CHAPTER 3
FORMULATION OF COMPONENT MODELS

3.1 INTRODUCTION

As indicated in Chapter 1, one of the first steps in computer simulation is to formulate mathematical models of the components of the system. TRNSYS requires that the models be programmed with a particular methodology such that TRNSYS can correctly identify the INPUTS, OUTPUTS and PARAMETERS and thereby perform the component interconnections as directed by the user.

This chapter presents the general TRNSYS rules and regulations governing the programming of the FORTRAN subroutines representing the mathematical models of components. Complete descriptions of components in the standard TRNSYS library are presented in Chapter 4.

3.2 CONCEPTS FOR MODELING COMPONENTS

To model a component of a system, the user must be able to adequately describe the performance of the component mathematically and identify the following:

i) INPUTS
ii) OUTPUTS
iii) PARAMETERS
iv) DERIVATIVES (time dependent variables)

As an example, consider a simple heater which raises the variable inlet temperature of a flowing fluid to some preset temperature T_{set}. The heater is assumed to be 100% efficient (the energy input to the heater is the energy delivered to the fluid) with no thermal storage effects.

Writing an energy balance on the fluid of inlet temperature, T_{in}, at a mass flow rate $\dot{m}$:

$$\dot{Q} = \dot{m} C_p (T_{out} - T_{in}) \qquad (3.2.1)$$

For the INPUTS T_{in} and $\dot{m}$, the OUTPUTS of interest may be $\dot{Q}$, the instantaneous heating rate, $\dot{m}$ and the outlet temperature, Tout. The PARAMETERS characterizing the heater will be Tset and C_{pf}, the specific heat of the fluid being heated. Note that $\dot{m}$ and T_{out} are included as OUTPUTS so that this information is available to other components. The OUTPUT $\dot{Q}$ may be of interest in order to determine the maximum heating rate for a given set of conditions (perhaps to provide a design point for selection of actual heater capacity). It may also be desirable to integrate $\dot{Q}$ over the period of operation so that the actual amount of energy used, and hence its cost, can be determined. The information flow diagram of this idealized heater is represented in Figure 3.2.1.

Note that in this example T_{set} has been chosen as a PARAMETER, but it could, if necessary, be considered as a variable INPUT from another component. This would entail writing

a slightly different FORTRAN subroutine as will become evident in the following sections, and underlines the importance of appreciating what function the component is to perform in a particular system.

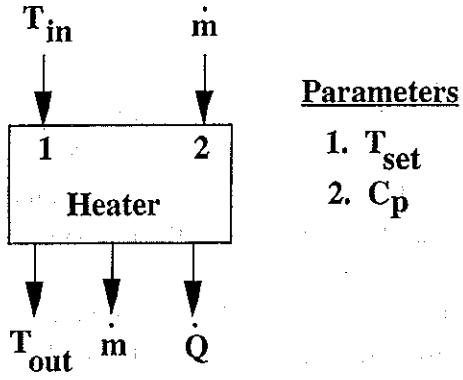

Figure 3.2.1: Information Flow Diagram for Idealized Heater

A special consideration applies to the formulation of components that incorporate some kind of internal "on-off" control decision so that a solution can be obtained without oscillation problems. Examples of these components are differential controllers, thermostats, and models of hardware with "built-in" controls. For further information on these types of components, see Section 4.4.

3.3 COMPONENT SUBROUTINES

Every component subroutine must have the name TYPEn, where n is an integer number between 1 and 99, excluding 24, 25, 26, 27, 28, and 29. Users formulating their own components should be careful to choose a TYPE number not currently in the TRNSYS library. Typically, type numbers 66 and above are reserved for user written component routines.

The first line in every component subroutine must be of the following form and order:

```
SUBROUTINE TYPEn(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*)
```

For a TYPE 35 component, n above would be 35.

In the subroutine parameter sequence shown above, the following variables are used

TIME	-	the simulation time
XIN	-	the double precision array containing the component INPUTS
OUT	-	the double precision array which the component fills with its appropriate OUTPUTS
T	-	the real array containing the dependent variables for which the derivatives are evaluated in the particular model
DTDT	-	the real array containing the derivatives of T which are evaluated by the model

PAR - the real array containing the PARAMETERS which characterize the component
INFO - the integer array described in Section 3.3.3
ICNTRL - the integer array described in Section 3.3.4
* - the alternate return described in Section 3.3.5

As XIN, OUT, T, DTDT, PAR, ICNTRL and INFO are arrays, an appropriate DIMENSION statement must be included in the subroutine to signify that they are arrays. As array space has already been allocated in the main program TRNSYS, the size of each DIMENSION statement within the particular routine is immaterial (to most compilers) but it is recommended that the values corresponding to the actual number of each in the subroutine be used: i.e., if a component model has 5 INPUTS, 2 OUTPUTS, 1 dependent variable (and hence 1 DERIVATIVE), and 4 PARAMETERS, the initialization statements should be,

```
DOUBLE PRECISION XIN, OUT
REAL T, DTDT, PAR
INTEGER INFO, ICNTRL
DIMENSION XIN(5),OUT(2),T(1),DTDT(1),PAR(4),INFO(15)
```

If a particular component model has, for example, no PARAMETERS, then PAR can be left out of the DIMENSION statement. The INFO and ICNTRL arrays must be dimensioned if they are used. Remember also that additional declaration statements may be necessary to allocate storage space for calculations within the particular subroutine, according to the rules of FORTRAN.

3.3.1 ORDERING OF PARAMETERS, INPUTS AND OUTPUTS

The user must choose the sequential order of the PARAMETERS, INPUTS, and OUTPUTS from the component model as indicated in Chapter 1. For example, with the idealized heater considered previously, the first and second PARAMETERS may be chosen as T_{set} and C_p respectively. Thus, choosing FORTRAN mnemonics TSET and CP as T_{set} and C_p, the statements:

```
TSET = PAR(1)
CP = PAR(2)
```

will assign the values contained within PAR(1) and PAR(2) to TSET and CP. Note that once it has been decided that TSET is the first PARAMETER and CP is the second, this is the order in which the user must assign values of these PARAMETERS in the TRNSYS input file unless he or she wishes to change them by altering the FORTRAN statements.

With TRNSYS 14, the XIN and OUT arrays should be double precision.
INFO is dimensioned to 15 with release of version 14.

Similarly, an order is assigned to the variable INPUTS as

TIN = XIN(1)
FLOW = XIN(2)

where FLOW is the mnemonic for $\dot{m}$.

The OUTPUTS are also ordered sequentially and for the heater example, we may choose

OUT(1) = TSET
OUT(2) = FLOW
OUT(3) = QDOT

Thus, a component subroutine could be written for an idealized heater as follows. (Note that this is only an example and does not represent the more complex auxiliary heater modeled in Chapter 4.)

SUBROUTINE TYPE 35 (TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*)

```
C     THIS EXAMPLE ILLUSTRATES THE STRUCTURE OF THE
C     COMPONENT SUBROUTINE AND IS BASED UPON AN IDEALIZED
C     HEATER CONSIDERED PREVIOUSLY.  IT IS ASSUMED TO BE A
C     TYPE 35 COMPONENT AS DESCRIBED BY THE DESIGNATION
C     TYPE 35

      DOUBLE PRECISION XIN, OUT
      REAL T, DTDT, PAR
      INTEGER INFO
      DIMENSION XIN(2), OUT(3), PAR(2), INFO(15)

      NP=2
      NI=2
      CALL TYPECK(1,INFO,NI,NP,0)

      TSET = PAR(1)
      CPF = PAR(2)
C
      TIN = XIN(1)
      FLOW = XIN(2)
C
      QDOT = FLOW*CPF*(TSET-TIN)
C
      OUT(1) = TSET
      OUT(2) = FLOW
      OUT(3) = QDOT
C
      RETURN 1
      END
```

Referring once more to Figure 3.2.1, the information flow diagram representation of the idealized heater, the sequential numbering of the INPUT, OUTPUTS and PARAMETERS should now be self evident.

3.3.2 NUMERICAL SOLUTION OF DIFFERENTIAL EQUATIONS

TRNSYS Versions 11.1 and subsequent versions provide two methods for solving differential equations which may occur in component models: an approximate analytical solution using the subroutine DIFFEQ (see Section 1.10.1 and Section 3.4.4), and a numerical solution using one of three numerical methods (see Sections 1.10.2 and 2.2.8). The analytical method is recommended whenever practical (see Section 1.10.1), since the numerical method generally requires shorter timesteps and more computation for comparable accuracy and numerical stability. However, users intending to utilize the new TRNSYS equation solver (SOLVER 1) should use the DIFFEQ subroutine whenever possible.

Numerical solutions to differential equations, when required, can be obtained through the DTDT and T arrays in the subroutine calling sequence. To use a very simple example, if the equation

$$\dot{Q}_i = C\frac{dT_1}{dt} \qquad (3.2.2.1)$$

were part of the mathematical model and the DERIVATIVE of the temperature T_1 with respect to time t were to be integrated for a known heat flow rate $\dot{Q}$ and a constant value C, it would appear in the subroutine in a manner similar to that shown below.

```
.
.
DTDT(1) = QDOT / C
.
.
OUT(2) = T(1)
.
.
```

DTDT(1) is the differential dT_1/dt, and T(1) is the result of integrating DTDT(1). In this example T_1 is also desired to be the second OUTPUT of the sequentially ordered OUTPUT list. Note that the subscripts of OUT and T do not have to be the same, but that T(n) will always be the result of integrating DTDT(n).

A DERIVATIVES control statement followed by an initial values line must be included among the component control statements for any model which uses the DTDT array. The presence of a DERIVATIVES command signals TRNSYS to integrate derivatives placed in the DTDT array and to return results in the T array, repeating the procedure at each timestep until either the appropriate error TOLERANCE or the iteration LIMIT is reached. The initial values of the dependent variables (i.e., T(n)) are available to the component subroutine, if needed, through the T

array at the first call of the simulation (i.e., when INF0(7) = - 1; see Section 3.3.3).

3.3.3 USE OF THE INFO ARRAY

The INFO array conveys to the component subroutines (TYPE1, TYPE2, etc.) and to the TRNSYS kernel subroutine EXEC information about the current UNIT. There are many optional uses for the INFO array. A user formulating his or her own TYPE routines should be aware of a few of them. In user-formulated routines, INFO (6) should be set equal to the number of OUTPUTS.

INFO(9) should be set equal to 0 if the TYPE's OUTPUTS depend only on the values of inputs and not explicitly on time (see below), equal to 2 if the TYPE is to be called only at the end of each timestep after the call to integrators and printers, and equal to 3 if the TYPE is to be called only after convergence has been attained and before the call to integrators and printers (see below).

INFO allows TYPE subroutines to check that the TRNSYS user has specified the proper number of PARAMETERS, INPUTS, and DERIVATIVES, and allows unnecessary calculations to be eliminated during a simulation.

The contents of the INFO array for a given unit are as follows:

INFO(1)	-	UNIT number
INFO(2)	-	TYPE number
INFO(3)	-	number of INPUTS
INFO(4)	-	number of PARAMETERS
INFO(5)	-	number of DERIVATIVES
INFO(6)	-	number of OUTPUTS
INFO(7)	-	number of iterative calls to the UNIT in the current timestep
INFO(8)	-	total number of calls to the UNIT in the simulation
INFO(9)	-	indicates whether TYPE depends on the passage of time (see below)
INFO(10)	-	used to allocate storage (see Section 3.5)
INFO(11)	-	indicates number of discrete control variables (see Section 3.3.4)
INFO(12-15)	-	reserved for future considerations

The following points should be noted:

(i) INFO(1) thru INFO(5) are set by TRNSYS from information in the input file.

(ii) INFO(6), the number of OUTPUTS: A minimum of 20 OUTPUTS are reserved for each UNIT. If a user written component has less than 20 outputs, INFO (6) should.be set to the correct number of outputs to provide an accurate trace. If a user written component has more than 20 outputs, INFO (6) must be set to the correct value to reserve the additional outputs.

(iii) INFO(7) indicates how many times the UNIT has been called in the current timestep. This information is very useful for building stability into user written controller routines (see Section 4.4 and Appendix I) and for eliminating unneeded calculations at each timestep.

The following notation is used:

-1	-	initial call in simulation for this UNIT
0	-	first call in timestep for this UNIT.
+1	-	first iterative call (second call) in timestep for this UNIT.
+2	-	second iterative call (third call) in timestep for this UNIT.
+n	-	nth iterative call (n+1th call) in timestep for this UNIT.

(iv) On the first call in the simulation to a UNIT, INF0(7) = -1, INFO(8) = 1. Either of these conditions can be used to identify the first call of the simulation for purposes of error checking, assigning UNIT specific storage space, etc.

(v) INFO(9) is initialized to 1 for every TYPE routine in the main program at the beginning of the simulation. It may be reset to 0, 2, or 3 inside the individual TYPE routines. INFO(9) = 1 signifies to EXEC that the TYPE's OUTPUTS depend upon the passage of TIME and must therefore be called at least once every timestep even if the values of inputs do not change. A user formulating his or her own TYPE routines should set INFO(9) = 0 if the TYPE's outputs depend only upon the input values and not explicitly upon time. Failure to do this, however, will result only in some unnecessary calls to the TYPE routine. When INFO(9) = 0, the TYPE routine is called only if its INPUTS have changed beyond a user-set tolerance since the previous call. The user should set INFO(9) = 2 if the TYPE should be called only at the end of each time step (i.e. after all other components have converged) and after the call to printers and integrators. User-written output subroutines and controllers are situations for which this would be appropriate. The user should set INFO(9) = 3 if the TYPE should be called only at the end of each time step (i.e. after all other components have converged) and before the call to printers and integrators. User-written statistic subroutines that send their output to integrators is a situation where this may be appropriate.

3.3.4 USE OF THE ICNTRL ARRAY

The ICNTRL array, which is new to version 14, conveys to the component subroutines (TYPE1, TYPE2, etc.) and to the TRNSYS equation processor, the states of the controller variables. With the release of TRNSYS 14, a new control strategy was designed to eliminate the problems caused by previous TRNSYS controllers "sticking" in order to solve the system of equations (see Sections 2.2.10 and 4.4). With TRNSYS 14, the TRNSYS executive program, rather than the component models, directly controls discrete variables. At the start of each time step, the values of all discrete variables are known. TRNSYS 14 forces these values to remain at

their current state until a converged solution to the equations is obtained or an iteration limit is encountered. During these calculations, the component models calculate the 'desired' value of the discrete variables but they do not actually change the settings. After a converged solution is obtained, or the maximum number of iterations is exceeded, TRNSYS 14 compares the current discrete variable values with the 'desired' values. If they do not differ and a converged solution has been obtained, the calculations are completed for the time step. Otherwise, TRNSYS 14 changes the values of the discrete variables to the 'desired' setting and repeats this process, taking care to not repeat the calculations with the same set of discrete variables used previously.

To incorporate the new control strategy into user-written component routines, the ICNTRL and INFO arrays must be set up correctly. INFO(11) should specify the number of discrete control variables in the component routine. ICNTRL(2*(i-1)+1) and ICNTRL(2i) will contain the control state at the previous iteration for controller i in the component model. (controlled by the TRNSYS processor) and the desired control state at this iteration respectively.

The ICNTRL array is an integer array and must therefore pass and receive only integer values. The assumption in TRNSYS is that an ICNTRL value of 1 implies the control signal was enabled while an ICNTRL value of 0 denotes a disabled control signal.

For example, a user wishes to describe an on/off controller with hysteresis effects (control signal depends on previous control signal). The component formulation would be of the following format:

```
UDB=PAR(1)              ! UPPER DEAD BAND TEMP. DIFFERENCE
LDB=PAR(1)              ! LOWER DEAD BAND TEMP. DIFFERENCE
   .
INFO(11)=1              ! 1 CONTROL VARIABLE)
   .
   .
TH=XIN(1)               ! UPPER INPUT TEMPERATURE
TL=XIN(2)               ! LOWER INPUT TEMPERATURE
   .
   .
STATEOLD=ICNTRL(1)      ! CONTROL STATE AT PREVIOUS SOLUTION
STATENEW=STATEOLD

IF(STATEOLD.EQ.1) THEN
      IF((TH-TL).LT.LDB) STATENEW=0
ELSE
      IF(TH-TL).GT.UDB)) STATENEW=1
ENDIF
ICNTRL(2)=STATENEW
   .
```

At every iteration, the value of ICNTRL(1) holds the controller state at the previous solution while ICNTRL(2) holds the desired controller state at this iteration. The TRNSYS processor will check to see if ICNTRL(1) is equal to ICNTRL(2) at convergence. If the values are

not equal at convergence, ICNTRL(1) will be set to the desired controller state at convergence (ICNTRL(2)) and the process will be repeated.

3.3.5 USE OF THE ALTERNATE RETURN

TRNSYS 14 checks to make sure that all routines required by the TRNSYS input file are included in the TRNSYS executable file by utilizing the FORTRAN alternate return. In previous TRNSYS versions run with compilers allowing unresolved externals, unlinked subroutines were not checked and were often the source of strange errors. The TRNSYS routines and all user-written routines which call other subroutines should include an alternate return. The following example should help the user understand the workings of the alternate return.

```
** CALLING PROGRAM **
      CALL TYPE2(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*101)
100         WRITE(*,*) 'ERROR - TYPE 2 IS NOT LINKED !'
            CALL MYSTOP(1001)
            RETURN 1
101   CONTINUE

      .
      .

** CALLED SUBROUTINE **
      SUBROUTINE TYPE2(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*)

      .
      RETURN 1
```

The calling program will call the TYPE 2 subroutine and return to line 101 when the subroutine calculations are completed due to the RETURN 1 line in the TYPE 2 subroutine. If the RETURN 1 is missing from the TYPE 2 subroutine or the subroutine is not linked, the program will return to line 100 of the calling program and the error message will be flagged.

All user written routines should contain an asterisk (*) at the end of the subroutine designation and 'RETURN 1's every place a RETURN is desired. Failure to do so will cause the unlinked subroutine error message to be displayed and the program to terminate.

All user-written routines that call other subprograms should also include the alternate return feature to check for unlinked subprograms. To activate the alternate return:

1) Call the subprogram with an asterisk-line number combination at the end of the subprogram designation.
2) Specify the line number given in the step above as a CONTINUE line
3) Between the subprogram call and the CONTINUE statement, place an error message and a CALL MYSTOP (1001) command - a call may instead be placed to the LINKCK subroutine (see Section 3.4.11)
4) Make sure all RETURN's in the subprogram are replaced with a RETURN 1 statement Refer to the example given or a FORTRAN manual for more details.

3.4 UTILITY SUBROUTINES

TRNSYS includes some subroutines which perform general utility functions for the TYPE routines. A user writing his or her own TYPE routines or modifying existing TYPE routines may wish to take advantage of some of these built-in capabilities. Subroutines TYPECK, DATA, TALF, DIFFEQ, ENCL, VIEW, TABLE, INVERT (DINVRT), FIT (DFIT) PSYCH, LINKCK RCHECK, and FLUIDS are such routines. **NOTE**: The **TYPECK** subroutine should be used in **ALL** user written components.

3.4.1 SUBROUTINE TYPECK

The purpose of TYPECK is to perform some simple checks on the TRNSYS input file to ensure the proper number of INPUTS, PARAMETERS, and DERIVATIVES have been specified for the given component, and to terminate the simulation after compilation of the component input file with appropriate error messages if errors are found.

The form of a TYPECK call is:

CALL TYPECK(IOPT,INFO,NI,NP,ND)

where

IOPT - indicates to TYPECK what kind of check to perform:

- <0 the compilation will be terminated immediately after printing an explanatory error message.
- >0 the simulation will be terminate after all UNITS have been called once or 20 error messages have been issued.
- 1 or -1 check the number of user furnished INPUTS, PARAMETERS, and DERIVATIVES from the INFO array against the values of NI, NP, and ND. An error message will be printed only if TYPECK finds an error.
- 2 or -2 TYPE routine has found unspecified error and wants TYPECK to flag it.
- 3 or -3 TYPE routine has found a bad INPUT or number of INPUTS and wants TYPECK to flag it.
- 4 or -4 TYPE routine has found a bad PARAMETER or number of PARAMETERS and wants TYPECK to flag it.
- 5 or -5 TYPE routine has found a bad DERIVATIVE or number of DERIVATIVES and wants TYPECK to flag it.
- 6 TYPE routine has determined that the maximum number of UNITS of that TYPE has been exceeded and wants TYPECK to flag it and stop the simulation.

INFO - TYPE's INFO array containing pertinent bookkeeping data on the UNIT (see Section 3.3.3).

NI - correct number of INPUTS

NP - correct number of PARAMETERS

ND - correct number of DERIVATIVES

3.4.2 SUBROUTINE DATA

Subroutine DATA is available to read user supplied data from FORTRAN logical units and interpolate this data during the course of the simulation. For each data set, up to five dependent (Y) functions may be specified in terms of up to 3 independent (X) variables. In a simulation only 5 data sets (i.e. 5 different logical units) may be specified. Examples of existing routines that use DATA are the solar collector (TYPE 1) and the conditioning equipment (TYPE 44).

The form of a call to DATA is:

CALL DATA (LU, NIND, NX, NY, X, Y, INFO,*N)

where

LU	is the FORTRAN logical unit number through which the performance data is accessed - see ASSIGN statement (Section 2.2.11).
NIND	is the number of independent variables ($1 \leq NIND \leq 3$)
NX	is an array dimensioned to NIND containing the number of values of each independent variable to be read from LU. NX(1), NX(2), and NX(3) are the number of values of X_1, X_2, and X_3, respectively. The following restrictions apply: $NX(1) \leq 10$, $NX(2) \leq 5$, $NX(3) \leq 5$.
NY	is the number of dependent (Y) values associated with each set of independent (X) values ($1 \leq NY \leq 5$)
X	is an array dimensioned to NIND containing the values of the independent variables for which interpolated Y values are desired
Y	is an array of dimension NY containing interpolated values of the dependent variables
INFO	is the calling TYPE's INFO array
N	is the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

The data is read from logical unit LU at the start of the simulation. Thereafter, DATA linearly interpolates for Y values given X values. If DATA is called with an X out of range of the supplied data, then the closest specified value is used. DATA does not extrapolate beyond the user supplied data.

The data supplied on the logical unit is read in free format. If NIND equals 3 and NX(3) is greater than 1, then NX(3) values of the third independent variable X_3 are read first. Similarly, if $NIND \geq 2$ and $NX(2) > 1$, then a set of NX(2) values of the second independent variable, X_2 is read. Next, NX(1) values of the primary independent variable, X_1, are read. The values of the independent variables must be in ascending order, but need not be at regular intervals. The Y values are read beginning with values corresponding to the smallest values of X_1, X_2, and X_3. NX(1) sets of NY values of Y are read for each value of X2. NX(2) sets of NX(1) . NY values of Y are read for each value of X_3. In all, if 3 independent variables are employed, then NX(3) * NX(2) * NX(1) * NY values of Y must be specified.

Example:

A user wishes to write a model for a water-to-water heat pump whose performance depends upon the inlet flowstream temperatures to both the evaporator (T_e) and condenser (T_c). DATA is to be used to evaluate both the capacity and COP. The experimental data is shown in Table 3.4.2-1.

TABLE 3.4.2.1

	$T_c = 20°C$			$T_c = 50°C$	
T_e (°C)	Capacity (kJ/hr)	COP	T_e (°C)	Capacity (kJ/hr)	COP
10	35000	2.47	10	21000	1.73
20	41000	2.80	20	25000	1.96
30	49500	2.99	30	29000	2.09

The data is to be read from logical unit 10. Three values of T_e and two values of T_c will be supplied. The call to DATA within the TYPE routine might appear as

```
      .
      .
      NX(1) = 3
      NX(2) = 2
      X(1)  = TE
      X(2)  = TC

      CALL DATA (10, 2, NX, 2, X, Y, INFO,*101)
      CALL LINKCK('TYPE74','DATA ',3,DUM)
101   CONTINUE

      CAP   = Y(1)
      COP   = Y(2)
      .
      .
```

The data accessed through logical unit 10 would be as follows:

```
20.  50                 (Tc values)
10.  20.  30.           (Te values)

35000.  2.47            (Capacity and COP
41000.  2.80            for Tc=20°C,
49500.  2.99            Te=10, 20, 30°C)

21000.  1.73            (Capacity and COP
25000.  1.96            for Tc=50°C,
29000.  2.09            Te=10, 20, 30°C)
```

3.4.3 FUNCTION TALF

General Description

Function subroutine TALF calculates the transmittance-absorptance product ($\tau\alpha$) of a solar collector as a function of angle of incidence of radiation and collector construction. The transmittance τ of one or more sheets of glass or plastic can be obtained by specifying an absorptance of 1.

A call to function TALF is of the form

TAUALF = TALF(N,THETA,XKL,REFRIN,ALPHA,RHOD)

The function arguments are interpreted as follows:

N	-	number of covers
THETA	-	angle of incidence (degrees)
XKL	-	product of cover thickness and extinction coefficient
REFRIN	-	index of refraction
ALPHA	-	absorber surface absorptance
RHOD	-	reflectance of the N covers to diffuse radiation

If the value of RHOD is not known, then TALF can calculate it: When the sixth argument in the function reference is less than zero, the reflectance of the N covers to diffuse radiation is determined. The variable used as the sixth argument in the call to TALF is then set to this value so that on subsequent calls to TALF the value of RHOD may be supplied.

Due to FORTRAN considerations, functions can not be checked for unlinked subroutines. Use the LINKCK subroutine to warn the user that this function must be linked to the TRNSYS executable (see Section 3.4.11)

Mathematical Description

The function subprogram TALF calculates the transmittance-absorptance product ($\tau\alpha$) of a solar collector for a given angle of incident radiation. The collector is described by the number of glazings, N, each with refractive index n_g and extinction length kL, and the absorptance of the collector plate α. The model uses Fresnel's equation for the specular reflectance at a planar interface (assuming unpolarized incident radiation) and accounts for multiple reflections. The transmittance for diffuse radiation is approximated as the transmittance for specular radiation at an incident angle of 60°. Fresnel's equations for reflectance at a planar interface for perpendicular and parallel polarized radiation are the following:

perpendicular

$$\rho_1 = \frac{\sin^2(\theta_1 - \theta_2)}{\sin^2(\theta_1 + \theta_2)} \qquad (3.4.3.1)$$

parallel

$$\rho_2 = \frac{\tan^2(\theta_1 - \theta_2)}{\tan^2(\theta_1 + \theta_2)} \qquad (3.4.3.1)$$

The angles θ_1 and θ_2 are related by Snell's Law with the index of refraction for air of unity.

$$\sin\theta_2 = \frac{\sin\theta_1}{n_g} \tag{3.4.3.3}$$

The transmittance and reflectance of a single glazing for each component of polarization, including absorption and multiple reflections within the glazings, can be derived to give the following [1,2].

$$T_{i,1} = \frac{\tau_a (1 - \rho_i)^2}{1 - \tau_a^2 \rho_i^2} \tag{3.4.3.4}$$

$$R_{i,1} = \rho_i \left[1 + \frac{\tau_a^2 (1 - \rho_i)^2}{1 - \tau_a^2 \rho_i^2} \right] \tag{3.4.3.5}$$

The internal transmittance, τ_a, is a function of the extinction length and the transmission angle.

$$\tau_a = \exp\left(\frac{-kL}{\cos\theta_2}\right) \tag{3.4.3.6}$$

The transmittance and reflectance of N parallel plates, of the same material and thickness, can be determined by the following recursive formulae.

$$T_{i,N} = \frac{T_{i,1} T_{i,N-1}}{1 - R_{i,1} R_{i,N-1}} \tag{3.4.3.7}$$

$$R_{i,N} = R_{i,N-1} + \frac{R_{i,1} T_{i,N}}{1 - R_{i,1} R_{i,N-1}} \tag{3.4.3.8}$$

The transmittance-absorptance product of the collector plate and glazing assembly can then be calculated as

$$(\tau\alpha) = \frac{(T_{1,N} + T_{2,N})\,\alpha}{2\left(1 - (1 - \alpha) R_D\right)} \tag{3.4.3.9}$$

where R_D is the reflectance of the N covers to diffuse radiation. R_D is calculated as the reflectance for specular radiation at 60° incident angle.

3.4.4 SUBROUTINE DIFFEQ

The subroutine DIFFEQ provides analytical solutions to first-order linear differential equations that can be written as

$$\frac{dT}{dt} = aT + b \qquad (3.4.4.1)$$

Users intending to utilize the new TRNSYS solver should use the DIFFEQ subroutine discussed here instead of the optional DTDT array discussed in section 3.3.2.

The form of the call to DIFFEQ is:

CALL DIFFEQ(TIME, AA, BB, TI, TF, TBAR)

where

TIME	-	current value of time
AA	-	is the a coefficient of the differential equation
BB	-	is the b coefficient of the differential equation
TI	-	is the value of the dependent variable (T) at the beginning timestep
TF	-	is the value of the dependent variable (T) at the end of the timestep
TBAR	-	is the average value of the dependent variable over the timestep.

The DIFFEQ subroutine does not require link checking since it is considered to be a kernal routine; those routines which must be present to successfully run TRNSYS.

DIFFEQ solves differential equations at each call based on parameters and the current set of inputs as described in Section 1.10.1. If the inputs have not converged, the iteration proceeds by direct substitution. Additional notes on the analytical solution of differential equations are as follows:

1) DIFFEQ requires the arguments AA, BB, and TI, while returning TF and TBAR. For calls to DIFFEQ at the start time of the simulation (TIME= TIME0), the initial condition TI is returned for both TF and TBAR.

2) It is necessary to save the final values of dependent variables each timestep as initial conditions for the following interval. This can be done using the OUT or S arrays (see Section 3.5).

3) Inputs will often be a function of the dependent variable of the differential equation. For instance, collector outlet temperature which might be an input to a storage component, might also be a function of the storage temperature. In order to get a good estimate of the average inputs over each interval, TBAR should be set as an output if required as an input of other components. To be consistent with this formulation and the rectangular integration provided by TYPE 24, etc., any outputs that are rates and are expressed as a function of the dependent variable T should be evaluated at TBAR.

As an example of a model formulation, suppose that one wishes to study the temperature response of a one-node house to ambient conditions. The instantaneous energy balance for this situation is given as

$$C\frac{dT_R}{dt} = -(UA)_L(T_R - T_a) \qquad (3.4.4.2)$$

This can be written in the form of equation (3.4.4.1) if

$a = -(UA)_L/C$

$b = (UA)_L T_a/C$

The following code would establish initial conditions, solve the differential equation for the final and average temperatures, and determine the average energy loss rate from the house for each time interval.

```
.
.
If(INFO(7).EQ.0) S(ISTORE + 1) = S(ISTORE + 2)
TI = S(ISTORE + 1)
AA = -UAL/C
BB = UAL*TA/C
CALL DIFFEQ (TIME,AA,BB,TI,TF,TBAR)
S(ISTORE + 2) = TF
QLOSS = UAL*(TBAR - TA)
.
.
.
```

For this particular example, the component would not be called during any timestep after its inputs have converged within the algebraic error tolerance. There is no need for a differential equation convergence check, since the solution will always be consistent with the current set of inputs. A DERIVATIVES control statement should <u>not</u> be used for the component in the simulation input file; the initial value of the dependent variable at the beginning of the simulation should be supplied to the subroutines as a PARAMETER. In this respect the example is similar to TYPES 12, 21 and 23.

3.4.5 SUBROUTINE ENCL

General Description

Subroutine ENCL calculates view factors between all surfaces of a rectangular parallelpiped. Up to 9 windows or doors may be located on any of the 6 wall surfaces. This routine is currently used by the TYPE 19 zone model.

A call to ENCL is of the form

CALL ENCL (SPAR, WPAR, FV, INFO,*N)

where

SPAR	-	an array of 10 values containing the 10 zone geometry parameters described in the TYPE 19 documentation (parameters 2-11 of geometry Mode 1) (see Section 4.8.3).
WPAR	-	an array dimensioned to 54, containing the complete list of window or door geometry parameters described in the TYPE 19 section for geometry Mode 1 (see Section 4.8.3).
FV	-	an array containing the view factors for all surfaces as calculated by the subroutine
INFO	-	The calling TYPE's INFO array
N	-	is the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

Subroutine ENCL outputs a table of the view factors between surfaces. The surface numbers are as specified in the SPAR array.

Mathematical Description

Subroutine ENCL utilizes function subroutine VIEW to obtain view factors between continuous rectangular surfaces. Reciprocity is also used to reduce the number of computations. In order to determine view factors between surfaces that contain windows or doors, it is necessary to perform some view factor algebra.

Consider the general case of m windows located on a surface i and n windows on a surface j as shown in Figure 3.4.5.1.

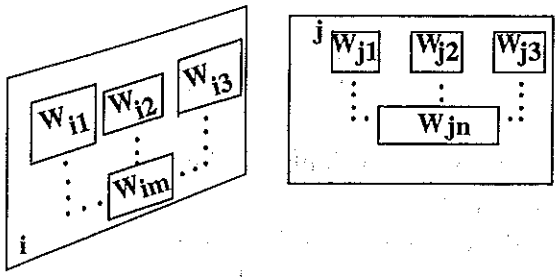

Figure 3.4.5-1

In general, the view factor between i and j is

$$F_{i\text{-}j} = F_{i\text{-}(j+W_{ji}+\ldots W_{jn})} - \sum_{k=1}^{n} F_{i\text{-}(W_{jk})} \qquad (3.4.5.1)$$

where

$$F_{i-(j+W_{j1}+\ldots W_{jn})} = \frac{(A_j + A_{W_{j1}} + \ldots A_{W_{jn}})\, F_{(j+W_{j1}+\ldots W_{jn})-i}}{A_i} \qquad (3.4.5.2)$$

$$F_{(j+W_{j1}+\ldots W_{jn})-i} = F_{(j+W_{j1}+\ldots W_{jn})-(i+W_{i1}+\ldots W_{im})} - \sum_{k=1}^{m} F_{(j+W_{j1}+\ldots W_{jn})-W_{ik}} \qquad (3.4.5.3)$$

$$F_{i-W_{jk}} = \frac{A_{W_{jk}} F_{(W_{jk})-i}}{A_i} \qquad (3.4.5.4)$$

$$F_{W_{jk}-i} = F_{W_{jk}-(i+W_{i1}+\ldots W_{im})} - \sum_{l=1}^{m} F_{W_{jk}-W_{il}} \qquad (3.4.5.5)$$

The variables W_{i1} through W_{im} are the surface numbers associated with windows on wall i. Likewise, W_{j1} through W_{in} refer to windows on wall j. Thus, $(i+W_{i1}+\ldots W_{in})$ and $(j+W_{j1}+\ldots W_{jm})$, each denote surfaces that are collections of individual surfaces. The variable A refers to the area of a surface whose number is used as a subscription.

3.4.6 FUNCTION VIEW

General Description

Function subroutine VIEW calculates the VIEW factor between two planer rectangles of any orientation. For common orientations, analytical expressions are used.

A call to VIEW is of the form

F21 = VIEW(ITYPE,A,B,C,D,E,F,G,H,X3,Y3,Z3,NA1,NA2)

The variable ITYPE defines the type of orientation between the two surfaces. There are five possibilities. The arguments A, B, C, D, E, F, G, H, X3, Y3, and Z3 are dimensions or coordinates that depend upon the orientation chosen. Not all of these variables are used for each orientation. The possible orientations and required dimensions are shown in Figures 3.4.6.1 to 3.4.6.5. The VIEW factor from surface 2 to 1 is determined by the routine. The arguments NA1 and NA2 are the number of differential areas for each surface, when numerical integration is used to determine VIEW factors. NA1 is only used for ITYPE equal to 3, 4, or 5. NA2 is only for ITYPE equal to 5.

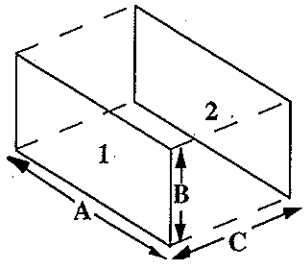

Figure 3.4.6.1 ITYPE=1; parallel identical rectangles.

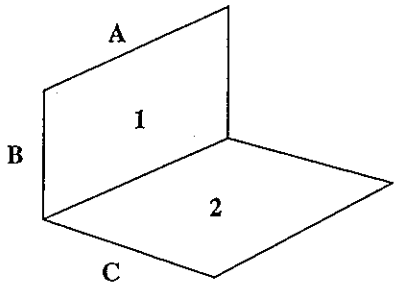

Figure 3.4.6.2 ITYPE=2; perpendicular rectangles with a common edge.

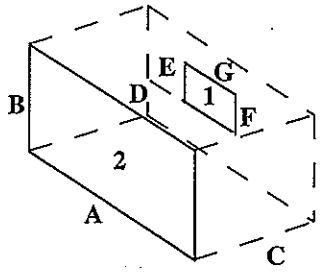

Figure 3.4.6.3 ITYPE=3; nonidentical parallel rectangles.

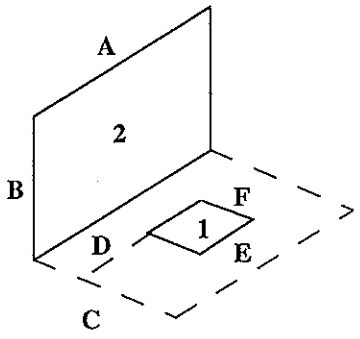

Figure 3.4.6.4 ITYPE=4; perpendicular rectangles without a common edge.

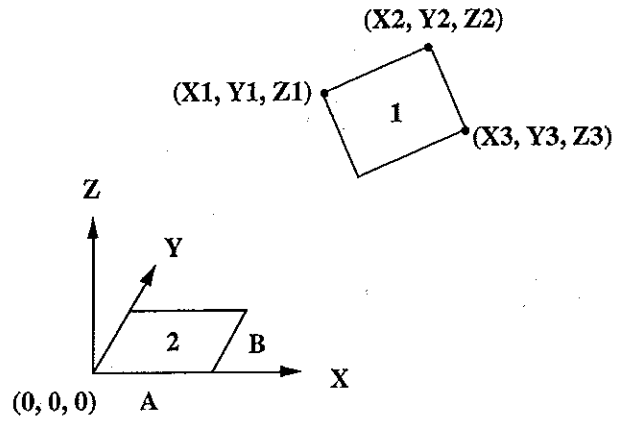

Figure 3.4.6.5 ITYPE=5; two rectangular surfaces of any orientation (X1=C, Y1=D, Z1=E, X2=F, Y2=G, Z2=H from subroutine call statement).

Mathematical Description:

ITYPE = 1

The view factor from surface two to one of Figure 3.4.6.1 is determined with the following expression taken from Reference 1.

$$F_{21} = \frac{2}{\pi XY}\left\{ \ln\left[\frac{(1+X^2)(1+Y^2)}{1+X^2+Y^2}\right]^{1/2} + \sqrt{1+Y^2}\tan^{-1}\frac{X}{\sqrt{1+Y^2}} + Y\sqrt{1+X^2}\tan^{-1}\frac{Y}{\sqrt{1+X^2}} - X\tan^{-1}X - Y\tan^{-1}Y \right\} \quad (3.4.6.1)$$

where,

X = A/C

Y = B/C

ITYPE = 2

From Reference 1, the view factor from surface two to one of Figure 3.4.6.2 is

$$F_{21} = \frac{1}{\pi Y}\left(Y\tan^{-1}\frac{1}{Y} + X\tan^{-1}\frac{1}{X} - \sqrt{X^2+Y^2}\tan^{-1}\frac{1}{\sqrt{X^2+Y^2}} + \frac{1}{4}\ln\left\{\frac{(1+Y^2)(1+X^2)}{1+X^2+Y^2}\left[\frac{Y^2(1+X^2+Y^2)}{(1+Y^2)(X^2+Y^2)}\right]^{Y^2}\left[\frac{X^2(1+X^2+Y^2)}{(1+X^2)(X^2+Y^2)}\right]^{X^2}\right\}\right) \quad (3.4.6.2)$$

where,

X = B/A

Y = C/A

ITYPE = 3

The VIEW factor from surface two to one of Figure 3.4.6.3 is found by numerical integration using the following expression:

$$F_{21} = \frac{\int_{A_1} F_{d1\text{-}2}\, dA_1}{A_2} \qquad (3.4.6.3)$$

where $F_{d1\text{-}2}$ is the view factor from an elemental area dA_1 on surface 1 to surface 2 and A_1 and A_2 are the areas of surfaces 1 and 2, respectively. The number of differential areas used in the integration, NA1, is specified as an argument in the call to VIEW.

The view factor $F_{d1\text{-}2}$ is determined by breaking surface 2 into 4 sections such that the normal to the center of the element passes through a common corner of each section of surface 2.

From Reference 1,

$$F_{d1\text{-}2} = \sum_{i=1}^{4} \frac{1}{2\pi}\left(\frac{X_i}{\sqrt{1+X_i^2}}\tan^{-1}\frac{Y_i}{\sqrt{1+X_i^2}} + \frac{Y_i}{\sqrt{1+Y_i^2}}\tan^{-1}\frac{X_i}{\sqrt{1+Y_i^2}}\right) \qquad (3.4.6.4)$$

where

$$X_i = \frac{A_i}{C}$$

$$Y_i = \frac{B_i}{C}$$

and A_i and B_i are dimensions of each section on surface 2, measured in the same directions as A and B.

ITYPE = 4

The view factor from surface 2 to 1 of Figure 3.4.6.4 is also determined using Equation 3.4.6.3 The view factor $F_{d1\text{-}2}$ is found by breaking surface 4 into 2 sections such that the normal to a common corner of the sections splits the element in half. From Reference 1;

$$F_{d1\text{-}2} = \sum_{i=1}^{2} \frac{1}{2\pi}\left(\tan^{-1}\frac{1}{Y_i} - \frac{Y_i}{\sqrt{X_i^2+Y_i^2}}\tan^{-1}\frac{1}{\sqrt{X_i^2+Y_i^2}}\right) \qquad (3.4.6.5)$$

where

$$X_i = \frac{A}{B_i}$$

$$Y_i = \frac{C'}{B_i}$$

B_i is the dimension of each section measured in the direction of B. C' is the distance from the edge of surface 2 to the center of the element on surface one.

ITYPE = 5

For two rectangular surfaces of arbitrary orientation as shown in Figure 3.4.6.5, the view factor from 2 to 1 is:

$$F_{21} = \frac{1}{A_2}\int_{A_1}\int_{A_2} \frac{\cos\theta_1 \cos\theta_2}{\pi S^2} dA_2 dA_1 \qquad (3.4.6.6)$$

where S is the distance between the centers of elemental areas on each surface, θ_1 and θ_2 are angles between the normals to each element and the line connecting their centers, and A_1 and A_2 are the areas of surfaces 1 and 2. Equation (3.4.6.6) is solved by numerical integration. The number of elemental areas used in the integration procedure, NA1 and NA2, are specified in the argument list in the call to VIEW.

The distance S is

$$S = \sqrt{(X_{d2} - X_{d1})^2 + (Y_{d1} - Y_{d2})^2 + (Z_{d2} - Z_{d1})^2} \qquad (3.4.6.7)$$

where (X_{d1}, Y_{d1}, Z_{d1}) and (X_{d2}, Y_{d2}, Z_{d2}) are the coordinates of the elemental areas on surfaces 1 and 2, respectively.

For Surface 2,

$$\cos\theta_2 = \frac{Z_{d1} - Z_{d2}}{S} \qquad (3.4.6.8)$$

For Surface 1,

$$\cos\theta_1 = \frac{\vec{S} \cdot \vec{N}}{|N|\,|S|} \qquad (3.4.6.9)$$

where $\vec{S}$ is the vector originating at the center of the elemental area on surface 1 and ending at the center of the second element on surface 2 and $\vec{N}$ is a normal vector to the elemental area on surface 1.

In general,

$$\vec{S} = S_X \vec{i} + S_Y \vec{j} + S_Z \vec{k} \qquad (3.4.6.10)$$

$$\vec{N} = n_X \vec{i} + n_Y \vec{j} + n_Z \vec{k} \qquad (3.4.6.11)$$

where $\vec{i}$, $\vec{j}$, and $\vec{k}$ are unit vectors in the x, y, and z directions, respectively. Substitution of Equations (3.4.6.10) and (3.4.6.11) into Equation (3.4.6.9) gives

$$\cos\theta = \frac{S_x n_x + S_y n_y + S_z n_z}{\sqrt{S_x^2 + S_y^2 + S_z^2}\ \sqrt{n_x^2 + n_y^2 + n_z^2}} \qquad (3.4.6.12)$$

where,

$$Sx = Xd2 - Xd1 \qquad (3.4.6.13)$$

$$Sy = Yd2 - Yd1 \qquad (3.4.6.14)$$

$$Sz = Zd2 - Zd1 \tag{3.4.6.15}$$

If n_z is arbitrarily chosen to be 1, then

$$n_x = \frac{K - n_y\, Y1 - Z1}{X1} \tag{3.4.6.16}$$

$$n_y = \frac{K\left(1 - \frac{X2}{X1}\right) + \frac{X2}{X1} Z1 - Z2}{Y2 - \frac{X2}{X1} Y1} \tag{3.4.6.17}$$

where,

$$K = \frac{\left(\frac{X3}{X1} Z1 - Z3\right)\left(Y2 - \frac{X2}{X1} Y1\right) - \left(\frac{X2}{X1} Z1 - Z2\right)\left(Y3 - \frac{X3}{X1} Y1\right)}{\left(1 - \frac{X2}{X1}\right)\left(Y3 - \frac{X3}{X1} Y1\right) - \left(1 - \frac{X3}{X1}\right)\left(Y2 - \frac{X2}{X1} Y1\right)}$$

The coordinates X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3 are as defined in Figure 3.4.6.5.

3.4.7 SUBROUTINE TABLE

Subroutine TABLE returns transfer function coefficients corresponding to the standard walls, partitions, floors, and ceilings given in the ASHRAE Handbook of Fundamentals. It is currently used by the TYPE 19 Zone Model.

A call to TABLE is of the form

CALL TABLE (ITYPE,NTABLE,IU,NBS,NCS,NDS,B,C,D,*N)

where

ITYPE - a variable equal to 1, 2, or 3 denoting which type of surface is desired. ITYPE = 1 corresponds to roof coefficients, ITYPE = 2 is for exterior walls, and ITYPE = 3 denotes interior partitions, floors and ceilings. The tables containing the surface descriptions can be found in the documentation for TYPE 19; both in this manual and in the Component Description manual.

NTABLE - number of wall, roof, etc., from the table specified by ITYPE.

NBS - the number of b coefficients for the desired wall as returned by the subroutine.

NCS - the number of c coefficients for the desired wall as returned by the subroutine.

NDS - the number of d coefficients for the desired wall as returned by the subroutine.

B - an array dimensioned to 10 that contains the b coefficients. These are the coefficients for the current and previous sol-air temperatures (i.e. b_o to b_{NBS-1}).

C - an array dimensioned to 10 that contains the c coefficients. These are the coefficients for the current and previous room temperatures (i.e. c_o to c_{NCS-1}).

D - an array dimensioned to 10 that contains the d coefficients. These are the coefficients for the previous heat fluxes (d_1 to d_{NDS}).

N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

Subroutine TABLE reads the transfer function coefficient from the file ASHRAE.COF

*Transfer function coefficients in the file ASHRAE.COF are different from those actually found in

accessed through FORTRAN logical unit 8. This logical unit can be changed by modifying a data statement within the subroutine. After the coefficients have been read for a particular wall, the coefficients are stored internally within the subroutine for use on subsequent calls to TABLE.

3.4.8 SUBROUTINES INVERT AND DINVRT

Subroutine INVERT is available to invert a matrix of up to 50 by 50 entries. INVERT is a single precision version and DINVRT is double precision.

A call to INVERT is of the form

CALL INVERT (NRC, N, A, IFLAG,*N)

where

- NRC - is the column and row dimensions of the two-dimensional A array as defined in the program that calls INVERT.
- N - the number of equations and unknowns associated with the problem.
- A - a two-dimensional array (A(NRC, NRC)). Upon calling INVERT A should contain the matrix to be inverted. INVERT also returns the inverted matrix in the A array.
- IFLAG - a flag which is set to 0 if the inversion proceeds correctly, 1 if the number equations and unknowns, N, exceeds 50, and 2 if the matrix is singular.
- N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

The inverted matrix is returned in the A array. The method used to invert the matrix is the Gauss-Jordan reduction with maximum pivoting (1).

3.4.9 SUBROUTINES FIT AND DFIT

General Description

Subroutines FIT and DFIT use linear least-squares regression (1) to determine coefficients of a user defined correlation. FIT uses single precision, while DFIT uses double precision.

A call to FIT is of the form:

CALL FIT(NROW,NCOL,NCOEF,NDATA,X,Y,PHI,IFLAG,*N)

where,

- NROW - Row dimension of X in routine calling FIT.
- NCOL - Column dimension of X in routine calling FIT.

---

the ASHRAE handbooks. Because the TYPE 19 single-zone model handles the inside radiative heat transfer seperately, the inside radiative resistance is not included in the ASHRAE.COF transfer coefficients. Thus while the file ASHRAE.COF is well-suited for use with TYPE 19 and other models which calculate radiative heat transfer separately, it must not be assumed that this file matches the ASHRAE handbook values. See Section 5.3 for more information.

NCOEF - Number of coefficients in model.
NDATA - Number of data points for the regression.
X - Two dimensional array containing the values of the independent variables to the model. X(I,J) is the value of the independent variable associated with the ith coefficient of the model and the j^{th} data point.
Y - Vector containing values of the dependent variable corresponding to independent variable data points.
PHI - Vector containing the coefficients determined from the regression.
IFLAG - Integer flag returned indicating the error condition (0 - no error, 1 - dimension error, 2 - coefficients cannot be determined).
N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

Least-squares regression is used to minimize a function J, where:

$$J = \sum_{j=1}^{N_{data}} (Y_j - Y_{model,j})^2 \qquad (3.4.9.1)$$

$$Y_{model,j} = \sum_{i=1}^{N_{coef}} \phi_i X_{i,j} \qquad (3.4.9.2)$$

and,

Y_j - jth data point for dependent variable
$Y_{model,j}$ - dependent variable of the user's model for the jth data point
ϕ_i - ith coefficient to be determined with FIT
$X_{i,j}$ - user defined functions of the independent variable or variables.

EXAMPLE

As an example, consider the equation used in the TYPE 53 Chiller component. The dimensionless power consumption is correlated with performance data using the following bi-quadratic equation.

$$G = a_0 + a_1 E + a_2 E^2 + a_3 F + a_4 F^2 + a_5 EF$$

This is the same form as equation (3.4.9.2) with:

ϕ_1-ϕ_5 = a_0-a_5
$X_{1,j} = 1.0$
$X_{2,j} = E$
$X_{3,j} = E^2$
$X_{4,j} = F$
$X_{5,j} = F^2$
$X_{6,j} = EF$

If the following data existed for E, F, and G:

E	F	G
3.0	0.2	10.0
5.0	0.4	12.5
7.0	0.6	13.1

Then FIT would be called with

NROW = 6 or more (minimum is the # of coefficients,i's)
NCOL = 3 or more (minimum is the # of data points,j's)
NCOEF = 6 (# of coefficients, i's)
NDATA = 3 (# of data points, j's)

$$X = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ E_1 & E_2 & E_3 \\ E_1^2 & E_2^2 & E_3^2 \\ F_1 & F_2 & F_3 \\ F_1^2 & F_2^2 & F_3^2 \\ EF_1 & EF_2 & EF_3 \end{bmatrix} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \\ 3.0 & 5.0 & 7.0 \\ 9.0 & 25.0 & 49.0 \\ 0.2 & 0.4 & 0.6 \\ .04 & .16 & .36 \\ 0.6 & 2.0 & 4.2 \end{bmatrix}$$

$$Y = \begin{bmatrix} 10.0 \\ 12.5 \\ 13.1 \end{bmatrix}$$

The FIT subroutine would be called with the above input data and return values for PHI and IFLAG. PHI would be a vector containing the values for a_0-a_5.

3.4.10 SUBROUTINE PSYCH

Subroutine PSYCH is used to calculate various moist air properties based on ASHRAE correlations and ideal gas laws.

A call to PSYCH is of the form

CALL PSYCH(TIME,INFO,IUNITS,MODE,WBMODE,PSYDAT,EMODE,STATUS,*N)

where

TIME - This is the standard TIME used in TRNSYS. It is used in the PSYCH subroutine only to identify the time at which warnings or errors occur.

INFO - This is the standard INFO array used in TRNSYS (see Section 3.3.3). It is used in the PSYCH subroutine to identify the unit and type number of the component which calls the subroutine, which will be printed during a simulation if warnings or errors occur.

IUNITS - an integer variable equal to 1 or 2 identifying the units desired for the properties. IUNITS = 1 results in properties in SI units. IUNITS = 2 results in properties in English units.

MODE - an integer variable from 1 through 6. Three properties are needed to specify the moist air state. The MODE identifies which two properties besides pressure are need to be input, leaving the other properties to be calculated. The modes are as follows:

1: input dry bulb and wet bulb temperatures (PSYDAT(2) and PSYDAT(3))
2: input dry bulb temperature and relative humidity (PSYDAT(2) and PSYDAT(4))
3: input dry bulb and dew point temperatures (PSYDAT(2) and PSYDAT(5))
4: input dry bulb temperature and humidity ratio (PSYDAT(2) and PYSDAT(6))
5: input dry bulb temperature and enthalpy (PSYDAT(2) and PSYDAT(7))
6: input humidity ratio and enthalpy (PSYDAT(6) and PSYDAT(7))

WBMODE - an integer variable equal to 0 or 1. If WBMODE = 0, the wet bulb temperature will not be calculated in MODEs 2 through 6. If WBMODE = 1, the wet bulb temperature is calculated. At PATM = 1.0 and temperatures common to HVAC applications, a correlation (1) is used to find the wet bulb temperature given enthalpy. For other pressures and temperatures, a Newton's iterative method is used with the equations given by ASHRAE (2) to calculate the wet bulb temperature. If the wet bulb temperature is not needed, WBMODE should be set to 0 to decrease the computational effort. If WBMODE = 0, the wet bulb temperature is set to the dry bulb temperature.

PSYDAT - an array containing the properties of the moist air. This array must be dimensioned to 9 in the routine that calls PSYCH. Each location in this array is described below:

PSYDAT(1) - The total system pressure in atmospheres. The subroutine prints a warning if PATM is over 5 atmospheres, which is considered here to be an upper limit for using ideal gas relations for air and water mixtures.

PSYDAT(2) - dry bulb temperature (°C).

PSYDAT(3) - wet bulb temperature (°C).

PSYDAT(4) - relative humidity (fraction).

PSYDAT(5) - dew point temperature (°C).

PSYDAT(6) - humidity ratio (kg water/kg dry air).

PSYDAT(7) - enthalpy (kJ/kg dry air).

PSYDAT(8) - density of the air water mixture (kg/m^3).

PSYDAT(9) - density of the air portion of the mixture (kg dry air/m^3).

EMODE - an integer variable equal to 0, 1, or 2 specifying how warnings are handled. If EMODE=0, no warnings will be printed. If EMODE=1, only one warning for each condition will be printed throughout the simulation. If EMODE=2, warnings will be printed every timestep. (EMODE=2 is useful for program and simulation development, but it can produce an excessive amount of output).

STATUS - an integer variable equal to 0 through 13 identifying the warning condition that occurred within subroutine PSYCH. 0 indicates no warnings. For a complete listing of warnings 1 through 13 the subroutine source code should be reviewed.

N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

Mathematical Description

In determining a number of moist air properties the water vapor saturation pressure (pws) is required. The correlations (2) used for pws are accurate over the temperature range of -100°C to 200 °C. A warning is printed if moist air states occur outside this temperature range.

The correlations for the dew point temperature (2) are accurate over the temperature range of -60 degrees C to 70 degrees C. A warning is printed if moist air states occur outside this dew point range.

Enthalpies are based on the following reference states.
air enthalpy is zero at 0° C
liquid water enthalpy is zero at 0° C

This subroutine checks for many improper inputs, such as relative humidities less than 0. or greater than 1., dew point temperatures greater than the dry bulb temperature, and the input of two properties that cannot be correct for one state (such as a humidity ratio greater than saturation humidity ratio for dry air at a given dry bulb temperature). For these and other improper inputs, the subroutine either prints a warning, corrects one of the inputs, and continues; or prints an error and halts the simulation.

3.4.11 SUBROUTINE LINKCK

The LINKCK subroutine is a utility subroutine used for the detection and subsequent error message printing of unlinked subroutines.

A call to LINKCK is of the form:

CALL LINKCK(ENAME1,ENAME2,ILINK,LNKTYP,*N)

where

ENAME1 - a 6-character variable identifying the subroutine from which the call originated.

ENAME2 - a 6-character variable identifying the subroutine that was called

ILINK - an integer indicating the steps to be taken by the LINKCK program
- =1 an unlinked subroutine has been found, generate an error message and stop the program
- =2 an unlinked subroutine has been found, generate a warning but keep running
- =3 an unlinked TYPE subroutine has been found, generate an error message and stop the program
- =4 warn the user that a subroutine requires use of an external function which can not be link checked

LNKTYP - integer variable corresponding to the TYPE subroutine which was not found in the TRNSYS executable

The LINKCK subroutine is provided to the users as a means to standardize the TRNSYS error and warning messages associated with unlinked subroutines. The user should only call LINKCK when an unlinked subroutine is detected or an external function is required.

The following example should clarify the use of the LINKCK subroutine. Refer to section 3.3.5 for more details.

```
      SUBROUTINE TYPE75(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*)

      CHARACTER*6  ENAME1,ENAME2

*     CALL THE ENCL SUBROUTINE AND INFORM LINKCK IF NOT PRESENT
      CALL ENCL (SPAR, WPAR, FV, INFO,*101)
*           STOP THE PROGRAM IF ENCL IS NOT PRESENT
      ILINK=1
      IDUM=75
      ENAME1='TYPE75'
      ENAME2='ENCL '
      CALL LINKCK(ENAME1,ENAME2,ILINK,IDUM)

*     ENCL IS PRESENT - CONTINUE ON
101   CONTINUE


      RETURN 1
```

3.4.12 SUBROUTINE RCHECK

The RCHECK subroutine was added to TRNSYS 14 to provide input-output mismatch checking for all components.

A call to RCHECK is of the form:

CALL RCHECK(INFO,YCHECK,OCHECK)

where

INFO - This is the standard INFO array used in TRNSYS (see Section 3.3.3). It is used in the PSYCH subroutine to identify the unit and type number of the component which calls the subroutine, which will be printed during a simulation if warnings or errors occur.

YCHECK - A 3 character array containing the expected input types for the component.

OCHECK - A 3-character array containing the output types for the component

The RCHECK array is passed the input and output types for each component in the simulation and stores these variables in arrays with the same structure as the XIN and OUT arrays. When the TRNSYS processor checks for input-output mismatches, the arrays are inspected to ensure that the variable types for an input-output connection are consistent.

A user formulating a new component should utilize the input/output checking feature of TRNSYS for two reasons: input/output checking ensures that TRNSYS connections are correctly defined, and output information such as output type and output units are carried with the output and are able to be processed by the TYPE 25 printer and the TYPE 57 unit conversion routine.

The first step in input/output checking is to characterize the inputs and outputs using Table 3.4.12.1. Users wishing to create additional input-output types or add a unit conversion to an existing input-output type should modify the file "UNITS.LAB", keeping the same format and style as the original. Refer to the TYPE 57 unit conversion routine for more details on modifying this file. The input-output types should be stored in the YCHECK and OCHECK arrays respectively.

The final step in using the input/output checking is to call the RCHECK array at the proper time, with the proper arrays. The RCHECK routine should be called during the first iteration in most cases (INFO(7)=-1 see Section 3.3.3) and after the call to the TYPECK subroutine (see Section 3.4.1).

The following example should help clarify the use of the RCHECK subroutine:

```
      SUBROUTINE TYPE75(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*)

*     3 INPUTS AND 4 OUTPUTS FOR THIS TYPE
*     INPUTS: TEMPERATURE IN  (F)
*             MASS FLOWRATE IN  (LBM/HR)
*             CONTROL SIGNAL  (0 OR 1)
*     OUTPUTS: TEMPERATURE OUT  (C)
*              MASS FLOWRATE OUT (KG/HR)
               HEAT TRANSFER RATE (KJ/HR)
               HEAT TRANSFER RATE (W)
      CHARACTER*3 YCHECK(3),OCHECK(4)
      DATA YCHECK/'TE2','MF3','CF1'/
      DATA OCHECK/'TE1','MF1','PW1','PW2'/
        .
        .
      IF (INFO(7).EQ.-1) THEN
          .
          CALL TYPECK(1,INFO,NI,NP,ND)
          CALL RCHECK(INFO,YCHECK,OCHECK)
          .
      ENDIF
```

TABLE #1: TEMPERATURE

VAR. TYPE #	VAR. UNITS	VAR.TYPE	MULT. FACTOR	ADD. FACTOR
1	°C	TE1	1.0	0
2	°F	TE2	1.8	32
3	K	TE3	1.0	273.15
4	R	TE4	1.8	492

TABLE #2: LENGTH

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m	LE1	1	0
2	cm	LE2	100	0
3	km	LE3	1000	0
4	in	LE4	39.3701	0
5	ft	LE5	3.28084	0
6	miles	LE6	6.21371 E-04	0

TABLE #3: AREA

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m2	AR1	1	0
2	cm2	AR2	1 E+04	0
3	km2	AR3	1 E-06	0
4	in2	AR4	1550	0
5	ft2	AR5	10.7639	0
6	mi2	AR6	3.86102 E-07	0

TABLE #4: VOLUME

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m3	VL1	1	0
2	l	VL2	1000	0
3	ml	VL3	1 E+06	0
4	in3	VL4	6.10237 E+04	0
5	ft3	VL5	35.3147	0
6	gal	VL6	264.172	0

TABLE #5: SPECIFIC VOLUME

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m3/kg	SV1	1	0
2	l/kg	SV2	1000	0
3	ft3/lbm	SV3	16.0185	0
4	in3/lbm	SV4	2.76799 E+04	0

TABLE #6: VELOCITY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m/s	VE1	1.0	0
2	km/hr	VE2	3.6	0
3	ft/s	VE3	3.28084	0
4	ft/min	VE4	196.85	0
5	mph	VE5	2.23694	0

TABLE #7: MASS

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kg	MA1	1	0
2	g	MA2	1000	0
3	lbm	MA3	2.20462	0
4	ounces	MA4	35.274	0
5	ton	MA5	1.10231 E-03	0

TABLE #8: DENSITY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kg/m3	DN1	1.0	0
2	kg/l	DN2	0.001	0
3	lbm/ft3	DN3	6.2428 E-02	0
4	lbm/gal	DN4	8.3454 E-03	0

TABLE #9: FORCE

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	N	FR1	1.0	0
2	lbf	FR2	0.224809	0
3	ounce	FR3	3.59694	0

TABLE #10: PRESSURE

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	BAR	PR1	1	0
2	kPa	PR2	100	0
3	Pa	PR3	1 E+05	0
4	ATM	PR4	0.986923	0
5	psi	PR5	14.5038	0
6	lbf/ft2	PR6	2.08854 E+03	0
7	in. H2O	PR7	401.463	0
8	in. Hg	PR8	29.53	0

TABLE #11: ENERGY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ	EN1	1	0
2	kWh	EN2	2.77778 E-04	0
3	Cal	EN3	238.846	0
4	ft-lbf	EN4	737.562	0
5	hp-hr	EN5	3.72506 E-04	0
6	BTU	EN6	0.947817	0

TABLE #12: POWER

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/hr	PW1	1	0
2	W	PW2	0.277778	0
3	kW	PW3	2.77778 E-04	0
4	hp	PW4	3.72505 E-04	0
5	BTU/hr	PW5	0.947817	0
6	BTU/min	PW6	1.57969 E-02	0
7	Tons	PW7	7.89847 E-05	0

TABLE #13: SPECIFIC ENERGY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/kg	SE1	1	0
2	BTU/lbm	SE2	0.429923	0
3	ft-lbf/lbm	SE3	334.553	0

TABLE #14: SPECIFIC HEAT

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/kg-K	CP1	1	0
2	W-hr/kg-K	CP2	0.277778	0
3	BTU/lbm-R	CP3	0.238846	0

TABLE #15: FLOW RATE

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kg/hr	MF1	1	0
2	kg/s	MF2	2.77778 E-04	0
3	lbm/hr	MF3	2.20462	0
4	lbm/s	MF4	6.12395 E-04	0

TABLE #16: VOLUMETRIC FLOW RATE

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m3/hr	VF1	1	0
2	m3/s	VF2	2.77778 E-04	0
3	l/hr	VF3	1000	0
4	l/s	VF4	0.277778	0
5	ft3/s	VF5	9.80958 E-03	0
6	ft3/hr	VF6	35.3144	0
7	gpm	VF7	4.40286	0

TABLE #17: FLUX

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/hr-m2	IR1	1	0
2	W/m2	IR2	0.277778	0
3	BTU/hr-ft2	IR3	8.8055 E-02	0

TABLE #18: THERMAL CONDUCTIVITY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/hr-m-K	KT1	1	0
2	W/m-K	KT2	0.277778	0
3	BTU/hr-ft-R	KT3	0.160497	0

TABLE #19: HEAT TRANSFER COEFFICIENTS

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	kJ/hr-m2-K	HT1	1	0
2	W/m2-K	HT2	0.277778	0
3	BTU/hr-ft2-R	HT3	4.89194 E-02	0

TABLE #20: DYNAMIC VISCOSITY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	N-s/m2	VS1	1	0
2	kg/m-s	VS2	1	0
3	poise	VS3	10	0
4	lbf-s/ft2	VS4	2.08854 E-02	0
5	lbf-hr/ft2	VS5	5.80151 E-06	0
6	lbm/ft-hr	VS6	2419.08	0

TABLE #21: KINEMATIC VISCOSITY

VAR. TYPE #	VAR. UNITS	VAR. TYPE	MULT. FACTOR	ADD. FACTOR
1	m2/s	KV1	1	0
2	m2/hr	KV2	3600	0
3	ft2/s	KV3	10.7639	0
4	ft2/hr	KV4	3.87501 E+04	0

MISCELLANEOUS

	VAR. TYPE
Dimensionless	DM1
Degrees	DG1
Percentage	PC1
Month	MN1
Day	DY1
Hour	TD1
Control function	CF1

Table 3.4.12.1

In user-written routines that have output variable types dependent on input variable types, such as a statistics or integrator routine, the call to RCHECK must come on the second iteration of the first timestep, after a call to the EQUATER subroutine. The EQUATER subroutine will determine the output connected to the unknown input and provide the appropriate input type.

A call to the EQUATER subroutine has the following form:

```
      CALL EQUATER(IU,INNO,INDES,TRUTYP,OUTU,OUTNO,ODES,X1,A1,X2,A2,
     1                    ICK1,ICK2)
```

where

IU - the unit number of the user-written component routine calling the EQUATER subroutine (integer variable)

INNO - the input number with the undetermined input type (integer variable)

INDES - the desired input variable type - not used in this capacity, set to 'NAV' for not available (3-character variable)

TRUTYP - a 3 character array returning the output variable type connected to this input (this is the desired result for this application) (3-character variable)

OUTU - the unit number of the output connected to the unknown input (not used in this capacity) (integer variable)

OUTNO - the output number of the OUTU unit connected to the unknown input (not used in this capacity) (integer variable)

ODES - the desired output variable type for this unknown input - not used in this capacity (3-character variable)

X1 - the multiplication factor for the INDES variable - not used in this capacity (real value)

A1 - the addition factor for the INDES variable - not used in this capacity (real value)

X2 - the multiplication factor for the OUTDES variable - not used in this capacity (real value)

A2 - the addition factor for the OUTDES variable - not used in this capacity (real value)

ICK1 - error flag for calculation (1 = error found) (integer variable)

ICK2 - error flag for calculation (1 = error found) (integer variable)

Users attempting to use the input/output checking algorithm in this capacity should refer to the TYPE 55 Periodic Integrato r subroutine and the EQUATER subroutine source code for more details.

3.4.13 SUBROUTINE FLUIDS

Subroutine FLUIDS is used to calculate the thermodynamic properties of various refrigerants based on correlations. Given two state properties, the FLUIDS routine will return the complete state of the refrigerant.

A call to FLUIDS is of the form

CALL FLUIDS (UNITS,PROP,NREF,ITYPE,IFLAGR,*N)

where

UNITS - two character variable denoting unit system to be employed by the fluids routine; 'SI' for metric units and 'EN' for english units

PROP - real array holding the two known thermodynamic properties of the refrigerant in question upon entry and the complete state of the refrigerant upon return from the FLUIDS routine. This array must be dimensioned to 9 in the routine that calls FLUIDS. Each location in this array is described below:

PROP(1) - temperature of refrigerant (F, C)
PROP(2) - pressure of refrigerant (PSI, MPa)
PROP(3) - enthalpy of refrigerant (BTU/lbm, kJ/kg)
PROP(4) - entropy of refrigerant (BTU/lbm R, kJ/kg K)
PROP(5) - quality (0<x<1)
PROP(6) - specific volume (ft^3/lbm, m^3/kg)
PROP(7) - internal energy (BTU/lbm, kJ/kg)
PROP(8) - dynamic viscosity (lbm/ft hr, N s/m^2)
PROP(9) - thermal conductivity (BTU/hr ft R, kJ/hr m K)

NREF - an integer variable denoting the refrigerant to be analyzed. Available refrigerants are listed along with the required NREF code number:

R-11	(11)
R-12	(12)
R-13	(13)
R-14	(14)
R-22	(22)
R-114	(114)
R-134A	(134)
R-500	(500)
R-502	(502)
AMMONIA	(717)

ITYPE - integer variable denoting which two properties are supplied as known properties. ITYPE = 10* PROP indicator 1 + PROP indicator 2 (12 to 98)

IFLAGR - an integer variable returning the error messages from the FLUIDS call:
0 - no error found, calculations completed
1 - error found, calculations could not be completed

N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

An example illustrating the use of the FLUIDS routine is shown below with the following considerations:

SI units are desired. (UNITS='SI')
Refrigerant R12 (NREF=12)
Temperature (PROP(1)) and Quality (PROP(5)) are known. (ITYPE=10*1+5)

```
REAL PROP(9)
CHARACTER*2 UNITS
    .
    .
TEMP=XIN(1)
QUALITY=XIN(2)
    .
    .
PROP(1)=TEMP
PROP(5)=QUALITY
CALL FLUIDS(UNITS,PROP,12,15,IFLAG,*100)
    .
    .
PRESSURE=PROP(2)
ENTHALPY=PROP(3)
```

Mathematical Description

The correlations required to solve for the refrigerant properties are from the EES program (8), a numerical solver employing thermodynamic correlations from many different sources. Interested users should contact the reference for more details on the correlations.

Enthalpies for ammonia and R134a are based on the following reference states.
enthalpy equal to zero at -40° C

Enthalpies for all other refrigerants are based on the following reference states.
enthalpy equal to zero at 0° C

This subroutine checks for many improper inputs, such as qualities less than 0. or greater than 1., and the input of two properties that cannot be correct for one state. For these and other improper inputs, the subroutine either prints a warning, corrects one of the inputs, and continues; or prints an error and halts the simulation.

Subcooled properties are not available for refrigerants. If a subcooled state is specified, the saturated liquid results will instead be provided.

3.4.14 SUBROUTINE STEAM

Subroutine STEAM is used to calculate the thermodynamic properties of steam based on correlations from the National Bureau of Standards. Given two state properties, the steam routine will return the complete state of the fluid.

A call to STEAM is of the form

CALL STEAM (UNITS,PROP,ITYPE,IERR,*N)

where

UNITS - two character variable denoting unit system to be employed by the fluids routine; 'SI' for metric units and 'EN' for english units

PROP - real array holding the two known thermodynamic properties of steam upon entry and the complete state upon return from the STEAM routine. This array must be dimensioned to 7 in the routine that calls STEAM. Each location in this array is described below:

PROP(1) - temperature of refrigerant (F, C)
PROP(2) - pressure of refrigerant (PSI, MPa)
PROP(3) - enthalpy of refrigerant (BTU/lbm, kJ/kg)
PROP(4) - entropy of refrigerant (BTU/lbm R, kJ/kg K)
PROP(5) - quality (0<x<1)
PROP(6) - specific volume (ft^3/lbm, m^3/kg)
PROP(7) - internal energy (BTU/lbm, kJ/kg)

ITYPE - integer variable denoting which two properties are supplied as known properties. ITYPE = 10* PROP indicator 1 + PROP indicator 2 (12 to 76)

IERR - an integer variable returning the error messages from the FLUIDS call:
0 - no error found, calculations completed
> 0 - error found, calculations could not be completed

N - the corresponding line number to jump to if the subprogram is present (see Section 3.3.5)

An example illustrating the use of the STEAM routine is shown with the following considerations:

SI units are desired. (UNITS='SI')

Temperature (PROP(1)) and Quality (PROP(5)) are known. (ITYPE=10*1+5)

```
REAL PROP(7)
CHARACTER*2  UNITS
      .
      .
TEMP=XIN(1)
QUALITY=XIN(2)
      .
      .
PROP(1)=TEMP
PROP(5)=QUALITY
CALL FLUIDS(UNITS,PROP,ITYPE,IERR,*100)

PRESSURE=PROP(2)
ENTHALPY=PROP(3)
```

Mathematical Description

The correlations required to solve for the steam properties are from the EES program (8), a numerical solver employing thermodynamic correlations from many different sources. Interested users should contact the reference for more details on the correlations.

Enthalpies for steam are based on the following reference state.

enthalpy equal to zero at 0° C

This subroutine checks for many improper inputs, such as qualities less than 0. or greater than 1., and the input of two properties that cannot be correct for one state. For these and other improper inputs, the subroutine either prints a warning, corrects one of the inputs, and continues; or prints an error and halts the simulation.

Subcooled properties are <u>not</u> available for steam. If a subcooled state is specified, the saturated liquid results will instead be provided.

3.5 STORAGE FOR COMPONENT ROUTINES

Since a single TYPE subroutine may be used for several UNITS in one simulation, care must be taken when storing values from one timestep to the next. If a tank model, for instance, requires the initial tank temperature throughout the simulation, simply setting a variable TZERO to the tank temperature when INF0(7) = -1 will not work. When two tanks are used, the local variable TZERO will be set twice. Throughout the simulation, it will contain the initial temperature of the second tank. As a result, the first tank will not be modeled correctly.

One solution to this problem is to use the OUT array. Since each unit is allocated a minimum of 20 outputs, there is often enough space to store a few extra numbers which are not actual outputs. The number of outputs can be increased beyond 20 by using INFO(6). However,

for large storage requirements (greater than 10 or so), it is recommended that the S array in the STORE common block be used. (The COMMON statement for the STORE common block should be inserted into the subroutine, as shown below.)

Subroutine TYPECK will allocate storage space in the S array upon request. If a routine requires N storage locations, INFO(10) should be set to N before TYPECK is called with IOPT = 1. TYPECK will then reset INFO(10) to an index into the S array in the STORE common block. Throughout the rest of the simulation, locations S(INFO(10)), S(INFO(10) + 1), . . . , S(INFO(10) + N - 1) in common block STORE will be reserved for the component routine. Typically a suitable integer variable is given the value in INFO(10).

A tank model which uses the S array to store the initial tank temperature might contain the following lines:

```
.
.
INCLUDE '/TRNWIN/KERNAL/PARAM.INC'
.
.
COMMON/STORE/NSTORE, IAV, S(NUMSTR)
.
.
.
IF (INFO (7) .EQ. -1) THEN
      INFO (10) = 1
      CALL TYPECK (1, INFO, NI, NP, ND)
      I = INFO (lO)
      S (I) = T
      .
      .
      .
ENDIF
I = INFO(10)
TZERO = S(I)
.
.
.
```

COMMON block STORE makes the S array available to the component subroutine. NSTORE, IAV, and NUMSTR are used internally by TRNSYS and should not be changed in a TYPEn subroutine. NUMSTR is new in version 14.2 and is the total number of storage spaces in the S array. To change the number of storage spaces available for the S array (NUMSTR), see manual section 6.5. NUMSTR is currently set to 5000. NSTORE is also the dimension of the S array. IAV is a pointer to the next available location in the S array. INF0(7) is used in this example to identify the first call of the simulation to the UNIT, so that storage space in the S array is only allocated to the UNIT once.

3.6 SUMMARY OF COMPONENT SUBROUTINE FORMULATION

The concepts and rules for writing component subroutines may now be summarized.

(i) The component subroutine statement must have the form:
SUBROUTINE TYPEn (TIME, XIN, OUT, T, DTDT, PAR, INFO, ICNTRL, *)
where n is the TYPE number defining that component type (see Section 3.3). It may have a value of 1 through 23 or 30 through 99. (Type numbers 24 through 29 are already used by special "built-in" components described in Chapter 4.)

(ii) If a component model has PARAMETERS, INPUTS, OUTPUTS, DERIVATIVES (time-dependent variables), or uses the INFO array, an appropriate DIMENSION statement must appear after the subroutine name (See Section 3.3).

(iii) The INFO array may be used as described in Section 3.3.3. Possible applications are in identifying the first call of the simulation for purposes of error checking, input/output checking and allocation of UNIT-specific storage space in the S array, in avoiding unnecessary calls to routines whose OUTPUTS do not explicitly depend on the passage of time, and in providing stability for controller routines.

(iv) Values are assigned to the mnemonic model parameters from the PAR array, noting that the subroutine parameter list then designates the sequential order of the parameter for the component model. Parameter values must be assigned on each call to the subroutine, and not just on the first call of the simulation, since two or more UNITS of the same type but with different parameter values may share the same TYPE n subroutine.

(v) The mnemonic variables within the subroutine are set equal to the input variables in the XIN array, once again noting the sequential order.

(vi) The desired outputs of the subroutine must be placed in the OUT array before the RETURN 1 statement. As before, this action will set the outputs order. A minimum of 20 OUTPUTS are reserved for each UNIT (see description of INFO(6) in Section 3.3.3).

(vii) First order differential equations are integrated by calling subroutine DIFFEQ or by placing the DERIVATIVES in the DTDT array. The use of DIFFEQ is computationally more efficient and should be used when the new TRNSYS solver is employed, but may not be applicable to some situations (see Section 1.10). When the DTDT array is used, the values of the dependent variables T at any time appear in the T array in the same order as the DERIVATIVES. Systems of coupled differential equations can be solved using both DIFFEQ, to integrate the individual derivatives, and the DTDT array, to take advantage of the TRNSYS apparatus for testing individual solutions and forcing iteration until the solutions converge.

The user will find it convenient to draw an information flow diagram similar to the ones in Chapter 4, showing the sequentially ordered PARAMETERS, INPUTS, and OUTPUTS which characterize the component and specifying the need for a DERIVATIVES control command if the component makes use of the DTDT array.

3.7 EXAMPLE

The following example illustrates a number of special considerations in writing TRNSYS-compatible subroutines. This component takes from one to ten on/off control functions as input, and generates an exponential rise or fall for each output in response to a step change in the input, based on time constants supplied as parameters. The behavior of the component is shown in Figure 3.7-1, and is described by the following differential equations:

$$y = x - \tau_{on}\frac{dy}{dt} \qquad \text{if } x > y$$

$$y = x - \tau_{off}\frac{dy}{dt} \qquad \text{if } x < y$$

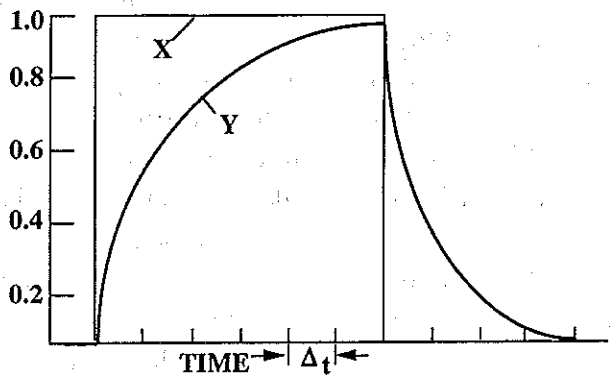

Figure 3.7.1

The information flow diagram for this component appears in Figure 3.7-2. For each control input, two outputs result. The first is the value of Y at the end of the timestep, while the second is the average value of Y over the interval.

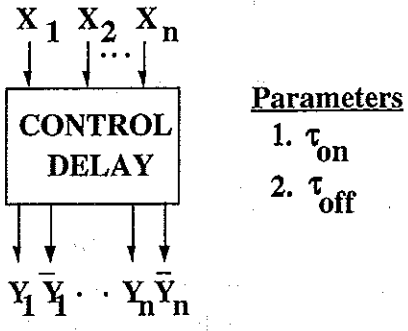

Figure 3.7.2

The source code for this model, specified as a TYPE 75, is:

```
      SUBROUTINE TYPE75(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL*)
      DOUBLE PRECISION XIN,OUT
      DIMENSION XIN(3),OUT(20),PAR(6),INFO(15)
      CHARACTER*3 YCHECK(3),OCHECK(6)
      COMMMON/STORE/NSTORE,IAV,S(5000)
      DATA YCHECK/'CF1','CF1','CF1'/
      DATA OCHECK/'CF1','CF1','CF1','CF1','CF1','CF1'/
      NI=INFO(3)
      IF(INFO(7)  .EQ.  -1) THEN
C
C         FIRST CALL OF SIMULATION
          NP=2*NI
          INFO(10)=NI
          CALL TYPECK(1,INFO,NI,NP,0)
          CALL RCHECK(INFO,YCHECK,OCHECK)
          DO 50 I=1,NI
50            OUT(2*I-1)=XIN(I)
          RETURN  1
      ENDIF
      INDEX=INFO(10)-1
      IF(INFO(7)  .EQ.  0) THEN
C
C         FIRST CALL IN CURRENT TIMESTEP
C         SAVE FINAL VALUES FROM PREVIOUS TIMESTEP

          DO 120 I=1,NI
120           S(INDEX+I)=OUT(2*I-1)
      ENDIF


      DO 220 I=1,NI
          X=XIN(I)
          Y=S(INDEX+I)
          TAU=PAR(2*I-1)
          IF(X .LT. Y) TAU=PAR(2*I)
          A=-1./TAU
          B=X/TAU
          CALL DIFFEQ(TIME,A,B,Y,YNEW,YBAR)
          OUT(2*I-1)=YNEW
          OUT(2*I)=YBAR
220   CONTINUE
      RETURN  1
      END
```

The analytical solution to the equation could have been included explicitly in the component model. This would require that the subroutine know the simulation timestep, which could have been made available by inserting the following FORTRAN statement:

```
COMMON/SIM/TSTART,TSTOP,DELT,IWARN
```

Instead, the differential equation is solved for each INPUT by SUBROUTINE DIFFEQ (see Section 3.4.4).

Since a TYPE subroutine may represent two or more UNITS in a given simulation, local variables such as NI, the number of inputs to the components, will not necessarily retain their values from one call to the next. NI is needed on every call to the subroutine, so it is placed where its value will be reset from the INFO array every time the subroutine is called.

Although not practical for this example, the need to reset local variables on every call to a subroutine could be avoided by including the following lines:

```
      IF(INFO(1).NE.IUNIT) THEN
         IUNIT = INFO(1)
C.       SET LOCAL VARIABLES
         .
         .
         .
      ENDIF
```

Local variables are set on the first call to the subroutine, and only reset if two or more UNITS of the subroutine appear in the simulation.

Lines 12-18 are only used the first time the subroutine is called with a given UNIT number (i.e., when INFO(7) = - 1). The call to SUBROUTINE TYPECK serves two purposes. First, it insures that the number of parameters specified in the simulation input file is consistent with the number of inputs, and that no DERIVATIVES statement is included. Second, it assigns to the UNIT NI storage spaces in the S array for saving UNIT-specific information from one timestep to the next. The initial values of the INPUTS are also placed in the OUT array at this time. The call to TYPECK is necessary for proper intialization of the component.

Lines 4, 6, 7 and 15 are required by the input/output checking algorithm new to TRNSYS 14. The algorithm will insure that the proper input types are connected to the newly developed type. The input/output algorithm also defines the output types from this component which may be hooked up to inputs of other components or to integrators and printers.

The TYPE subroutine may be called iteratively several times in a timestep, but stored values should only be updated once per timestep. At each iteration, results should be based on the final values from the previous timestep, rather than on the tentative values from the previous iteration. To accomplish this, final values from the previous timestep are transferred from the OUT array to the S array only on the first call of the current timestep (i.e., when INFO (7) = 0).

In this example, the NI differential equations are independent. Had the solutions been

interdependent, the NI values of YBAR could have been placed in the DTDT array to activate the TRNSYS apparatus for iteratively solving systems of equations. NI DERIVATIVES would then be required, and the DTDT array would have to be dimensioned.

This example is similar to the TYPE 24 Integrator in that it performs essentially the same operation on each of a variable number of INPUTS. In practice it might be more efficient to allow only one or two INPUTS, and to use the OUT array rather than the S array for storage.

3.8 CONVERTING FROM VERSION 13 to VERSION 14.1

TRNSYS routines written for version 13.1 or earlier have to be modified for use with TRNSYS 14.1 The steps are summarized below for convenience - refer to the sections listed for full details.

1) Change the TYPE specification from:
(TIME,XIN,OUT,T,DTDT,PAR,INFO)
to:
(TIME,XIN,OUT,T,DTDT,PAR,INFO,ICNTRL,*) Section 3.3

2) Replace all RETURN statements in the subroutine with RETURN 1 Section 3.3.5
Failure to replace all returns will result in an unlinked subroutine error.

3) Change control algorithms to accommodate new control strategy Section 3.3.4
(Optional)

4) Change the XIN and OUT arrays to DOUBLE PRECISION variables. Section 3.3

5) Change the INFO array dimension from 10 places to 15. Section 3.3.3

6) Define the input and output variable types in the YCHECK and Section 3.4.12
OCHECK arrays and call the RCHECK subroutine to perform input/output checking.

7) All calls to the following routines should be modified to include the Section 3.4.11
link checking option: DATA,ENCL,TABLE,INVERT,DINVRT,FIT,DFIT and PSYCH.

3.9 CONVERTING FROM VERSION 14.1 to VERSION 14.2

TRNSYS routines written for version 14.1 have to be modified for use with TRNSYS 14.2 for Windows The Windows operating system does not deal well with two common features of TRNSYS subroutines. The steps are summarized below for convenience.

1) Change any statements that contain a STOP statement from:

```
STOP
```

to the following if the Stop occurred in the primary Type subroutine:

```
CALL MYSTOP(1001)
RETURN 1
```

or to the following if the Stop occurred in a secondary subroutine:

```
CALL MYSTOP(1001)
RETURN
```

The number 1001 can be changed to another error number if so desired.

2) Windows does not allow any write-to-screen statements. Any WRITE or PRINT statements that print to the screen such as:

```
WRITE (*,*) ' There is an error in this type.'
```

need to be replaced with a statement that writes the text to the listing file.

```
WRITE(LUW,*) 'There is an error in this type'
```

Note: In most cases, writing to the list file requires the inclusion of the common block that contains the value for the logical unit. Include the following line:

```
COMMON /LUNITS/ LUR, LUW, IFORM, LUK
```

References

1. Whillier, A., "Solar Energy Collection and Its Utilization for House Heating," Ph.D. thesis in Mechanical Engineering, Massachusetts Institute of Technology, Appendix A2, (1953).
2. Duffie, J.A. and Beckman, W.A., Solar Engineering of Thermal Processes, Wiley-Interscience, New York (1980).
3. Robert Siegel and John R. Howell, Thermal Radiation Heat Transfer (New York: Hemisphere Publishing Corporation, 1981).
4. Carnahan, B., Luther, H.A, and Wildes, J.O., Applied Numerical Methods, Wiley, New York, 1969.
5. Robert Sedgewick, Algorithms, Addison-Wesley Publishing Company,Inc., 1983.
6. American Society of Heating, Refrigeration, and Air Conditioning Engineers, Procedure

for Determining Heating and Cooling Loads for Computerizing Energy Calculations, 1975, pp. 162-164.

7. American Society of Heating Refrigeration, and Air Conditioning Engineers, ASHRAE Handbook 1985 Fundamentals Volume (SI), Atlanta, GA, 1985.

8. *Engineering Equation Solver*, F-Chart Software, 4406 Fox Bluff Road, Middleton WI, 1994