The step function can be used to provide conditional assignments, similar to the If function. However, conditional assignments are more easily and clearly implemented with the If-Then-Else statement in functions and procedures, as described in Chapter 3.

Table A-5: Diagram Window Support functions.

Function	Description
DiagramHeight#(Name$) DiagramWidth#(Name$)	Returns the height or width in pixels of the Diagram Window or Child Diagram Window that is given by Name$ which can either be a string constant or string variable. The main Diagram Window is called 'Main'. If the specified name can not be found, this function returns the height of the Main Diagram Window. The height of the Diagram Window can be converted into inches using the PixelsPerInch# function. Knowing the height of the Diagram Window is useful when moving or placing objects, for example during animations.
Loop#	Loop# has no arguments and it should only be used with animation. It will return 0 if animation is not involved. The animation control in the Diagram Window allows the calculations to be restarted after they are completed thereby providing a continuous display. Loop# is a counter indicating the number of times the calculations have been started. Under some circumstances, this information can be used in the Equations Window. For example, it is possible to solve a dynamic steady-state problem in which the end condition is the starting condition using the animation control. In this case, the initial conditions are found by reading the value at the end of the Parametric Table, but this value does not exist until the table has completed one loop. The Loop# variable can be used in a function to test for this condition.
PixelsperInch#	Returns the number of pixels per inch used to display images on the screen. This function takes no argument. It is most useful in placing text and graphic objects on the Diagram Window at a particular location, such as is needed during an animation.
RGB(R,G,B)	Returns the value of color formed by providing the red, green, and blue color component values. The component values (R, G, and B) can range from 0 to 255. The RGB function is used to specify the color of objects or text items in the Diagram Window that have been associated with EES variables. For example, if an object has been created in the Diagram Window and given the name Ball, then Ball.FillColor=RGB(0,255,0) will set its color to be green.

Table A-6: Hyperbolic functions.

Function	Description
ArcCosh(X)	Returns the value which has a hyperbolic cosine equal to the value X.
ArcSinh(X)	Returns the value which has a hyperbolic sine equal to the value of X.
ArcTanh(X)	Returns the value which has a hyperbolic tangent equal to the value of X.
Cosh(X)	Returns the hyperbolic cosine of the value provided as the argument, X.
Sinh(X)	Returns the hyperbolic sine of the value provided as the argument, X.
Tanh(X)	Returns the hyperbolic tangent of the value provided as the argument, X.

Table A-7: Integral functions.

Function	Description
Integral(F,X) Integral(F,X,Start,Stop,Step)	Returns the integral of the expression represented by F with respect to variable X, i.e., $\int F\,dX$. There are two forms of the Integral function. If the values of the arguments Start, Stop, and Step are not provided, the Integral function can only be used in conjunction with a Parametric Table. In this case, X must be a variable that has values defined in one of the columns of the Parametric Table. F is a variable or algebraic expression involving X and other variables or values. If Start, Stop, and (optionally) Step are provided, EES will repeatedly solve all equations involving variable X, setting the value of X to values between Start and Stop as appropriate. If Step is not provided, EES will internally choose a step size using an automatic stepsize adjustment algorithm. The Integral function can be used to solve initial value differential equations. See Chapter 7 for additional information.
IntegralValue(t,X)	Returns a value from the Integral Table that is created using the $IntegralTable directive. t is the value of the independent integration variable for which the value of variable X is to be returned. X is a string constant indicating the name of a variable that has been included in the Integral Table. (The single quotes around the string constant are optional.) The value provided for t must be less than or equal to the current value of the independent integration variable.

Table A-8: Statistical functions.

Function	Description
Average(X1, X2, …XN)	Returns the average value of the arguments. The number of arguments must be between 1 and 1000. An array or subset of an array can be provided as an argument list by using array range notation, e.g., Average(X[1..50]). (See also the AvgLookup and AvgParametric functions in Table A-8.)
Chi_Square(X,k)	Returns the value of the cumulative chi-square distribution, $F\left(X,k\right)$ where: $$F\left(X,k\right)=\frac{\int_0^{X/2} t^{k/2-1}e^{-t}dt}{\int_0^{\infty} t^{k/2-1}e^{-t}dt}=\frac{\gamma\left(\frac{k}{2},\frac{X}{2}\right)}{\Gamma\left(\frac{k}{2}\right)}$$ where γ is the lower incomplete Gamma function and Γ is the Gamma function.
Erfc(X)	Returns the complementary error function defined as $$Erfc(X)=1-Erf\left(X\right)=\frac{2}{\sqrt{\pi}}\int_X^{\infty} e^{-t^2}dt$$
Gamma(X)	Returns the Γ function for the argument value X, defined as $$\Gamma(X)=\int_0^{\infty} t^{X-1}e^{-t}dt$$
Max(X)	Returns the largest argument value. The number of arguments must be between 1 and 1000 . An array or subset of an array can be provided in the argument list by using array range notation, e.g., Max(X[1..50]). (See also MaxLookup and MaxParametric in Table A-9.)
Min(X)	Returns the smallest argument value. The number of arguments must be between 1 and 1000 . An array or subset of an array can be provided in the argument list by using array range notation, e.g., Min(X[1..50]). (See also MinLookup and MinParametric in Table A-9.)

Table A-8: Statistical Functions (continued).

Function	Description
Probability(X1, X2, mu, sigma)	Returns the probability (as a fraction less than 1) that a value X will lie between X1 and X2 in a normal distribution having average mu and standard deviation sigma. The mathematical description is: $$\text{probability}(X_1, X_2, \mu, \sigma) = \int_{z_1}^{z_2} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right) dz$$ where $z_1 = \frac{X_1 - \mu}{\sigma}$ and $z_2 = \frac{X_2 - \mu}{\sigma}$
Product(Arg, Series_Info)	Returns the product of a series of terms. Arg can be any algebraic expression. Series_info provides the name of the product index variable and the lower and upper limits, which must be integers or variables which have been previously set to integer values. Product(j, j=1,4) will return 1*2*3*4 or 24, which is 4 factorial. The Product function is most useful when used with array variables, e.g., X[]. For example, the product of the squares of all 10 elements in the array X[] can be obtained as Product (X[j]*X[j], j=1,10).
Random(A, B)	Returns a uniformly distributed number in the range between A and B. This function can only be used within EES functions and procedures.
Sum(Arg, Series_Info) Sum(X1, X2, … XN)	There are two forms for the Sum function. EES determines which format is in use by context. The first form returns the sum of a series of terms. Arg can be any algebraic expression. Series_info provides the name of the summation index variable and the lower and upper limits. These limits must be integers or variables that have been previously set to integer values. For example, Sum(j, j=1,4) will return 1+2+3+4 or 10. The Sum function is most useful when used with array variables, e.g., X[]. For example, the scalar product of two vectors, X[] and Y[], each with 10 elements can be obtained as Sum(X[j]*Y[j], j=1,10). The second form returns the sum of the arguments. Array range notation is particularly convenient for this form. For example, Sum(X[1..100]) returns the sum of the 100 elements in the array X[].

Table A-9: Table Data functions.

Function	Description
AvgLookup(T$,C$,R1,R2)	Returns the average of all or selected cells in a specified column of a specified Lookup Table or Lookup Table file on disk. T$ is a string constant or string variable providing the table name. C$ is an integer, string constant or string variable identifying the column. If an integer value is provided for C$, it is taken to be the column number in the Lookup Table. Otherwise it is taken to be the name of the column. R1 and R2 are optional. If they are not provided, all rows in the Lookup Table will be averaged.
AvgParametric(T$,C$,R1,R2)	Returns the average of all or selected cells in a specified column of a specified Parametric Table. T$ is a string constant or string variable providing the name of the Parametric Table. The other variables are the same as for the AvgLookup function.
Differentiate(T$, C1, C2, C1=V) Differentiate1(T$, C1, C2,C1=V) Differentiate2(T$, C1, C2, C1=V)	Differentiate returns the numerically-determined derivative using cubic interpolation of data in the Lookup Table. T$ is a string constant or string variable that provides the name of the Lookup Table or the name of an existing Lookup file stored on disk. C1 and C2 are string constants (or string variables) that identify column header names. Since there is no ambiguity, the single quotes normally used for string constants are optional. The final argument identifies a non-integer row position in the table for which the value in either column C1 (as shown) or C2 is equal to V. The derivative of the values in the other column at this row position is returned. Differentiate1 and Differentiate2 return the derivatives based on linear and quadratic interpolation, respectively.
Interpolate(T$, C1, C2, C1=V) Interpolate1(T$, C1, C2, C1=V) Interpolate2(T$, C1, C2, C1=V)	Interpolate returns a one-dimensional interpolated or extrapolated value using cubic interpolation of data in the Lookup Table. T$ is a string constant or string variable that provides the name of the Lookup table or the name of an existing Lookup file stored on disk. C1 and C2 are string constants (or string variables) that identify column header names. Since there is no ambiguity, the single quotes that are normally required for string constants are optional. The final argument identifies a non-integer row position in the table for which the value in either column C1 (as shown) or C2 is equal to V. The interpolated value for the other column at this row position is returned. Interpolate1 and Interpolate2 return interpolated values based on linear and quadratic interpolation, respectively. See Section 2.3 for more information.
Interpolate2D(T$, X, Y, Z, X=V1, Y=V2, N)	Returns an interpolated or extrapolated value as a function of two independent variables from tabular data in a Lookup Table or a Lookup file (stored on disk) provided in three columns of data. A minimum of 8 data points are required. T$ is a string constant or string variable that provides the name of the Lookup Table or file. X, Y, and Z are the string constants providing the names of three different columns in the specified table. Since there is no ambiguity, the single quotes that are normally required for string constants are optional. Any two of the three variables are set to values V1 and V2. The value of the unspecified variable is determined by two-dimensional interpolation and returned. The last parameter, N, is optional. It serves two purposes. First, it indicates which of two different interpolation algorithms are used. If N < 0, then a bi-quadratic method is used in the interpolation. If N > 0 then a multi-quadric radial basis function interpolation method is used. The absolute value of N is the maximum number of data points used in the interpolation process. The two-dimensional interpolation algorithm is computationally intensive for large values of N. If the data table contains more than N points, EES will use the N points in the table that are nearest to the point for which the interpolate is requested in normalized coordinates. If the data table contains fewer than N points, N is set to the number of points in the table. If N is set to be less than 8, EES will reset it to 8 and issue a warning message. If N is not provided, a value of -16 is assumed so that the default case is a bi-quadratic method with 16 points. See Section 2.4 for more information.

Table A-10: Table Data functions (continued).

Function	Description
Interpolate2DM(T$, X, Y)	Interpolate2DM provides two-dimensional linear interpolation for data in a Lookup Table or Lookup file that are arranged in a row and column matrix format. T$ is a string constant or string variable that provides the name of the Lookup Table or the name of an existing Lookup file stored on disk. X is the value of the independent variable whose values appear in the first row of the table. The columns used to determine the interpolated value will be identified by this value. Y is the value of the independent variable whose values appear in the first column of the table. The rows used to determine the interpolated value will be identified by this value. See Section 2.4 for more information.
Lookup(T$, Row, C$)	Returns the value in Lookup Table T$ at the specified row and column. T$ is a string constant or string variable that provides the name of the Lookup Table. Row is the row in the table, but it does not need to be an integer value and linear interpolation will be used to determine values between rows. C$ indicates the column in the table. C$ is an integer, string constant, or string variable indicating the column. If an integer is provided, it is taken to be the column number. Otherwise, the string constant or variable must contain the name of one of the columns. The Lookup function can also be used to set the value of a cell in the Lookup Table window when it is used on the left-hand side of an equation in an internal function or procedure. Also see Lookup$ in Appendix B.
LookupCellEmpty(T$,Row,C$)	The function requires the same arguments as the Lookup function. It returns either a 0 (if the specified cell is empty) or 1 if it contains a value. The function can be used with logic statement such as If-Then-Else and Repeat-Until in internal functions and procedures to ensure that a cell in the Lookup Table or file contains a value.
LookupCol(T$, Row, V) LookupCol1(T$, Row, V)	The LookupCol function scans a specified Row in Lookup Table T$ and returns the column in this table at which the value is equal to V. The return value may not be an integer. A bisection method is used in finding the column, using all of the columns of data in the specified row. The data in the specified row of the table must be monotonic, either increasing or decreasing. LookupCol1 operates in exactly the same manner except that it ignores the data in the first column of the table.
LookupRow(T$, C$, V) LookupRow1(T$, C$, V)	The LookupRow function scans a specified column in Lookup Table T$ and returns the row in this table at which the value is equal to V. C$ is an integer, string constant, or string variable indicating the column. If an integer is provided, it is taken to be the column number. Otherwise, the string constant or variable must contain the name of one of the columns. The return value may not be an integer. A bisection method is used in finding the row, using all of the rows of data in the specified column. The data in the specified column of the table must be monotonic, either increasing or decreasing. LookupRow1 operates in exactly the same manner except that it ignores the data in the first row of the table. (Also see Lookup$Row in Appendix B.)
MaxLookup(T$, C$, R1, R2)	Returns the maximum of all or selected cells in a specified column of the Lookup Table or Lookup file on disk having name T$. T$ can be a string constant or string variable. C$ is an integer, string constant, or string variable indicating the column. If an integer is provided, it is taken to be the column number. R1 and R2 are optional arguments that provide the starting and stopping rows for which the maximum will be found.
MaxParametric(T$,C$,R1,R2)	Returns the maximum of all or selected cells in a specified column of the Parametric Table having name T$. T$ can be a string constant or string variable. C$ is an integer, string constant, or string variable indicating the column. If an integer is provided, it is taken to be the column number. R1 and R2 are optional arguments that provide the starting and stopping rows for which the maximum will be found.

Table A-11: Table Data functions (continued).

Function	Description
NArrayColumns(T$)	Returns the number of columns in the Arrays Window having name T$. The Arrays Window for the main program is named 'Main'. If the Arrays Table does not exist, the function call will return 0.
NArrayRows(T$)	Returns the number of rows in the Arrays Window having name T$. The Arrays window for the main program is named 'Main'. If the Arrays table does not exist, the function call will return 0.
NFileRows(T$)	Returns the number of rows in a text file that is stored on disk. T$ is a string constant or a string variable that contains the name of the file. If the file does not exist or it cannot be read, the function returns 0.
NLookupColumns(T$)	Returns the number of columns in the specified Lookup Table or a Lookup file that is stored on disk. T$ is a string constant or a string variable that contains the name of the Lookup Table or file. The function will return 0 if the table or file are not found.
NLookupRows(T$)	Returns the number of rows in the specified Lookup Table or a Lookup file that is stored on disk. T$ is a string constant or a string variable that contains the name of the Lookup Table or file. The function will return 0 if the table or file are not found.
StdDevLookup(T$,C$,R1,R2)	Returns the standard deviation of all or selected cells in a specified column of a Lookup Table or Lookup file on disk. T$ is a string constant or a string variable that contains the name of the Lookup Table or file. C$ is a integer value, string constant or string variable the identifies the column. If an integer value is provided, it is taken to be the column number. If a string constant is provided, it is assumed to be the name of the variable for the column. The single quotes that normally identify a string constant are not required. If a string variable is provided, it must contain the name of one of the columns in the Lookup Table. R1 and R2 are integers or EES variables specifying the starting and stopping rows for which the operation will be performed. These parameters are optional.
StdDevParametric(T$, C$, R1, R2)	Returns the standard deviation of all or selected cells in a specified column of Parametric Table T$. C$ is a integer value, string constant or string variable the identifies the column. If an integer value is provided, it is taken to be the column number. If a string constant is provided, it is assumed to be the name of the variable for the column. The single quotes that normally identify a string constant are not required. If a string variable is provided, it must contain the name of one of the columns in the Parametric Table. R1 and R2 are integers or EES variables specifying the starting and stopping rows for which the operation will be performed. These parameters are optional.
SumLookup(T$, C$, R1, R2)	Returns the sum of all or selected cells in a specified column of a Lookup Table or Lookup file on disk. T$ is a string constant or a string variable that contains the name of the Lookup Table or file. C$ is a integer value, string constant or string variable the identifies the column. If an integer value is provided, it is taken to be the column number. If a string constant is provided, it is assumed to be the name of the variable for the column. The single quotes that normally identify a string constant are not required. If a string variable is provided, it must contain the name of one of the columns in the Lookup Table. R1 and R2 are integers or EES variables specifying the starting and stopping rows for which the sum will be performed. These parameters are optional.

Table A-12: Table Data functions (continued).

Function	Description
SumParametric(T$,C$,R1,R2)	Returns the sum of all or selected cells in a specified column of a ParametricTable. T$ is a string constant or a string variable that contains the name of the Parametric Table. C$ is a integer value, string constant or string variable the identifies the column. If an integer value is provided, it is taken to be the column number. If a string constant is provided, it is assumed to be the name of the variable for the column. The single quotes that normally identify a string constant are not required. If a string variable is provided, it must contain the name of one of the column. R1 and R2 are integers or EES variables specifying the starting and stopping rows for which the sum will be performed. These parameters are optional.
TableRun#	Returns the Parametric Table run number for which calculations are currently in progress. This function has no arguments and it should only be used only when calculations are initiated with the Solve Table or Min/Max Table command in the Calculate menu. The name of the Parametric Table that is currently being used is accessed with the TableName$ function.
TableValue(T$, Row, C$)	Returns the value stored in the row and column of the specified Parametric Table identified by T$, which can either be a string variable or a string constant. C$ is the column, which may be either entered directly as an integer number or by supplying the variable name for the desired column within single-quotes. Alternatively a string variable may be used to provide the name of the column. The function is useful in the solution of some marching-solution type problems in which the current value of a variable depends on its value in previous calculations.
TimeStamp(T$)	Returns an integer coded representation of the date and time that disk file T$ was last modified. The larger this number is, the more recent the file. If the specified file is not found, the function returns -1. T$, which can either be a string variable or a string constant. (See also TimeStamp$)

Table A-13: Trigonometric functions.

Function	Description
ArcCos(X)	Returns the value which has a cosine equal to the value X.
ArcSin(X)	Returns the value which has a sine equal to the value of X.
ArcTan(X)	Returns the value which has a tangent equal to the value of X.
Cos(X)	Returns the cosine of the value provided as the argument, X.
Sin(X)	Returns the sine of the value provided as the argument, X.
Tan(X)	Returns the tangent of the value provided as the argument, X.

Table A-14: Unit functions.

Function	Description
Convert(From$, To$)	Returns the conversion factor need to change the units of unit designator From$ to To$. From$ and To$ can be string constants or string variables. If string constants are used, the single quote marks are optional. See Section 1.5.
ConvertTemp(F$,T$,T)	Converts a temperature value T represented in temperature scale F$ to its corresponding value in temperature scale T$. F$ and T$ can be string constants or string variables identifying the temperature scales, which must be C, K, F, or R. See Section 1.5.
UnitSystem(S$)	Returns 1 (for true) or 0 (for false) indicating whether the unit specification S$ is a current unit setting. S$ can be a string constant or string variable that is one of the following strings: 'SI', 'Eng', 'Mass', 'Molar', 'Deg', 'Rad', 'Pa', 'kPa', 'bar', 'MPa', 'psia', 'atm', 'C', 'K', 'F', 'R', 'J', and 'kJ'. The UnitSystem$ string function provides related information. See Section 11.1.

Appendix B: Built-In String Functions

EES strings consist of up to 255 single-byte (ANSI) characters. A string constant is an EES string, surrounded by single quotes. String variables are identified by having a name that ends with the $ character. Table B-1 lists all of the functions that are provided in EES to work with strings.

Table B-1: Built-In String functions.

Function	Description
Chr$(X)	Returns a string consisting of one character having an ASCII code as given by the argument value, X. The value must between between 0 and 255. This function is useful for providing characters that cannot be entered on the keyboard or would not be accepted by EES.
Concat$(A$, B$)	Returns a single string that is the concatenation of the two strings provided as arguments. A$ and B$ can be either string constants or string variables.
Copy$(S$, N, L)	Returns a substring of the string expression provided as the first argument, S$. The second argument, N, is an integer that indicates the character position at which the substring starts. The third parameter, L, is an integer that provides the substring length. If this length exceeds the length of the string expression, it is set to the length of the string expression.
Date$	Returns the current date. The function has no arguments. The format of the date is controlled by the settings in the Regional Options Control Panel in the Windows Operating System.
EESFileDir$	Returns a string containing the directory name of the current EES file. The function has no arguments. The string returned by this function ends with a \ character.
EESFileName$	Returns the complete file name (including the directory) of the current EES file. The directory information can be removed with the ShortFileName$ function. The function has no arguments.
EESProgramDir$	Returns the name of the directory of the EES application. The directory name ends with a \ character. The function has no arguments.
Lookup$(T$, Row, C$)	Returns the string in Lookup Table T$ at the specified row and column. T$ is a string constant or string variable that provides the name of the Lookup Table. Row is the row in the table. C$ indicates the column in the table. C$ is an integer, string constant, or string variable indicating the column. If an integer is provided, it is taken to be the column number. Otherwise, the string constant or variable must contain the name of one of the columns. Note that the format style of the column in the Lookup Table must be set to String if the column is to contain string data. (Also see the Lookup function in Appendix A.)
LookupColName$(T$,C)	Returns or changes the name of a specified column in a Lookup Table. T$ is the name of a Lookup Table or a Lookup file. It may be a string constant or string variable. C can be a number or an expression that evaluates to the column number in the table. If the column is not found, the function will return '??'. The name of an existing column in a Lookup Table can only be changed within an internal function or procedure. To change the column name, place the LookupColName$ function call on the left hand side of an equation in an internal function or procedure and set it equal to a string containing the column name and units, separated by the \ character.
Lookup$Row(T$, C$, N$)	Returns the row in column C$ of Lookup Table T$ in which this string N$ exists. T$ and N$ can be string constants or string variables. C$ can be an integer, a string constant or a string variable that indicates the column in the Lookup Table.
LookupTabName$(Tab)	Returns the name of the Lookup Table with tab number Tab. If no such tab exists then the function returns an empty string.

Table B-1: Built-In String functions (continued).

Function	Description
Lowercase$(S$)	Returns a string in which all letters in the string provided in argument S$ are converted to lowercase.
ShortFileName$(T$)	T$ is a string constant or string variable that contains a Windows filename including the path and filename name extension. The function returns a string that is set to the filename with the path information removed.
String$(V, {F$})	Returns a string representing the numerical value of the argument,V. If V is the only argument, the function uses Autoformat to set the format of the value before converting it to a string. However, a second format specification can be provided as a second argument, either as string constant or string variable. The format specification is a two-character string with the first character being E, F, G, or T and the second character being a number between 0 and 9. Specification F3 will provided fixed decimal format with 3 numbers to the right of the decimal separator. E and G indicate floating point format. T indicates time format.
StringLen(S$)	Returns the number of characters in S$, which can be a string constant or string variable.
StringPos(A$,B$)	Returns the character position of the first string (A$) within the second (B$). If the first string is not contained within the second, the function returns 0. This function is case-sensitive. A$ and B$ can be string constants or string variables.
StringVal(S$)	Returns the numerical value formed by the characters of a string constant or string variable. This function provides the inverse operation of the String$ function.
TableTabName$(Tab)	Returns the name of the Parametric Table with tab number Tab. If no such tab exists then the function returns an empty string.
Time$	Returns the current time in a string with a format that is controlled by the settings in the Regional Options Control Panel of the Windows Operating System. The function has no arguments.
TimeStamp$(T$)	Returns a string that displays the date and time that file T$ was last modified. If the file cannot be found, the function returns 'File not found'. (See also TimeStamp in Appendix A.)
Trim$(S$)	Returning a string that is equal to the argument, S$, but with any leading or trailing spaces removed.
Uppercase$(S$)	Returns a string in which all letters in the string provided in argument S$ are converted to uppercase.
UnitsOf$(X)	Returns a string that is the units of the EES variable X that is provided as an argument to the function.
UnitSystem$(U$)	The UnitSystem$ function provides string information relating to the current unit system settings. The function takes one argument, U$, that can be a string constant or string variable and must be one of the following keywords: 'Temperature', 'Pressure', 'Mass', 'Energy, 'Entropy', 'Volume', or 'Trig'. The function will return a string that represents the current unit setting corresponding to the keyword. Possible return strings for each of the arguments are as follows: Temperature: F, C, R, K Pressure: psia, atm, Pa, kPa MPa, bar Energy: Btu/lb_m, Btu/lbmol, J/kg, J/kmol, kJ/kg, kJ/kmol Entropy: Btu/lb_m-R, Btu/lbmol-R, J/kg-K, J/kmol-K, kJ/kg-K, kJ/kmol-K Volume: ft^3/lb_m, ft^3/lbmol, m^3/kg, m^3/kmol Trig: radians, degrees (See Section 3.5 for an example.)

Appendix C: Directives

A complete list of the directives implemented at the time of publication is provided in Table C-1. See Chapter 14 for more information and examples relating to the use of directives.

Table C-1: Summary of directives.

Related to Display & Formatting Options	
Directive	**Description**
$Arrays On/Off	Places array variables in the Arrays Window (On) or the Solutions Window (Off). This directive can also be used to put array variables in a function or procedure into an Arrays Window.
$Bookmark	Places bookmarks in large EES programs in order to allow easy navigation.
$HideWindow	Hides a specified window.
$Keyboard US/EU	Changes the decimal separator, list separator, and end of line indicator to be the US or EU conventions, overriding the setting made in the Windows operating system. The US setting uses a decimal point for the decimal separator whereas the EU setting uses a comma for the decimal separator. The US and EU list separators are the comma and the semicolon, respectively.
$Private	Prevents the EES equations within a library from being viewed from the Function Information dialog.
$ShowWindow	Specifies the window that will be shown after calculations are completed.
$SumRow On/Off	Specifies whether a summation row will be displayed in Parametric Tables.
$TabStops	Specifies the position of the tabs used in the Equations Window.
$Warnings On/Off	Specifies whether warning messages will be displayed or not.

Related to Units & Properties	
Directive	**Description**
$AutoSetUnits On/Off	Activates or deactivates EES' ability to automatically set the units for each variable.
$CheckUnits On/Off and AutoOn/AutoOff	Turns off unit checking for a specified subset of the equations or the entire set of equations.
$Reference	Allows the default reference state for a specified fluid to be changed.
$UnitSystem	Specifies the unit system used by EES.

Related to Functions, Procedures, Subprograms, & Modules	
Directive	**Description**
$Common	Allows variables to be declared as common; that is, they can be accessed from within a Function, Procedure, Module, or Subprogram without being passed as input arguments.
$Constant	Allows the declaration of constants within EES.
$RequiredOutputs	Allows the number of outputs used in the Call statement for a procedure to be less than the number of outputs used in the procedure declaration.

Related to Complex Algebra	
Directive	**Description**
$Complex On/Off	Activates or deactivates the option of using complex algebra within EES.
$Real	Declares variables to be real when EES is operating in complex mode.

Related to Saving & Copying Data	
Directive	**Description**
$CopyToLookup[1]	Activates or deactivates the option of using complex algebra within EES.

$Export	Exports selected variables to a text file.
$Import	Imports variables from a text file
$OpenLookup[1]	Uses a specified file to create a Lookup Table
$SaveLookup[1]	Saves a specified Lookup tabel to a disk file
$SaveTable	Saves the contents of a specified table to a disk file.

Related to Program Execution & Debugging

Directive	**Description**
$DoLast & $EndDoLast	Allows a set of equations to be compiled and executed after all other equations are compiled and executed.
$If, $IfNot, $Else, & $EndIf	Allows conditional exclusion of equations in the Equations window.
$Include	Provides a method for loading various types of files into an EES program. These files can include additional EES equations, Functions, libraries, macros, variable information, preferences, and units.
$IntegralAutoStep	Allows the specification of the parameters that are used to control the automatic step size specification in the Integral command.
$IntegralTable	Creates a table that contains the EES variables that are computed during the numerical evaluation of an integral with the Integral command.
$MaxCalls	Used to establish the maximum number of times that the equations in a function or procedure can be called.
$StopCriteria	Specifies the stop criteria that control the iterative solution process.
$Trace	Produces a table of intermediate values for specified variables that occur during the iterative solution process.
$UpdateGuesses	Automatically updates the guess values in a subprogram after calculations are completed.

1. Directive is only available in the Professional version of EES.

Appendix D: Macro Commands

A complete list of the macro commands implemented at the time of publication is provided in in Table D-1. The macro commands are listed by category in the following tables. Note that optional keywords are enclosed in curly braces.

- General macro commands related to program execution (D-1)
- Macro commands related to communicating with other programs (D-2)
- Macro commands related to file management (D-3)
- Macro commands related to plots (D-4)
- Macro commands related to tables (D-5)

See Chapter 18 for more information and examples relating to the use of macro commands.

Table D-1: General macro commands related to program execution

Macro Command	Description
Assign EESVariable=Value	Assigns a value to the variable EESVariable.
Beep {Type}	Generates an audible sound that is specified by the value of Type, as indicated in Table 18-2.
Check	Equivalent to selecting Check/Format from the Calculate menu.
Copy EquationWindow {LX CY LXX CYY}	Copies the contents of the Equation Window to the clipboard. The optional parameters {LX CY LXX CYY} specify the range of lines and characters to be copied; if these parameters are not included then the entire Equations Window is copied.
Copy Diagram	Copies the contents of the Main Diagram Window to the clipboard as a .wmf file.
Copy SolutionWindow	Copies the contents of the Main Solution Window to the clipboard.
Goto Line	Transfers execution to the macro command line labeled Line. Lines are labeled using the Label command. Line must be an integer between 1 and 29999.
HideWindow WindowName	Hides the window designated by the WindowName parameter which can be EES, Equations, Formatted, Solution, Residuals, Arrays, Parametric, Lookup, Integral, Plot, or Diagram. If WindowName is EES, the EES application and all its windows are hidden.
If (Conditional) Then MacroCommand1 {Else MacroCommand2}	Executes MacroCommand1 if the conditional test Conditional is true. The optional Else statement executes MacroCommand2 if the conditional test is false.
Label Line	Labels a line according to the parameter Line which must be a value between 1 and 30000.
Load #N	Loads program N in a distributable file.
Log Message$ (or On/Off)	Writes Message$ in the EESMacro.log that is created when the macro is played. The keyword Off causes logging to cease and the keyword On causes it to be resumed.
LogFileName File$	Changes the name of the macro log file to File$. Note that if File$ is not included then no log file is written.
Maximize G a b c {/Method=MethodKeyword}	Maximizes the variable G with respect to the variables in the list a b c. The optional MethodKeyword specifies the optimization method.

Table D-1: General macro commands related to program execution (continued).

Macro Command	Description
MaximizeTable G a b c {Method=MethodKeyword /Table=Table$ /Rows=RowStart..RowEnd}	Maximizes the variable G with respect to the variables in the list a b c for each of the rows in a Parametric Table. The optional MethodKeyword can be used to specify the optimization technique. The optional Table$ can be used to specify the Parametric Table. The optional RowStart..RowEnd parameters can be used to specify the rows within the table.
MessageDialog([Caption1$ {,Caption2$, Caption3$}], Message$)	Displays the message Message$ and provides a button labeled Caption1$. Optionally, additional buttons can be displayed with captions Caption2$, Caption3$, etc. The MessageDialog command will return an integer that indicates which button is selected. The message text Message$ can refer to a URL.
Minimize G a b c {/Method=MethodKeyword}	Minimizes the variable G with respect to the variables in the list a b c. The optional MethodKeyword specifies the optimization method.
MinimizeTable G a b c {Method=MethodKeyword /Table=Table$ /Rows=RowStart..RowEnd}	Minimizes the variable G with respect to the variables in the list a b c for each of the rows in a Parametric Table. The optional MethodKeyword can be used to specify the optimization technique. The optional Table$ can be used to specify the Parametric Table. The optional RowStart..RowEnd parameters can be used to specify the rows within the table.
New	Clears the work space and brings up an empty Equations Window. The existing file will be closed without saving.
Pause Time	Pauses execution for the number of seconds specified by Time.
Print Window	Prints the window specified by the Window parameter (see Table 18-7).
PrintSetup Printer=Printer$ {Orientation=Portrait (or Landscape) Copies=NCopies}	Sets the default printer to Printer$. The orientation and number of copies can be optionally specified.
Quit	Closes the EES application.
ShowWindow Arrays	Shows the Arrays Window.
ShowWindow Diagram {Name$}	Shows the Main Diagram Window or the Child Diagram Window specified by the optional parameter Name$.
ShowWindow EES	Shows the EES application if it was previously hidden.
ShowWindow Equations	Shows the Equations Window.
ShowWindow Format	Shows the Formatted Equations Window.
ShowWindow Integral	Shows the Integral Table Window.
ShowWindow Lookup {Table$}	Shows the Lookup Table Window. The optional parameter Table$ specifies the particular Lookup Table.
ShowWindow Parametric {Table$}	Shows the Parametric Table Window. The optional parameter Table$ specifies the particular Parametric Table.
ShowWindow Plot {Plot$ (or Plot#)}	Shows the Plot Window. The optional parameter Plot$ specifies the particular plot by name. The integer parameter Plot# specifies the plot by tab number.
ShowWindow Progress	Shows the Progress Window during calculations.
ShowWindow Residuals	Shows the Residuals Window.
ShowWIndow Solution {Solution$}	Shows the Solution Window. The optional parameter Solution$ specifies the particular Solution Window.
Solve	Initiates the solution process.
SolveTable Table$ {Rows=RowStart...RowEnd}	Initiates the Solve Table solution process using Parametric Table Table$. The optional parameters RowStart ... RowEnd specifies the rows in the table.
Stop	Execution of the Stop macro command immediately stops the macro.

StopCrit {It=ItValue Time=TimeValue Res=ResValue Var=VarValue}	Specifies the stop criteria for the solution process. The number of iterations (ItValue), elapsed time (TimeValue), relative residual (ResValue), and change in variables (VarValue) can be specified.
Macro Command	**Description**
Uncertainty X1 X2 ... Y1=UY1 Y2=UY2 ...	Initiates the uncertainty propagation calculations required to determine the uncertainty of the calculated variables in the list X1 X2 … due to the specified uncertainties (UY1, UY2, ...) of the selected measured variables (Y1, Y2, ...)
Units SI Mass (or Mole) Deg (or Rad) kPa (or Pa, bar, MPa) J (or kJ) C (or K)	Sets the unit system to SI and specifies the unit values.
Units Eng Mass (or Mole) Deg (or Rad) psia (or atm) F (or R)	Sets the unit system to SI and specifies the unit values.
Update Guesses	Sets all variable guess values to their last set of calculated values.
VarInfo Variable {Lower=LowerValue Upper=UpperValue Guess=GuessValue}	Sets the bounds (LowerValue and UpperValue) and guess value (GuessValue) for the variable Variable.

Table D-2: Macro Commands Related to Communicating with Other Programs.

Macro Command	Description
Excel.Cell(C$,Text$)	Places the text Text$ in the cell specified by C$ in an open Excel file. The string C$ must include the column letter followed by the row number (e.g., C$='B3')
Excel.FileNew	Starts Excel if it is not already open and creates a new, empty worksheet.
Excel.FileOpen(File$)	Opens the Excel file File$.
Excel.FileSaveAs(File$)	Saves the Excel file as File$.
Excel.Hide	Hides the currently open Excel application.
Excel.Paste	Pastes the contents of the clipboard into the currently open Excel application at the selection point defined by the Excel.Range macro command.
Excel.Range(Range)	Sets the selection point within the currently open Excel worksheet according to the Range parameter.
Excel.Quit	Closes the currently Excel application.
Excel.Sheet(Sheet$)	Activates the Excel worksheet Sheet$ in the currently open Excel application.
MATLAB.Execute(Command$)	Executes the command Command$ in the currently open MATLAB command window.
MATLAB.Open	Starts MATLAB and opens the command window.
MATLAB.Quit	Closes the instance of MATLAB opened within the macro.
Run Statement	Acts as if the parameter Statement were entered into the Windows Start menu.
Word.FileNew	Starts Microsoft Word with an empty document.
Word.FileOpen(File$)	Starts Microsoft Word and opens the file File$.
Word.FileSaveAs(File$)	Saves the current Microsoft Word document as File$.
Word.Hide	Hides the current Microsoft Word document.
Word.Insert(Text$)	Inserts the text Text$ into the current Microsoft Word document.
Word.Paste	Pastes the contents of the clipboard into the current Microsoft Word document.
Word.PasteSpecial(FormatType)	Pastes the contents of the clipboard into the current Microsoft Word document using the format specified by the FormatType parameter.
Word.Quit	Closes the current Microsoft Word document.
Word.Show	Makes the current Microsoft Word document visible.

Table D-3: Macro Commands Related to File Management.

Macro Command	Description
Delete File$	Deletes the file File$.
Export {/A} File$ X, Y, Z, ...	Opens the file File$ and writes the values of the variables in the list X, Y, Z, ... to the file. If File$ = 'clipboard' then the variables are written to the clipboard. The optional /A appends the values of the variables to an existing file.
GetDirectory	Returns the current Windows directory which can be assigned to a string variable.
Import File$ X, Y, Z, ...	Opens the file File$ and reads the values of the variables in the list X, Y, Z, ... If File$='clipboard' then the variables are read from the clipboard.
LoadLibrary File$	Loads the library file File$.
Open File$	Open the file File$.
Save {File$}	Saves the existing file. The optional File$ can be used to specify a name other than the existing file name.
SaveArrays Array$ File$ {/A /T /N /E}	Saves the contents of the Arrays Table with name Array$ to File$. The optional parameters /A /T /N and /E specify append, transpose, include column header and units, and save in Lookup Table format, respectively.
SaveIntegral File$ {/A /T /N /E}	Saves the contents of the Integral Table to File$. The optional parameters /A /T /N and /E specify append, transpose, include column header and units, and save in Lookup Table format, respectively.
SaveLookup Lookup$ File$ {/A /T /E /E}	Saves the contents of the Lookup Table with name Lookup$ to File$. The optional parameters /A /T /N and /E specify append, transpose, include column header and units, and save in Lookup Table format, respectively.
SavePlot Plot$ (or Plot#) File$	Saves the plot Plot$ to file File$. The extension of the file (.jpg, .bmp, .tif, or .emf) specifies the format. The plot can also be indicated by the integer Plot# which specifies the plot tab number.
SaveSolution File$ {/A}	Saves the contents of the Solution Window to the text file File$. The optional /A parameter appends the solution to the file.
SaveTable Table$ File$ {/A /T /N /E}	Saves the contents of the Parametric Table with name Table$ to File$. The optional parameters /A /T /N and /E specify append, transpose, include column header and units, and save in Lookup Table format, respectively.
SetDirectory(Directory$)	Sets the current Windows directory to Directory$.

Table D-4: Macro commands related to Plots.

Macro Command	Description
AddPlotText PlotName$ Xpos Ypos Text$ {FontSize FontStyle FontColor}	Adds the string Text$ at location Xpos Ypos on the plot named PlotName$. The optional parameters FontSize, FontStyle, and FontColor specify the size, style, and color of the font.
Copy PlotWindow Plot$ {PlotNumber}	Copies the graphics in the Plot Window named Plot$ to the clipboard. Alternatively, the PlotNumber parameter specifies the number of the tab.
DeletePlot Plot$ {PlotNumber}	Deletes the Plot Window named Plot$. Alternatively, the PlotNumber parameter specifies the number of the plot tab.
DuplicatePlot Plot$ PlotCopy$	Copies the plot Plot$ and names the copy PlotCopy$.
ModifyAxis Axis Name=Plot$ {Min=MinValue Max=MaxValue Linear (or Log) Int=IntValue Ticks239=on (or off) Grids=on (or off) Showscale=on (or off) Size=SizeValue Style=StyleKeyword Format=FormatValue Color=ColorValue}	Changes the appearance of the axis indicated by the Axis parameter (X, Y, X2, or Y2) in Plot$. The optional parameters the specify the characteristics of the axis are discussed in Table 18-3.
ModifyPlot Name=Plot$ Width=WidthValue cm (or in) Height=HeightValue cm (or in) {GridColor=ColorValue}	Changes the size of the plot Plot$ to according to WidthValue and HeightValue. Optionally, the grid color can be specified by the parameter ColorValue.
NewContourPlot Table=Table$ (or TableType#) X=XColumn Y=YColumn Z=ZColumn Type=TypeValue {Legend=yes (or no) Resolution=Res Xmin=XminValue Xmax=XmaxValue Xint=XintValue Ymin=YminValue Ymax=YmaxValue Yint=YintValue Zmin=ZminValue Zmax=ZmaxValue Zint=ZintValue Name=Plot$}	Creates a new contour plot using the data in Table$ columns XColumn, YColumn, and ZColumn. The type of the plot is specified with the TypeValue keyword. The remaining, optional keywords are discussed in Table 18-4.
NewPlot Name=Plot$ Table=Table$ (or TableType#) X=XColumn Y=YColumn {Rows=RowStart..RowEnd Line=LineType Symbol=SymbolType SymbolSize=SymbolSize Color=ColorType Legend}	Creates a plot with name Plot$ using data in Table$ and columns XColumn and YColumn. The remaining, optional keywords specify the rows (RowStart to RowEnd), the line type (LineType, listed in Table 18-5), the symbol type (SymbolType, listed in Table 18-6), the symbol size (SymbolSize), color (ColorType) and the use of a legend (Legend).
OverlayPlot Name=Plot$ Table=Table$ (or TableType#) X=XColumn Y=YColumn {Rows=RowStart..RowEnd Line=LineType Symbol=SymbolType SymbolSize=SymbolSize Color=ColorType Right Legend}	Overlay a plot on the existing plot Plot$. The OverlayPlot macro operates in the same way as the NewPlot macro with the exception that the Right keyword can be used to specify that the data be plotted using the secondary *y*-axis.
PropPlot Fluid$ Type NProp1 Val1Prop1 Val2Prop1 ... NProp2 Val1Prop2 Val2Prop2 ... {DoQLines}	Generates a property plot of the type specified by the Type keyword (see Table 18-8) for fluid Fluid$. The number of the constant property lines of the first type are specified by the parameters NProp1 and the values are specified in the list Va11Prop1 Val2Prop1 ...). The parameters NProp2 and list Val1Prop2 Val2Prop2... specify the number and values of the constant property lines of the second type. The type of constant property lines depend on the type of property plot, as listed in Table 18-8. The optional DoQLines parameter activates lines of constant quality.
PropPlot AirH2O Psy Nwb Twb1 Twb2 ... Nv v1 v2 ... P=Pval {Mollier}	Generates a psychrometric plot with Nwb lines of constant wet bulb whose values are specified in the list Twb1 Twb2 ... and Nv lines of constant specific volume whose values are specified in the list v1 v2 ... The pressure used to generate the plot is Pval. The optional parameter Mollier specifies that the plot should be generated using the Mollier format.
Rename Plot$ NewPlot$	Renames the plot Plot$ to NewPlot$.

Table D-4: Macro commands related to plots (continued)

Macro Command	Description
Repeat Command(s) Until(Conditional)	Executes the commands in the list Command(s) until the conditional test Conditional is true.
Reset X, Y, ...	Resets the status of the variables in the list X, Y, ... to unassigned.
ResetGuesses	Resets the guess values for the variables in the Equations Window to their default values.

Table D-5: Macro Commands Related to Tables.

Macro Command	Description
AlterValues Table$ Var Rows=RowStart..RowEnd First=FirstValue Last=LastValue {Inc=IncValue} {Mult=MultValue} {Clear}	Alters the values in the Parametric Table Table$ for the column corresponding to variable Var and the rows between RowStart and RowEnd. The values begin at FirstValue and are varied linearly to LastValue. Alternatively, the increment can be set with IncValue or the multiplier set with MultValue. The Clear keyword clears the selected cells.
ClearLookupColumn Table$ Column$ {ColumnNumber} RowStart..RowEnd	Clears the values in the Lookup Table Table$ within the column with name Column$ (or number ColumnNumber) between rows RowStart and RowEnd. If the parameters RowStart and RowEnd are not included then the entire column will be cleared.
Copy {/H} ArraysTable {Table$} {RX CY : RXX CYY}	Copies the Arrays Table to the clipboard. The /H header causes the headers and units to be copied. The Table$ parameter allows the name of the Arrays Table to be specified; if Table$ is not provided then the 'Main' Arrays Table will be copied. The range of values can be specified according to RX CY : RXX CYY; if these parameters are not provided then the entire table will be copied.
Copy {/H} IntegralTable {RX CY : RXX CYY}	Copies the Integral Table to the clipboard. The /H header causes the headers and units to be copied. The range of values can be specified according to RX CY : RXX CYY; if these parameters are not provided then the entire table will be copied.
Copy {/H} LookupTable Table$ {RX CY : RXX CYY}	Copies the Lookup Table Table$ to the clipboard. The /H header causes the headers and units to be copied. The range of values can be specified according to RX CY : RXX CYY; if these parameters are not provided then the entire table will be copied.
Copy {/H} ParametricTable Table$ {RX CY : RXX CYY}	Copies the Parametric Table Table$ to the clipboard. The /H header causes the headers and units to be copied. The range of values can be specified according to RX CY : RXX CYY; if these parameters are not provided then the entire table will be copied.
DeleteLookup Table$ {TableNumber}	Deletes the Lookup Table Table$. Alternatively, the TableNumber parameter specifies the number of the Lookup Table tab.
InsertLookupColumn Table$ After {'ColumnName\Units' ColumnFormat}	Inserts a new column in Lookup Table Table$ at position After. The optional 'ColumnName\Units' parameter sets the name and units and the optional ColumnFormat parameter specifies the display format.
InsertLookupRows Table$ After {NRows}	Inserts a row in Lookup Table Table$ at position After. The optional NRows parameter specifies the number of rows to insert.
InsertParametricRows Table$ After {NRows}	Inserts a row in Parametric Table Table$ at position After. The optional NRows parameter specifies the number of rows to insert.
Lookup[Table$, Row, Column]=Value	Sets the cell Row, Column in Lookup Table Table$ to Value. Note that Value cannot be an arithmetic expression.
LookupColInfo Table$ Column Name$ {Units$}	Sets the name of column Column in Lookup Table Table$ to Name$. Optionally, the units are set to Units$.
NewLookup Table$ Rows=NRows Cols=NCols	Creates a new Lookup Table with name Table$. The number of rows is NRows and the number of columns is NCols.

NewTable Table$ Rows=NRow X Y Z ...	Creates a new Parametric Table with name Table$ and NRow rows with columns for variables in the list X Y Z …
OpenLookup LookupFile$ {/Format FormatFile$}	Opens the lookup file LookupFile$. If ? is provided then a dialog will appear to allow the file to be selected. The optional FormatFile$ can be used to specify the format that should be used to read the file.
Parametric[Table$, Row, Col (or Col$)]=Value	Sets the cell in row number Row and column number Col in table Table$ to Value. The column can also be specified by the name of the column Col$.
Paste Lookup Table$ Row Col	Pastes the contents of the clipboard in the Lookup Table Table$ at the position specified by the parameters Row and Col.
Paste Parametric Table$ R1 C1	Pastes the contents of the clipboard in the Parametric Table Table$ at the position specified by the parameters Row and Col.