5 ADVANCED DESIGN FEATURES

The previous sections of this manual concentrate on traditional point design evaluations using steady state and transient solutions. This section describes how to perform high-level operations to find inputs given outputs, optimize (minimize weight, maximize performance), correlate (calibrate a model to test data or reverse engineer a design), find the worst-case design scenario, estimate the reliability of a design, and optimize the design for reliability. These high-level operations use low-level point design operations (the realm of traditional steady state and transient analyses described in Sections 2-4) as building blocks. Therefore, the novice user should be familiar with the basic operation of the code before proceeding to this higher level of tasking.

Table 5-1 presents a summary of this section.

Table 5-1 Overview of Advanced Design Features

Routine	Module	Uses	Section
SOLVER	Solver	Optimization (sizing)	5.1-9, 5.11-12
		Goal seeking (reverse inputs/outputs)	5.1-8, 5.13
		Test data correlation	5.1-8, 5.10, 5.14
DVSWEEP	Solver	Parametric sweep of a single design variable	5.15
DSCANLH	Solver	Latin Hypercube scan of sweep of the design space	5.15
DSCANFF	Solver	Full factorial scan of sweep of the design space	5.15
SAMPLE	Reliability Engineering	Monte Carlo sampling for reliability estimation	5.16-19, 5.22-25
DSAMPLE	Reliability Engineering	Descriptive sampling for reliability estimation	5.16-5.18, 5.20, 5.22-25
RELEST	Reliability Engineering	First order Taylor series reliability estimation	5.16-5.18, 5.21, 5.22-25
RVSWEEP	Reliability Engineering	Parametric sweep of a single design variable	5.26
<nested>	Robust Design	Optimization using reliability as a constraint or objective	5.27

The use of registers is intimately connected with the use of the advanced design features described in this section. Registers (Section 2.8) provide a means by which the user can describe sweeping changes to a model on the basis of a few simple parameters such as key dimensions, performance metrics, or material selections. Using the dynamic register feature (Section 4.9), users can impose these changes while SINDA/FLUINT is executing as needed to investigate various design or model variations. The advanced modules takes this sequence a step further: the user lets SINDA/FLUINT itself change certain key parameters (registers) as needed to meet high-level design or analysis objectives, such as those described next.

SOLVER: The Solver is a high-level “solution routine” that allows the user to prepare a task to be performed by SINDA/FLUINT using steady state and transient analyses described in Sections 2, 3, and 4 of this manual. Some prior knowledge of nonlinear programming (formal optimization) techniques is also helpful, but not required.

The "task" given the Solver might be to find a design that minimizes or maximizes some quantity subject to arbitrarily complex constraints, or to find a model that best fits available test data. Or the task might be to find I given O, where I is normally an input value to SINDA/FLUINT and O is normally an output value: *the Solver allows the normal input/output order to be reversed.*

Using the features presented in the previous portions of this manual, SINDA/FLUINT is a point design simulator: given a design and a set of boundary and initial conditions, SINDA/FLUINT will predict the steady-state and/or transient performance. In other words, *given a design, the user can predict the expected performance.* Using the Solver, the user can perform design tasks such as *given the required performance, what is the optimum design?* The Solver can be used to answer complex questions such as:

- What heater power and interface pad conductance will keep the temperature of a component within its upper and lower limits during various design cases, while requiring minimal electrical power?
- What should the flow rate and coolant loop line diameter be such that a component is kept at 20°C, while keeping the peak pumping power less than 20W?
- What sun-earth orbital angle and dissipation profile maximize the peak temperature of the battery (and thereby determines the worst-case design scenario)?
- Given three thermal balance tests and the transient changes in between, what are the best-estimate values for the bolted joint conductance, the thermal mass of the batteries, and the dissipation rate of the transformer?

Thus, the Solver makes SINDA/FLUINT applicable to preliminary design tasks and final model correlation tasks as well as traditional point design simulation tasks.

As with the rest of SINDA/FLUINT, to use the Solver efficiently the user must think about what they are trying to accomplish and how to best pose that task to the code. It is helpful to know what the code is doing internally, at least topically, to better understand what is being asking of it, especially when things go wrong.

RELIABILITY ENGINEERING: The Reliability Engineering module is a set of high level "solution routines" that in many ways parallel the Solver. However, instead of helping to size or select a design, these routines evaluate the reliability of a given design.

Input parameters (dimensions, properties, boundary conditions, etc.) are rarely known exactly, especially in preliminary design stages before any tests have been performed. Even late in the design stages, large uncertainties exist with respect to unit-to-unit variation due to manufacturing toleranc-ing, and also with respect to uncertainties in environmental conditions, installation, and usage scenarios. Even the design limits themselves are often "soft," meaning that the exact point of failure is rarely known and margins or factors of safety have already been applied. The treatment of uncertainties by using worst-case scenarios and stacked margins results in costly overdesign, perhaps to the point where no viable design exists. An alternative is to treat the uncertainties (both in inputs and requirements) statistically, relying on probability of success ("reliability") to produce a reasonable design.

To use this module, the user defines some input parameters to be uncertain, describing how they vary (e.g., Gaussian or normal distribution, uniform, etc.). Users also define key responses of interest, usually in the form or failure limits (e.g., “don’t allow the temperature at this point to exceed this limit”). Finally, the user describes how the design is to be evaluated in the form of traditional steady state and transient solutions. The code then perturbs the uncertain variables as needed to determine the reliability: the probability that the design won’t exceed the failure limits.

ROBUST DESIGN: If a design is optimized using a failure limit as a constraint, and if the resulting design is constrained by that failure limit, then the design will have roughly 50% chance of failing that limit if any uncertainties exist. An alternative is to use margin on the failure limit, but how much should be applied? The Reliability Engineering module can help determine this limit iteratively, but such an approach would be cumbersome.

Fortunately, an alternative exists. The Solver and the Reliability Engineering module are specifically design to be nested: reliability can be used to help guide the optimization towards a more robust design. Specifically, overall reliability and/or the probability of failing any limit can be used as an objective (“find the design that maximizes reliability subject to other constraints”) or as a constraint (“find the best performing or least weight design that has a minimum overall reliability of 99.7% (three standard deviations).” Even the degree of variation (i.e., range or standard deviation) of an uncertain variable over which the user has control (e.g., a machining tolerance) can be calculated to achieve the desired performance and reliability.

5.1 Introduction to the Solver

The next few subsections (comprising the first half of Section 5) describe the optimization, goal seeking, and data correlation features in SINDA/FLUINT, collectively referred to as the Solver. This subsection provides an introductory overview.

5.1.1 Typical Applications

As described in the previous sections, SINDA/FLUINT would appear to be strictly a point design simulation code: given a description of a system, SINDA/FLUINT predicts how it will behave. For example, given heat rates and conductances, SINDA predicts temperatures. However, employing the Solver SINDA/FLUINT can also be used to reverse the traditional input/output sequence: given temperatures, the Solver can find required heat rates and conductances. More importantly, it can be used as a general design code that minimizes mass or maximizes performance subject to arbitrarily complex constraints defining a viable design. The types of applications addressed by the Solver include:

1. **One Dimensional Design or Sizing.** SINDA/FLUINT can easily predict the temperature at the outlet of a heat exchanger as a function of the flow rate: T=T(F). If instead, the user wanted to find the value of flow rate that yielded a specific temperature at the outlet, F=F(T), the Solver can be used to invert the problem without any additional logic or controls. The user simply assigns the goal of the analysis to be the desired set-point temperature, and SINDA/FLUINT internally iterates until the required flow rate is predicted. For single design variables, see also DVSWEEP (Section 5.15.1).
2. **Generalized Goal Seeking and Adjunct Solutions.** The above example can be generalized for simulation purposes as well as design purposes. For example, in modeling heat pipes and vapor compression cycles, the pressure of the system is yielded by balancing mass and energy. FLUINT can achieve this functionality if high-fidelity elements such as tanks and tubes are used, but otherwise if users want to use less expensive elements (junctions, STUBEs), then they must write their own logic to find the reference pressure (e.g., the pressure of an attached plenum) that satisfies mass and energy balances. Similarly, certain classes of 2D natural convection problems, such as the laminar flow between two panes of glass, can be solved with two 1D flow solutions if extra equations are solved iteratively to find the stagnation point between the upflow and downflow. Using the Solver, the user can let the code perform such calculations to achieve a steady-state solution.

3. **Optimization.** The design and sizing example in item #1 above can be generalized into a more complex question with more than one *design variable*, such as: what are the values of X, Y, Z ... et cetera that yield the maximum (or minimum value) of A, subject to the constraints that X is between C_1 and C_2, and that D is less than E, etc.? For example, a fin cross section could be optimized by finding the thickness along its length (at say 10 evenly spaced cross sections) that maximizes fin efficiency while weighing less than 40 grams, or that minimizes weight while preserving a fin efficiency of at least 0.85, etc. Or, the analyst might wish to know which number of parallel pipes of what diameter maximize heat exchange without exceeding a 10 kPa pressure drop.

4. **Test Data Correlation.** Instead of an optimum design, the user can equivalently seek to find the values of uncertain performance parameters (e.g., contact conductances, surface optical properties, as-built insulation performance, natural convection coefficients) that produce the best fit to test data, whether steady state or transient or a complex set of steady states and transients. In this case, the goal of the analysis is a minimum difference between test data and predictions, perhaps as measured by the sum of squared differences (yielding a least squares curve fit). Although the best fit is usually found by varying all uncertainties simultaneously, the Solver also allows users to approach the correlation incrementally by adjusting the most uncertain parameter first, followed by the next most uncertain parameter, etc.

5. **Worst Case Scenario Seeking.** To demonstrate that a design will operate under all possible conditions, extreme scenarios are chosen that stress the design, such as a “hot case” and “cold case.” These worst cases are then used to verify the design analytically (and possibly as evaluation cases to synthesize the design automatically using optimization methods, as noted in item #1 above). However, identifying these cases can be difficult given performance uncertainties (e.g., solar panel position, coating degradation, dissipation levels, etc.), environmental variations (e.g., ambient temperature and pressure and humidity, orbital “beta” angle, etc.), and usage scenarios (e.g., equipment duty cycles). The Solver can be used to search for the values of these variables that maximize (hot case) or minimize (cold case) critical component temperatures: it can be used to seek the design cases.

5.1.2 Thermal Desktop “Dynamic Mode”

Thermal Desktop (page xlvii of the preface) is often used to calculate geometric factors (e.g., capacitance/conductances from FD and/or FE meshes), radiation fluxes and exchange factors, contact conductances, etc. When using the Solver for jobs such as optimization, test data correlation, and worst case scenario seeking, many design variables and evaluation procedures will need repeated runs of Thermal Desktop. Examples of such jobs include calibrating surface optical properties to test data, finding the worst case beta angle, sizing a heat pipe coldplate, etc.

In these cases, Thermal Desktop and RadCAD calculations can be invoked dynamically from within the SINDA run (perhaps from within PROCEDURE or RELPROCEDURE, to be defined later). The Thermal Desktop User’s Manual contains more information on the “Dynamic SINDA” option within the Case Set Manager.

5.1.3 Optimization Terms

The Solver uses nonlinear programming techniques to solve generalized optimization problems. It is therefore helpful to be familiar with the basic terms used in optimization theory. These terms are defined as follows, and their interrelationship is depicted in Figure 5-1:

Objective

The purpose of the analysis: the mass to be minimized, the performance metric to be maximized, or a performance target to be met, etc.

Design Variables

The inputs that can be changed as needed to achieve the purpose. Or, the unknowns that need to be determined. There needs to be at least one design variable, and most problems will have from one to ten of them even though many more are supported.

Constraints

The upper and lower limits on the design variables, or any other criteria distinguish a viable design from a useless one. Perhaps these constraints are based upon a complex function of the predicted performance of the design, and not just dimensional or configuration constraints. Although most engineering analyses tend to involve constraints and perhaps even contain thousands of them, it is possible to have an unconstrained problem (i.e., zero constraints).

Evaluation Procedure

The analytic operations that are required to evaluate a given design. This might be a simple steady state, a transient simulation, or some complex mixture of the two. Ultimately, to “evaluate” means to measure the value of the objective for a given set of design values, and to determine whether or not that design satisfied or violated the constraints, and by how much.

As an example, consider a simple rectangular heat transfer fin (extended surface) with a given root temperature and fixed environment. It is desired to minimize the mass of the fin by varying its length and thickness while keeping the fin efficiency above 80%.

In this example, the *objective* is the mass, defined by the product of the length, the thickness, the width, and the density. There are two *design variables*: the length and the thickness. Every other parameter is either a constant or is described as a function of these two variables. The *constraint* is the fin efficiency, which is required to be greater than 0.8 (80%).

The *evaluation procedure* is as follows: Given a length and a thickness, calculate the current mass (the value of the objective) per the above formula. Find the steady state for that configuration using a SINDA/FLUINT solution routine, and calculate the fin efficiency (the constraint).

In this example, a SINDA/FLUINT steady state is needed to evaluate a constraint but not the objective. In other cases, the objective might be based on required performance and therefore determined at least in part by a SINDA/FLUINT solution, and the constraint might be based on a configuration limit and therefore not require a SINDA/FLUINT solution. Any such combinations

are valid, including the lack of any SINDA/FLUINT solution altogether, as long as the evaluation procedure updates the objective and the constraints for a given set of design variables. The full range of SINDA/FLUINT functionality is available to help the user perform this evaluation.

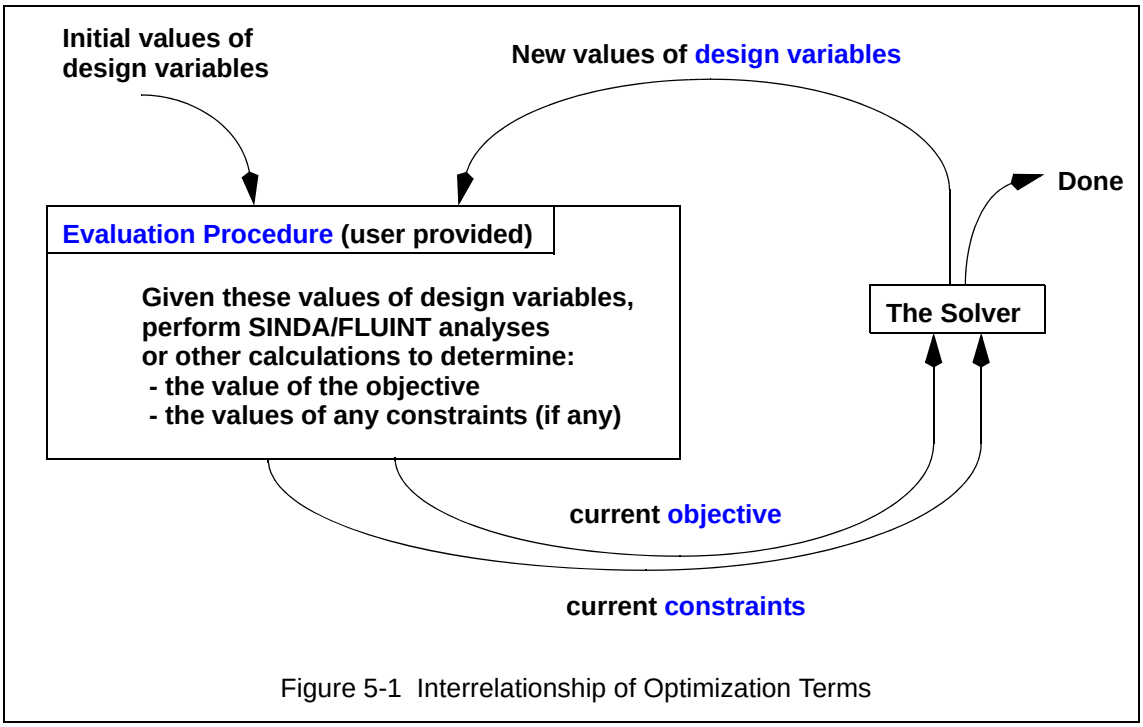


Figure 5-1 Interrelationship of Optimization Terms

5.1.4 Overview of the Solver

The Solver is a high-level solution routine that invokes an arbitrary analysis procedure to evaluate a design or model (see Section 5.2).

This procedure, input in a header block called PROCEDURE (Section 5.3), in its simplest form might simply consist of a call for a steady state solution (“CALL STEADY”). However, the procedure can be an arbitrarily complex process with one or more steady state or transient runs, complete with BUILD/BUILDF commands, internal loops or iterations, and internal parametric restarts or other I/O operations. In fact, the only distinction between PROCEDURE and OPERATIONS is that the logical instructions and analysis sequences in PROCEDURE are normally invoked many times, whereas OPERATIONS remains a one-pass set of instructions (i.e., executed once). The call to SOLVER itself is made in OPERATIONS, and SOLVER then internally calls PROCEDURE any time it needs to *evaluate* the user’s design or model. Using the Solver is therefore like setting up an automatic sequence of SINDA/FLUINT runs, wherein each single run is now a self-contained procedure.

What is meant by the term “evaluate a design or model” is ultimately controlled by the user according to the task assigned to the Solver. If the goal of the analysis task is to find the minimum weight of some design subject to certain performance constraints, then “evaluate” means to calculate the performance of the design (in terms of how well it met or violated the assigned constraints) and

to return to the Solver the weight of the current design. This evaluation process might therefore simply be the calculation of a single steady-state followed by the calculation of the current mass and the current performance constraints.

To use the Solver, as a minimum the user:

1. Builds a model that can be varied using registers (Section 4.9) as needed to fulfill the desired analysis goal.

   Although it is a good idea to use registers and register-containing expressions extensively throughout a model, it is also possible to retroactively add registers (perhaps as multiplying factors initialized to unity).

2. Chooses a subset of the registers that will participate in the solution. These are the *design variables*: one or more parameters that will be adjusted by the Solver during its operation. The designation of these special registers and any limits that might apply to their values are defined in DESIGN DATA (Section 5.5).

3. Defines the goal of the Solver. The goal is defined by default as a minimization (GOAL = -1.0E30), but could also be defined (in HEADER SOLVER DATA, Section 5.4) as a maximization (GOAL = 1.0E30). Or, GOAL could be defined as a target value by setting it to be a finite value for goal seeking (e.g., GOAL = 3.14).

4. Defines the solution procedure by which the Solver should evaluate a design to compare the current design (as dictated by the current Solver-selected set of design variables) against the goal. This evaluation procedure is defined in HEADER PROCEDURE (Section 5.3), a logic block similar to OPERATIONS.

5. Lets the Solver know how it is doing by providing a means for measuring any given design against the goal: the objective value. For example, if the goal is minimum mass, the objective would be the current mass. This single-valued objective, OBJECT, might be defined once in SOLVER DATA if it can be expressed in terms of the design variables and processor variables. However, OBJECT is more commonly updated in logic blocks such as at the end of the PROCEDURE. ***The Solver changes the design variables until OBJECT is as close as possible to GOAL*** (without violating constraints, if any).

6. Adds "CALL SOLVER" to OPERATIONS, with perhaps a call to REGTAB, DESTAB, and/or CSTTAB (all described in Section 7, with an overview in Section 5.7) afterwards to print final values of the design variables and other Solver information.

Although not strictly necessary, it is also possible to define arbitrarily complex constraints on the designs that are to be investigated. Indeed, most real engineering applications are heavily constrained. A constraint might be phrased in English as "the efficiency can be no less than 90%" or "the net positive suction head (NPSH) at the inlet of a pump can be no less than 2.5m" or "the thickness can be no less than the width, but no more than 10 times the width."

Sometimes, these constraints represent limits on the design variables themselves. For example, if both the diameter and thickness of a pipe were design variables, then in DESIGN DATA (Section 5.5) the user could impose limits and constraints on the values of those variables, such as "the inner diameter can be no less than 0.1cm but no more than 10 times the wall thickness."

However, if the constraint is based on performance of a design, then another mechanism is needed. This mechanism is provided by the CONSTRAINT DATA block described in Section 5.6. The user can create an arbitrary number of *constraint variables* or constraining expressions, and use these along with OBJECT to tell the Solver how any particular design measured up during the evaluation procedure. For example, a variable called EFF could be created in CONSTRAINT DATA and set to be greater than or equal to 0.90 (or 90%). At the end of PROCEDURE, for example, the user could then calculate the efficiency of the design and set it equal to EFF in logic. The Solver would then be able to compare not only how a design measured up in terms of OBJECT vs. GOAL, but also whether it met or violated any constraints and by how much.

Many more tools exist, such as the ability to temporarily hold design variables and to customize the internal workings of the Solver. These additional tools are documented in the rest of this chapter.

5.1.5 A Brief Example

The operation of the Solver may at first appear complex because it involves many options and input blocks. However, it can be quite simple to employ. As an example, assume that a user needed to find the diameter of a given length of stainless steel pipe carrying liquid propane such that: (1) weight was minimized, yet (2) the pressure drop did not exceed some threshold such as 10kPa as needed to prevent the liquid from flashing.

The user would then construct a SINDA/FLUINT model of the pipe using registers to define the diameter and its wall thickness (for mass calculation purposes), and perhaps other key parameters as well. The diameter, perhaps called DIAM, would then be declared as a design variable in DESIGN DATA.

An expression defining the constraint (the allowed pressure drop in the pipe) would be defined in CONSTRAINT DATA. If, for example, the difference between the pressure of the upstream lump #10 and the pressure of the downstream lump #20 in submodel PIPE should not exceed 10kPa, then the expression might appear as “pipe.pl20-pipe.pl10 <= 10000” or equivalently “pipe.pl20 <= pipe.pl10 + 10000.”

The user would then BUILD and BUILDF the appropriate submodels in OPERATIONS, and make a call to SOLVER. In SOLVER DATA, the user would define the GOAL to be -1.0E30 (minimize, the default). The evaluation procedure supplied in HEADER PROCEDURE would then be a call to a steady state solution (“CALL STEADY”) followed by an update of OBJECT to contain the current value of pipe mass. In other words, OBJECT would be calculated based on the current diameter (DIAM) and the current wall thickness.

Given that the default parameters in SOLVER DATA were appropriate for this problem, SINDA/FLUINT would then internally call PROCEDURE iteratively until it found the minimum mass design that did not violate the constraints. Although the user may have required a call to STEADY to calculate the current pressure drop, from the perspective of the Solver the only data that was relevant was the returned values of OBJECT and the expressions defining the constraints, since those were the values used to make decisions about how to vary the design variable DIAM.

5.2 SOLVER Routine

The main driver for the Solver is a top-level "solution routine" called SOLVER, *which may only be called from OPERATIONS.* SOLVER internally calls an evaluation procedure defined by the user separately in PROCEDURE (Section 5.3) according to the control parameters defined in SOLVER DATA (Section 5.4). SOLVER changes *design variables* defined in DESIGN DATA (Section 5.5) while meeting the *constraints* defined in CONSTRAINT DATA (Section 5.6) until the *objective value* (OBJECT, Section 5.4) meets the *goal* (GOAL, Section 5.4), or at least comes as close to it as it can. All of these inputs are described later; this subsection describes only the Solver routine itself and its own logic blocks.

Just as the instructions in OPERATIONS are converted to a Fortran (or C) routine named OPER, the instructions in PROCEDURE are converted into a routine named PROC. The Solver typically changes one or more design variables, propagates the changes throughout the rest of the model via an internal call to UPREG (Section 7.12.1),* and then calls for a new evaluation via a call to PROC to get new values of OBJECT and constraint variables and expressions. Based on accumulated data, the Solver selects how next to change the design variables. (Section 5.8 contains supplementary information on how the solver works in terms of finding an optimal solution.)

Analogous to the VARIABLES 0/1/2 and FLOGIC 0/1/2 blocks, the user can add optional logic blocks called SOLOGIC 0, 1, or 2 that are called within SOLVER. These are submodel-independent (global) blocks: only one of each of these blocks may be input in a single model. These blocks are described in Section 5.2.1. These blocks are converted into SLLOG0, SLLOG1, and SLLOG2.

An additional global logic block is available that is analogous to the submodel-dependent OUTPUT CALLS blocks. This block, SOLOUTPUT CALLS, is described in Section 5.2.2. It is converted in to SOLOUT.

* Unlike other automatic internal calls to UPREG, calls from within the Solver cannot be turned off using the NOUPDATE feature in OPTIONS DATA, though they *will* comply with any disconnections performed using the DEREG family of routines (Section 7). However, NOUPDATE *does* apply to UPREG calls made within solution routines, and these are normally called from within the Solver (see the PROCEDURE block).

In pseudo-code, a simplified rendition of the routine SOLVER is as follows:

```
SUBROUTINE SOLVER
[initialization]
[for a maximum of NLOOPO iterations:]
       CALL SLLOG0    $ Optional initialization
       [update limits and constraints etc.]
       [change or perturb design variables]
       CALL SLLOG1    $ Optional customization
       CALL UPREG     $ update registers and formulas
       CALL PROC      $ call for a new evaluation:
                      $ new OBJECT, new constraints
       [if time to output] CALL SOLOUT $ Optional
       [if done return]
[issue nonconvergence warning and return]
```

In many ways, each SOLVER step is like a single steady-state relaxation step or a single transient time step in that the user can initialize, tailor, and call for output using solver-level logic blocks. Of course, since PROCEDURE can be as arbitrarily complex as a separate SINDA/FLUINT run, the Solver clearly works at a higher level than a traditional steady-state or transient solution routine.

The flow chart for SOLVER is provided as Figure 5-2.

Example:

```
HEADER OPERATIONS
BUILD CONF, MOD1, MOD2, MOD2
       CALL SOLVER
```

5.2.1 SOLOGIC Blocks

HEADER SOLOGIC 0 should contain any initializations or customizations of constraints, design limits, etc. that are required *before* an internal call to UPREG (Section 7.12.1) that updates Solver data (formula-based limits in DESIGN DATA and CONSTRAINT DATA, formula-based parameters in SOLVER DATA). Otherwise, most one-time initializations can be performed in OPERATIONS before the call to SOLVER, and PROCEDURE should contain any initializations of the model itself.

HEADER SOLOGIC 1 should contain any modifications to the design variables or other parameters that are necessary before a full network update is performed via an internal call to UPREG (Section 7.12.1). A call to CALREG (Section 7.12.1) should be made if changes to the design variables need to be propagated to other registers (but not to other parts of the model, such as the network) before performing these modifications.

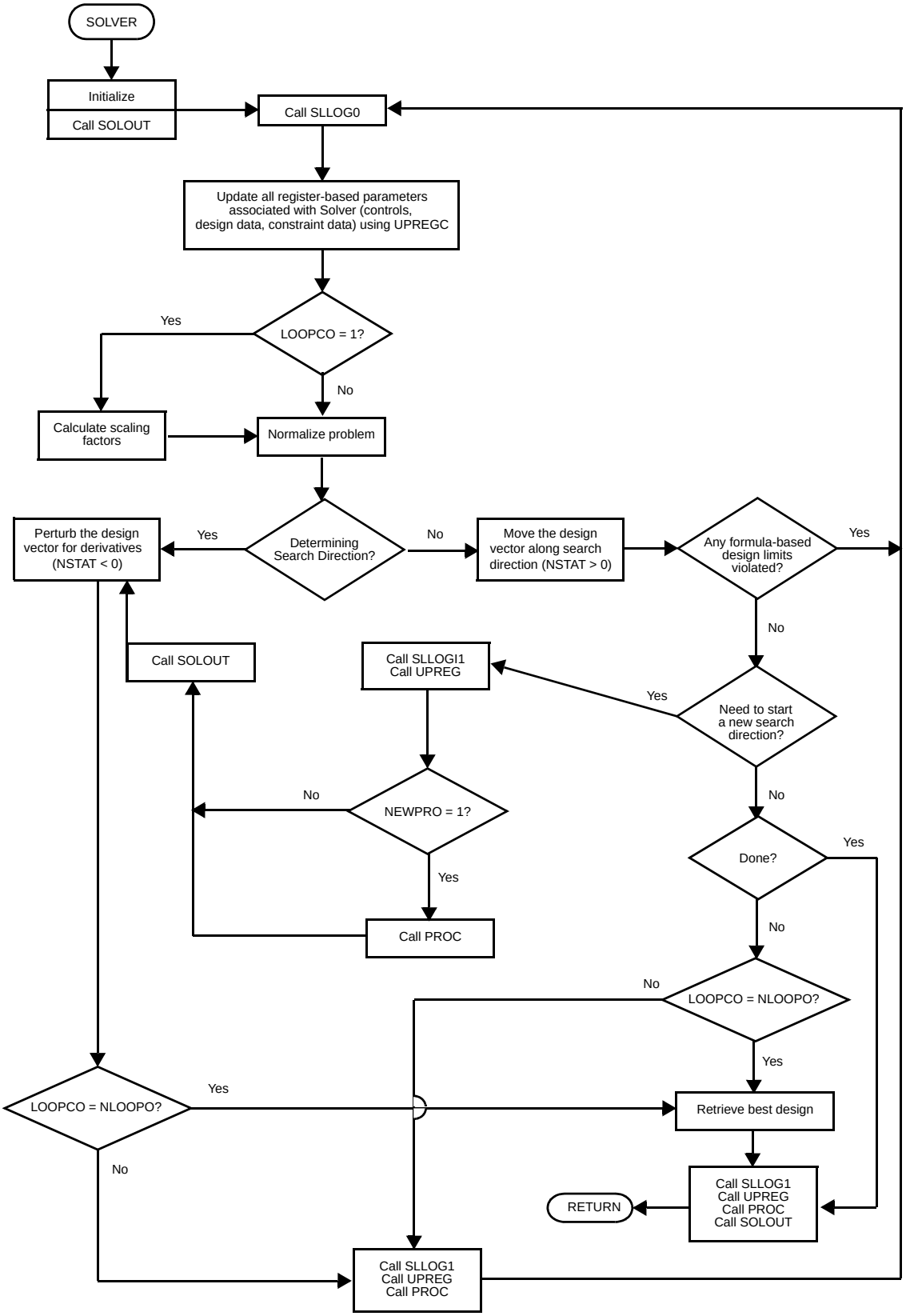


Figure 5-2 Flow Chart for SOLVER

Changes should not be made to the design variables. However, sometimes extra logic is needed depending on the values of those variables as changed internally by the Solver. For example, if the widths on each end of a wedge (of length LONG, thickness THICK, and conductivity COND) were defined as design variables W1 and W2, then the conductance across that wedge might have been input as the expression "COND*(W1-W2)*THICK/(LONG*/LN(W1/W2))." However, if W1 is ever equal to W2 or nearly so, then that expression will fail and a replacement expression "COND*W1*THICK/LONG" (for a rectangular bar) is needed. Detecting and replacing or modifying such an expression before UPREG is called is one use of SOLOGIC 1. Other initializations can be made separately in the first parts of PROCEDURE.

SOLOGIC 2 is not currently used: it is reserved for future options. Logic input into HEADER SOLOGIC 2 is preprocessed, but is not *currently* invoked by any prepackaged SINDA/FLUINT option.

SOLOGIC 0, 1, and 2 are converted into routines SLLOG0, SLLOG1, and SLLOG2, respectively.

Since these blocks are global (i.e., not dependent on any submodel), submodel prefixes and/or the DEFMOD command must be used when referring to any submodel-dependent keyword.

5.2.2 SOLOUTPUT CALLS

An additional global logic block is available that is analogous to the submodel-dependent OUTPUT CALLS blocks. Like those blocks, output operations (particularly REGTAB, DESTAB, and CSTTAB, as described in Section 7.13) are commonly placed in the HEADER SOLOUTPUT block, but few restrictions exist as to what can be placed in this block.

Since this block is global (i.e., not dependent on any submodel), submodel prefixes and/or the DEFMOD command must be used when referring to any submodel-dependent keyword.

SOLOUTPUT CALLS is converted into a routine named SOLOUT.

Normally, SOLOUT is called at the end of the Solver. However, it can also be called several times during the course of the solution to show results in progress. The value of LOOPCO, the current Solver iteration count defined in Section 5.4, is unpredictable at the points where SOLOUT is called: it is not called at a regular interval but rather at an appropriate point in the optimization where an intermediate design point has been found and the Solver is about to embark on a new search.

At those intermediate points, the values of the OBJECT and constraint functions may be stale (old, not current). At the cost of several extra (and otherwise unnecessary) calls to PROCEDURE, the NEWPRO control constant (Section 5.4) may be used to update these parameters to their current values before these intermediate calls to SOLOUT.

5.3 HEADER PROCEDURE

In addition to HEADER OPERATIONS, the user may add another block called HEADER PROCEDURE. All rules and capabilities associated with OPERATIONS apply to the PROCEDURE block as well, and so that description (Section 4.1.3.1) will not be repeated here.

PROCEDURE is converted into a routine named PROC.

Although the primary purposes of the HEADER PROCEDURE block is to define the evaluation sequence to be performed iteratively by the Solver, it can also be used as a convenience to replace repetitive procedures in OPERATIONS. The user can call for these procedures to be performed by inserting "CALL PROC" in OPERATIONS. By default, the logic placed in PROCEDURE is not called unless the user either invokes the Solver or calls the subroutine PROC.

Caution: Commands such as BUILD, BUILDF, DRPMOD, ADDMOD, HTRNOD, HLDLMP, etc. that are performed in OPERATIONS remain in effect upon the call to PROCEDURE, and any such commands made in PROCEDURE remain in effect upon returning to OPERATIONS.

Guidance: Calls to RESPAR or RESTAR within PROCEDURE are common when the evaluation procedure involves transient runs and initial conditions must be reset. If registers were saved in the original SAVPAR, SVPART, SAVE, or RESAVE call (i.e., the '-R' option was *not* invoked), then restoring those values would overwrite design variable manipulations made by the Solver. To avoid this problem, RESPAR and RESTAR automatically propagate updates for any expression containing (directly or indirectly) the design variables after reading back in the data. This allows the user to save and reset other (non-design variable) registers via a prior SAVPAR (for example) call. See also Section 4.9.6.3 and Section 4.9.7.

Example:

```
HEADER OPERATIONS
        CALL PROC
HEADER PROCEDURE
BUILD CON,MYMOD
        CALL STEADY
        OBJECT = rho*volume
DEFMOD HOTR
        delT = t44-t55
```

If REPROCEDURE (Section 5.22) is the same as PROCEDURE, then one can simply be called from the other. For example, if RELPROCEDURE has already been input and is identical to PROCEDURE:

```
HEADER PROCEDURE
        CALL RELPROC
```

5.4 HEADER SOLVER DATA

HEADER SOLVER DATA contains parameters used to control the operation of the Solver *and of the Reliability Engineering module*, which is otherwise described later. Also relevant to this section are parameters returned by the Solver for use in user logic, and parameters that are expected to be provided by the user in their logic during the course of the Solver operation.

HEADER SOLVER DATA is analogous to HEADER CONTROL DATA, GLOBAL and following its formats and rules of operation. Therefore, descriptions repetitive to those made in Section 4.3 will not be made here.

Normally, the defaults values suffice. Therefore, values which the user should consider inputting or updating are listed first in Table 5-2, with more obscure options listed later. See also Section 5.8, which explains how some of these parameters affect Solver execution. For Reliability Engineering controls,

Most problems will be constrained. A truly unconstrained problem has no CONSTRAINT DATA (Section 5.6) *and* no upper or lower limits on design variables are defined using registers in DESIGN DATA (Section 5.5).

Table 5-2 Solver Control Data Definitions

Name	Module	Definition and Description	Default or Type
Input under SOLVER DATA and/or Updated in Logic			
NLOOPO	Solver	Maximum iterations (calls to PROCEDURE). As a guideline, this should generally be at least 10*N*(1+MDERO)+50, where N is the number of design variables defined in HEADER DESIGN DATA, and MDERO is defined below.	100
NLOOPR	Rel. Engr.	Maximum iterations (calls to RELPROCEDURE). SAMPLE will take up to this many samples, although it may converge first. DSAMPLE will take exactly this many samples. RELEST will fail if NLOOPR is less than N+1, where N is the number of random variables.	100
OBJECT	Solver	Value of the design or optimization objective. For example, if the purpose of the analysis is to minimize mass, OBJECT should contain the value of the mass according to the current set of design variables. OBJECT normally must be updated in logic within logic blocks or at end of PROCEDURE, but can alternatively be defined once in SOLVER DATA using a formula in certain cases. For best results, OBJECT should be a continuous and differentiable function of the design variables or other parameters (e.g., performance predictions) that are themselves a strong function of the design variables.	NONE; real
GOAL	Solver	Desired value for OBJECT (a fixed value, or -1.0E30 to minimize, or 1.0E30 to maximize). *GOAL should not change during the course of the solution.*	-1.0E30 (minimize)

Table 5-2 Solver Control Data Definitions

METHO	Solver	Solver method to use: Unconstrained (no CONSTRAINT DATA nor any formula-based limits in DESIGN DATA): 1 - BFGS method[a] 2 - Fletcher-Reeves method[b] (default) Constrained: 1 - Modified method of feasible directions[c] 2 - Sequential linear method[d] (default) 3 - Sequential quadratic method[e] Note that the default method is often not the best. Try other methods if performance of the default method is unsatisfactory.	2
RERRO	Solver	Maximum error in OBJECT (*relative* to *current value*) between design solutions (not consecutive Solver iterations) to be considered converged. Values too small will result in more accurate answers but at a cost of more calls to PROCEDURE.	0.01 (1% of *current* value)
AERRO	Solver	Maximum error in OBJECT (*relative* to *initial value*) to be converged. Use instead of RERRO when GOAL= 0.0 or if OBJECT is expected to be zero in the final converged model, otherwise *either* RERRO *or* AERRO must be satisfied (i.e., the *larger* of the RERRO and AERRO result is used for a convergence test).	0.0001 (0.01% of *initial* value after the first call to PROCEDURE)
RERRR	Rel. Engr.	Maximum change in all responses (*relative* to *current value*) between consecutive iterations for SAMPLE to be considered converged. Used only in SAMPLE. Values too small will result in more accurate answers but at a cost of more calls to RELPROCEDURE.	0.001 (0.1% of *current* value)
AERRR	Rel. Engr.	Maximum change in all responses (*relative* to *initial value*) between consecutive iterations for SAMPLE to be considered converged. Used only in SAMPLE when the parameter being tested is very small. Used instead of RERRR as an absolute check (not a relative check) when the response itself is less that RERRR. Thus, the *larger* of the RERRR and AERRR result is used for a convergence test.	0.00001 (0.001% of *initial* value)
RDERO	Solver	Minimum size of perturbations to design variables (*relative* to *current value*) for estimating derivatives (changes in OBJECT and constraints) as part of determining the search direction. As guidance, RDERO should be larger than RERRF, REBALF, EBALSA, etc. Consider larger values if MDERO=1, or if OBJECT or an important constraint is a weak or noisy function of the design variables (which by itself may indicate a faulty choice of design variables or a poorly formulated problem statement).	0.02 (2% of *current* value)
ADERO	Solver	Minimum size of perturbations to design variables (*relative* to *initial value*) for estimating derivatives as part of determining the search direction. Uses the *larger* of the RDERO and ADERO result.	0.0001 (0.01% of *initial* value)
MDERO	Solver	Method for estimating derivatives (sensitivity of OBJECT and constraints to changes in design variables): 0=forward, 1=central. If the accuracy of each evaluation procedure is questionable (noisy) and cannot be improved, use MDERO=1 and perhaps increase RDERO, but expect the number of calls to PROCEDURE to almost double: NLOOPO should also be increased.	0 (forward)
RDERR	Rel. Engr.	Minimum size of perturbations to random variables (*relative* to *current value*) for estimating derivatives (changes in reliability constraints). Used only in RELEST. As guidance, RDERR should be larger than RERRF, REBALF, EBALSA, etc. Consider larger values if an important reliability constraint is a weak or noisy function of the random variables	0.02 (2% of *current* value)

Table 5-2 Solver Control Data Definitions

ADERR	Rel. Engr.	Minimum size of perturbations to random variables (*relative* to *initial value*) for estimating derivatives (changes in reliability constraints). Used only in RELEST. Uses the *larger* of the RDERR and ADERR result. If the mean of a random variable is zero, consider well the value of ADERR, which assumes the units of the random variable.	0.01 (1% of *initial* value)
MDERR	Rel. Engr.	(Reserved for future use.)	N/A
RCHGO	Solver	Maximum size of changes to design variables (*relative* to *current value*) during pursuit of a solution (when moving along a search direction). Should be larger than RDERO. Small values will approach a solution more gradually, but at a cost of more calls to PROCEDURE and a potential danger of stalling early (false convergence) due to inadequate progress. Too large a value may cause changes that cannot be tolerated by the model or the user's evaluation procedure.	0.9 (90% of *current* value)
ACHGO	Solver	Maximum size of changes to design variables (*relative* to *initial value*) during pursuit of a solution (should be larger than ADERO). Uses the *larger* of the RCHGO and ACHGO result.	0.1 (10% of *initial* value)
RCACTO	Solver	Relative measure of proximity of a constraint for it to be considered active. Must be greater than RCERRO. Too large or too small a value will cause additional calls to PROCEDURE: an optimum value exists for each problem. Too large a value will slow progress, but too small a value can cause the solver to overshoot or oscillate inefficiently. See also Figure 5-4.	0.1 (10% of *current normalized* value)
RCERRO	Solver	Relative error tolerated in violation of a constraint in a final solution. Values too small will result in more accurate answers (if constrained) but at a cost of more calls to PROCEDURE.	0.003 (0.3% of *current normalized* value)
RCVIO	Solver	Relative error that can be tolerated in violation of a formula-based design variable limit during pursuit of a solution (when moving along a search direction). Should be greater than RCERRO. Increase for added speed if violating the design variable limits does not cause solution problems. Otherwise, decrease if excessive violation of design variable limits cannot be tolerated by the evaluation procedure. Does not apply to hard-wired (non-formula-based) design variable limits. (See also RCHGO/ACHGO. Use small RCHGO/ACHGO if answer lies near an infeasible solution, or if derivatives change quickly with changes to the design variables, and use small RCVIO if formula-based limits are being violated excessively during the solution.)	0.2 (20% of *current normalized* value)
PUSHO	Solver	Ambitiousness of solver during pursuit of a solution (when moving along a search direction): less than 1.0 implies caution or slow initial changes, greater than one implies acceleration or aggressiveness. Must be positive.	1.0
NERVUS	BOTH	Tolerance of problems during SINDA/FLUINT network solutions: 0 ... stop at first sign of trouble 1 ... tolerate converged, stable but unbalanced thermal 2 ... tolerate also nonconvergence in TRANSIENT 3 ... tolerate also nonconvergence in STDSTL and STEADY Set the value of NERVUS to as high as tolerable in order to avoid aborting a run, as long as the accuracy of the resulting solutions is adequate. The default value is safe, but too often causes runs to abort.	0 (abort at first sign of trouble). For DVSWEEP and design scanning, ignores rather than aborts.
NEWPRO	Solver	If NEWPRO=1, an additional call to PROCEDURE (unnecessary for the Solver) will be made before intermediate calls to SOLOUTPUT CALLS to assure that the data output or otherwise manipulated or updated in that block is up to date. Does not apply to final calls to SOLOUTPUT CALLS, which will contain the latest data.	0 (no extra calls to PROCEDURE)

Table 5-2 Solver Control Data Definitions

NCONVO	Solver	Number of final search directions to check before assuming convergence. Use 3 or perhaps 4 to better check the final answer if it is suspect, at the cost of more calls to PROCEDURE.	2
NWRKRO	Solver	Real workspace array size for NLP program. Values too low can cause solver to stop. If this happens, increase as directed by program output.	calculated by program
NWRKIO	Solver	Integer workspace array size for NLP program. Values too low can cause solver to stop. If this happens, increase as directed by program output.	calculated by program
NSEED	Rel. Engr.	Random sequence seed. This is calculated by the program automatically. No two SAMPLE or DSAMPLE runs will be exactly alike for this reason. If for some reason a run must be repeatable, this value can be input rather than calculated. Not used by RELEST. ***An 8 digit limit applies.*** To specify a value larger than 8 digits, set NSEED at the beginning of OPERATIONS using an “F” in column 1 to suspend translation. Values larger than 2^{31} are illegal in any event.	calculated by program; can be input (large positive integer)
Provided by SINDA/FLUINT for Use in Logic			
LOOPCO	Solver	Current iteration count (calls to PROCEDURE)	integer
LOOPCR	Rel. Engr.	Current iteration count (calls to RELPROCEDURE)	integer
DELOBJ	Solver	Last change in best value of OBJECT (positive or negative). This is not updated every call to PROCEDURE, but is updated every call to SOLOUTPUT CALLS. It is not directly comparable to RERRO/AERRO, but is a useful measure of progress.	real
NSTATO	Solver	Status of solver: -n: performing perturbation of the n^{th} design variable (each perturbation requires one call to PROCEDURE, and N perturbations are required to determine a new search direction, where N is the number of design variables) 0: finished or terminating +n: performing the n^{th} one dimensional search (two or more search directions are required for each call to SOLVER, and each search takes one or more calls to PROCEDURE)	integer
OVEREL	Rel. Engr.	The tallied overall reliability: the ratio of samples in which no reliability constraints were violated compared to the total number of samples. Will be -1.0 before being calculated. Only valid with SAMPLE and DSAMPLE: not calculated by RELEST. Use with caution with DSAMPLE. Will underestimate overall reliability if some reliability constraints are parallel (exceeding one is not failure, exceeding all parallel constraints *is* failure).	real

a. Broyden, C.G., “The Convergence of a Class of Double Rank Minimization Algorithms, Parts I and II,” J. Inst. Math. Appl., Vol 6, pp. 76-90,222-231, 1970.
Fletcher, R., “ A New Approach to Variable Metric Algorithms,” Computer J., Vol. 13, pp. 317-322, 1970.
Goldfarb, D., “A Family of Variable Metric Methods Derived by Variational Means,” Math. Comput., Vol 24, pp 23-26, 1970.
Shanno, D.F., “Conditioning of Quasi-Newton Methods for Functional Minimization,” Math. Comput., Vol 24, pp 647-656, 1970.
b. Fletcher R. and Reeves, C.M, “Funtional Minimization by Conjugate Gradients,” Br. Computer J., Vol. 7, No. 2, pp. 149-154, 1964.
c. Zoutendijk, K.G, **Methods of Feasible Directions**, Elsevier Publishing Co., Amsterdam, Netherlands, 1960.
Vanderplaats, G.N., “An Efficient Feasible Direction Algorithm for Design Synthesis,” AIAA Journal, Vol. 22, No. 11, Nov. 1984.
d. Kelly, J.E., “The Cutting Plane Method for Solving Convex Programs,” J. SIAM, Vol. 8, pp. 702-712, 1960.
e. Powell, M.J.D., “Algorithms for Nonlinear Constraints that use Lagrangian Functions,” Math. Prog., Vol. 14, No. 2, pp. 224-248, 1978.

Example:

```
HEADER SOLVER DATA
        RERRO = 0.02
        METHO = 2, NLOOPO = 50
        GOAL = 1.0E30
        OBJECT = FRED.T1 - WILMA.T22
HEADER PROCEDURE
        CALL STEADY
```

5.4.1 Using OBJECT and GOAL

OBJECT is a *variable* input or updated as needed to tell the Solver how well the current design suits the intents, or how well the current model matches data, etc.

GOAL is a *fixed* value used to signal the program whether to minimize OBJECT (GOAL less than -1.0E20), to maximize OBJECT (GOAL greater than 1.0E20), or to match GOAL (GOAL = desired value of OBJECT).

The Solver will change the values of design variables until the value of OBJECT, as reported by the user in the evaluation procedure, ***is as close as possible to GOAL***.

For example, if performance must be maximized (GOAL = 1.0E30), then by the end of PROCEDURE OBJECT should contain a value that measures system performance as a function of the current design. If a correlation to test data is being performed, and the goal is therefore to minimize (GOAL = -1.0E30, the default) the differences between predictions and test data, then OBJECT should contain a value that measures the mismatch between the two as calculated using the latest call to PROCEDURE. If the user is seeking to find a value of some parameter that yields a result B, then the GOAL is set equal to B and OBJECT is updated to contain the latest value of the target result produced by a call to PROCEDURE.

In all instances, GOAL is defined only in SOLVER DATA (or at least initialized in OPERATIONS outside of the call to Solver), whereas OBJECT must either be updated by the user directly in logic, or indirectly as a result of changes to registers or processor variables if OBJECT has been defined by a formula in SOLVER DATA.

Normally, the user updates OBJECT at the end of the solution sequence defined in HEADER PROCEDURE. However, OBJECT might also be updated or modified at other points during the evaluation procedure, and in rare instances can be defined itself by an expression in SOLVER DATA and never *directly* updated in logic. For example, it is perfectly legal to assign OBJECT to be one of the design variables (Section 5.5) in SOLVER DATA, or perhaps a function of the design variables. It is often convenient to define OBJECT in SOLVER DATA based on processor variables, eliminating the need to update OBJECT explicitly in logic blocks.

For example, to find the flow rate in a fluid loop that causes an outlet temperature to be 70 degrees, specify the flow rate in the loop as a design variable (or as a function of one or more design variables), set GOAL=70.0, and tell the program what the current value of the outlet temperature is, perhaps via a Fortran assignment statement at the end of PROCEDURE:

```
OBJECT          = MODL.T(401)
```

The code will then adjust the model flow rate until OBJECT (which in this case is simply the temperature of node 401 in submodel MODL) equals GOAL. If one design variable were used, the solution is unique if it exists. If more than one design variable were chosen and any of these could have been changed to meet the objective, then the results may not be unique and may simply represent the first solution found. If the system is overconstrained, then the code may be unable to find the desired solution and if so, it will report this result.

More often, the value in OBJECT (mass, efficiency, power, etc.) will either be maximized or minimized, as signaled to the code by setting a GOAL value either smaller than -1.0E20 or larger than 1.0E20. In this case, any number of design variables and constraints may be used, although the solution speed can be slow if the number of design variables becomes excessive.

As another example of the use of these variables, consider that the best-fit model to some transient test is defined on the basis of the minimum error at the end of the transient (e.g., final temperatures). In this case, GOAL would be set to -1.0E30 (the default, meaning minimize) or to 0.0 (essentially the same thing in this case, although a true minimization would be better) and a single line could be added at the end of PROCEDURE (if a single data point were the basis of the comparison):

```
OBJECT          = (Tpredict - MYMOD.T304)**2
```

The calculation of the objective variable may be complex, perhaps performed cumulatively during the course of the PROCEDURE execution. For example, assume the best-fit model to the above transient test is defined instead as the minimum temperature error at one point in the system (as measured every X seconds), accumulated over the whole event. This could be achieved by initializing OBJECT to be zero at the start of PROCEDURE, and then adding the following line in the appropriate OUTPUT CALLS block (such that it is called at even intervals to produce an even weighting):

```
Tpredict        = ....
OBJECT          = OBJECT + (Tpredict - T304)**2
```

More information on performing test data correlation is presented in Section 5.10.

5.5 HEADER DESIGN DATA

Of the registers defined in REGISTER DATA (Section 2.8), some or all may be selected to act as *design variables*. If the Solver is to be used, at least *one* design variable must be selected. These design variables will be automatically modified by the Solver as needed to make the objective (OBJECT) meet the goal (GOAL) using internal calls to UPREG and PROC. Therefore, *the user should not independently modify the design variables while the Solver is executing.*[*]

Both upper and lower limits may be placed on these design variables. Normally, these limits are numeric constants if they are present at all. However, these limits may themselves be defined by registers or expressions, perhaps even containing references to other design variables.

In the current version, only real registers may be used as design variables, and only continuous values are supported. Refer to Section 5.9.1 for methods for dealing with integer or discrete variables.

The format is:

```
HEADER DESIGN DATA
        [ lower_limit <=] reg_name [<= upper_limit]
```

where:

lower_limit . . . optional lower limit or expression
reg_name . . . . name of valid real register to be used as a design variable
upper_limit . . . optional upper limit or expression

Only one design variable and its limits may be entered per line. Limits are optional. If both limits are defined, the upper and lower limit may be equal, but the lower limit cannot exceed the upper limit at any time.

Restriction: To avoid scaling problems, the initial value of a design variable cannot be zero.

Example:

```
HEADER DESIGN DATA
                fred
        0.01 <= wilma
        -10.0 <= barney <= 10.0
        bambam <= 0.5*pebbles
        -pi <= betty <= pi
```

* Design variables can, however, be changed *between* calls to Solver. See Section 5.9.1.

For example, if the inner diameter of a pipe were a design variable defined as DIAM and its wall thickness were defined as THICK, and both were design variables, one might wish to constrain the wall thickness to be no less 0.001 nor less than 10% of the diameter (which itself cannot be less than 0.01). This could be expressed as follows:

```
0.01                    <= DIAM
MAX(0.001,0.1*DIAM)     <= THICK
```

Or, reversing the constraint (i.e., applying it to DIAM instead of THICK) works equally well:

```
0.01                    <= DIAM         <= 10.0*THICK
0.001                   <= THICK
```

However, both variables should **not** be simultaneously constrained by one another:

```
C DON'T OVERCONSTRAIN! NOT RECOMMENDED:
      0.01                    <= DIAM         <= 10.0*THICK
      MAX(0.001,0.1*DIAM)     <= THICK
```

The above double constraint might prevent the Solver from making adequate progress towards a solution. Instead, apply the constraint to only one of the involved variables.

Fixed limits (numeric constants or expressions that do not contain registers nor processor variables) will *never* be violated during the course of the solution. Therefore, the value of DIAM in the above case will never drop below 0.01. However, formula-based limits (e.g., limits defined by expressions containing registers or processor variables) are treated as variable constraints that may be violated slightly during the course of the Solver execution, even if the values of those expressions *happen* to never change. Thus, the value of THICK may drop below 0.001 if the lower limit is defined as "MAX(0.001,0.1*DIAM)." Otherwise, the Solver could get stuck and not consider any variations below the *current* value of 0.1*DIAM (vs. the next *potential* value of 0.1*DIAM, since DIAM is a register and therefore might change during the optimization). The Solver control constant RCVIO regulates the tolerance of the model to the violation of these formula-based design variable limits. Small values of RCVIO will prevent excessive violation, but at a cost of slower solution speed and potentially stalled solutions.

Caution: Although the limits on design variables can be defined by formulas and therefore are subject to change during the course of the solution, the user is discouraged from making drastic changes to these limits, and *the user should never change the limits such that the current solution (design vector) is rendered invalid*.

Consider the case where the wall thickness is *identically* equal to 10% of the inner diameter at all times. If *both* DIAM and THICK are listed as design variables, then simply defining THICK = 0.1*DIAM in REGISTER DATA would not work since this initial expression is soon overwritten by the Solver because THICK was designated as a design variable. It is theoretically possible to constrain the value of THICK by a command such as:

```
C NOT RECOMMENDED IF "DIAM" IS A DESIGN VARIABLE
      0.1*DIAM <= THICK <= 0.1*DIAM
```

However, *the above usage is not recommended* not because it is illegal but because it is inefficient. Since THICK and DIAM are always related to each other, one or the other should be eliminated as a design variable. Eliminating THICK from the DESIGN DATA section leaves an expression of THICK = 0.1*DIAM intact in REGISTER DATA, making this the recommended way of handling the above problem.

More generalized constraints may also be imposed via HEADER CONSTRAINT DATA, defined in Section 5.6.

Caution: Since expressions may be used as both upper and lower limits, and such expressions might contain a single register name, ambiguity can result if only one such limit is defined:

```
dino <= bedrock
```

The code will issue an error in such a case.* The intention can be clarified by adding a “1*” or “+0” to the limit, or by placing it inside of parentheses. In the following example, “dino” is a register and “bedrock” is a limit:

```
dino <= bedrock*1
```

while in the following example, “dino” is a limit and “bedrock” is a variable:

```
(dino) <= bedrock
```

The DESTAB routine (Section 7.13.1) provides a convenient method for reporting either the current status or final values of the design variables. Other routines documented in that section are useful for changing the limits of a design variable or temporarily holding constant the value of a design variable.

* Because of its use of form-based input methods, this is not a problem in Sinaps.

5.6 HEADER CONSTRAINT DATA

Additional variables and expressions may be created to apply general constraints on the Solver. While many constraints can be expressed as expressions of registers and processor variables, more complex constraints can be defined using arbitrarily-named *constraint variables.*

If used, these constraint variables are real (noninteger) values that are intended to be updated along with OBJECT, perhaps during or near the end of the PROCEDURE.* The values of the constraint variables may therefore be arbitrarily complex, based on system parameters (temperatures, pressures, etc.) or derived data. Either an upper or a lower limit may be placed on these constraint variables, or both. These limits may be themselves defined by registers or expressions, including expressions containing design variables.

The basic format is similar (*but not identical!*) to DESIGN DATA as described in Section 5.5. There is also an option to generate several constraints with the same upper and lower limits, which is useful for "minimax" data correlations as described in Section 5.10. The format is:

```
HEADER CONSTRAINT DATA
       [ lower_limit <=] cst_name [<= upper_limit]
       [ lower_limit <=] expression [<= upper_limit]
       GEN basis,init,num,incr,lower_limit[,upper_limit]
```

where:

lower_limit. . . . optional lower limit or expression (*not* optional for GEN)

cst_name . . . . . unique name (8 characters or less) for a real constraint; *cannot be a register name (Section 2.8) nor a global user data name (Section 2.9) nor any reserved word (Section 6.18.5). In logic block references, all constraint variables are of type real, independent of their names.*

upper_limit. . . . optional upper limit or expression

expression . . . . an expression based on registers and/or processor variables, effectively replacing a constraint variable (and therefore sometimes called an "unnamed constraint"). Cannot be a register alone: use "(register)" or "1*register" to avoid an apparent conflict between a constraint name and a register name.

* Despite the apparent similarity with DESIGN DATA, constraint functions actually have more in common with the objective function (OBJECT): both are means by which the user tells the Solver how suitable is a particular set of design variables. In fact, OBJECT and the constraint variables are similar enough that confusion may result. For example, one might minimize the mass (OBJECT) of a heat transfer fin subject to the constraint (perhaps named "EFF") that the fin efficiency be no less than 90%. Or one might maximize the fin effectiveness (OBJECT) of the fin subject to the constraint (perhaps named MASS) than the fin weigh no more than 0.1kg. In both cases, OBJECT and the constraint variables must contain a measurement of the mass and fin effectiveness of the design currently being evaluated by the Solver after PROCEDURE has been called.

basis . . . . . 1 to 7 alphanumeric characters, basis of generated constraint names; cannot be a reserved SINDA/FLUINT word (e.g., T, Q, TL, COMP, etc.)
init. . . . . initial integer ID (integer 1 to 7 digits in length) for generated constraints; leading zeroes are significant and will be retained in the generated name
num. . . . . number of constraints to generate (positive integer)
incr . . . . . increment for numeric ID (cannot result in negative IDs)

Only one constraint variable and its limits, or one constraining expression, or one GEN command may be entered per line. *At least one limit must be defined*, and these definitions may include expressions containing registers or processor variables. However, if a named constraint is used, then the selected name must be unique. If both limits are defined, the upper and lower limit may be equal, but the lower limit cannot exceed the upper limit at any time.

When generating constraints with the GEN command, the names of the generated constraints will all begin with the characters defined as the basis, and will end with the number of digits as dictated by the *init*, *num*, and *incr* arguments. Leading zeroes in *init* are significant. For example (see below), if basis=CST, init=00, num=13, and inc=1, then the following named constraints would be generated: CST00, CST01, CST02, CST03, CST04, CST05, CST06, CST07, CST08, CST09, CST10, CST11, CST12. If on the other hand, init=0 (instead of init=00), the generated sequence would have been: CST0, CST1, CST2, CST3, CST4, CST5, CST6, CST7, CST8, CST9, CST10, CST11, CST12.

The number of characters in the basis plus the number of digits in the numeric ID (i.e., the length of the generated constraint names) cannot exceed 8 characters.

Example:

```
HEADER CONSTRAINT DATA
       0.01 <= deltaP
       -20.0 <= 0.5*(rod.T15+rod.T20) <= 40.0
       0.0 <= mass <= 10.0
       side1 <= 0.5*thick
       -pi <= angle <= pi
       flow.fr20 <= flow.fr30
       GEN CST,00,13,1,-errmax,errmax
C THE ABOVE* GENERATES:
C      -errmax <= CST00 <= errmax
C      -errmax <= CST01 <= errmax
C      -errmax <= CST02 <= errmax
C      ...
C      -errmax <= CST12 <= errmax
       GEN MORE,1,13,1,-errmax,errmax
C THE ABOVE GENERATES:
C      -errmax <= MORE1 <= errmax
C      -errmax <= MORE2 <= errmax
C      -errmax <= MORE3 <= errmax
C      ...
C      -errmax <= MORE13 <= errmax
```

For example of how to use a constraint variable, consider a system in which the pressure drop were not allowed to exceed 10,000 Pa. To accommodate this constraint, a constraint variable named "deltaP" could be created in CONSTRAINT DATA:

```
       deltaP <= 10000.0
```

At the end of HEADER PROCEDURE, a single calculation might be added to update the *current* value of deltaP, where lumps 10 and 23 define the pressure drop to be constrained:

```
       deltaP = pl10 - pl23
```

Alternatively, the above constraint can be imposed directly in CONSTRAINT DATA with a single statement that avoids the creation of a constraint variable that must be updated later. Any of the following impose the above constraint equally well (assume the lumps are in submodel† "water"):

```
       water.pl10 - water.pl23 <= 10000
       water.pl10 <= water.pl23 + 10000     $ equivalent
```

* If this option is used to automate "minimax" correlations and the COMPARE routine (Section 5.10 and Section 7) is used, the user must be able to predict the resulting naming scheme (or use a preliminary run with a call to CSTTAB) in order to pass the appropriate last argument to COMPARE.

† There is no equivalent of DEFMOD in CONSTRAINT DATA: submodel names must be used explicitly.

Like the calculation of OBJECT, the calculation of the constraint variable may be complex, perhaps performed cumulatively during the course of the PROCEDURE solution. For example, if a certain temperatures in the system should never drop below 100 degrees during the course of a transient, this could be achieved by defining:

```
100 <= Tmin
```

in CONSTRAINT DATA, setting Tmin to be huge (say, 1.0E30) at the start of PROCEDURE, and then adding the following line* in the appropriate VARIABLES 2 blocks for whichever nodes are of interest:

```
Tmin    = min(Tmin,T101,T304,T504)
```

In the current version, constraint variables are all real, and only continuous values are supported.

Upper and lower limits on constraints may be set equal, which means that a specific value is desired for that constraint variable or expression. Indeed, any named constraint can be made to be a temporary "equality constraint" via a call to the auxiliary routine HLDCST (Section 5.7).

Note that if named constraints are used instead of expressions, then even if the *desired* values are held constant, the *current* values must still be updated in logic during or at the end of PROCEDURE such that the Solver can measure its progress (i.e., by how much the constraint is violated or how far the design is from being limited by any particular constraint). In other words, the Solver must have a recent value of the constraint variable in order to compare it with the desired value (as dictated by the constraint variable limits). ***Constraints do not prevent invalid designs from being investigated during the course of the optimization, they merely prevent the final design solution from being invalid.***

Caution: During the course of performing the solution, the constraints imposed in HEADER CONSTRAINT DATA (unlike the fixed limits imposed in HEADER DESIGN DATA) will occasionally be violated: a constraint imposes a limit on the final design (and even then with slight tolerancing, per RCERRO), but does not guarantee the model will never be exposed to conditions that violate the constraint during the course of the solution. It is the user's responsibility to either assure that the model can survive the range of possible variations, or to restrict those variations inside of HEADER DESIGN DATA, or to employ parameters such as RCHGO (Section 5.4).

Guidance: The is no equivalent of DEFMOD in CONSTRAINT DATA. Submodel prefixes must be used explicitly.

Guidance: Some constraints on variations in design variables are cumbersome to apply in DESIGN DATA. For example, if X1, X2, and X3 are all design variables, but the sum should not exceed 3, then one could state in DESIGN DATA (in the definition of X1):

```
HEADER DESIGN DATA
       X1 <= 3-X2-X3
```

* To avoid discontinuities, if would actually be better to assign a separate constraint to each node.

However, it may be more convenient to impose such limits in CONSTRAINT DATA:

```
HEADER CONSTRAINT DATA
C X1, X2, X3 are design variables
       X1+X2+X3 <= 3
```

A “bare” register (that is, a register that is not part of an expression such as “X1”), however, may cause apparent conflicts since the program cannot allow a constraint variable to be defined that conflicts with a register name. Thus, the following is illegal:

```
HEADER CONSTRAINT DATA
C ILLEGAL IF X1 IS A REGISTER
       X1 <= 1
```

To make sure the program can discern an expression from the declaration of a constraint name, the user must “assure” the program that an expression is being specified by multiplying by 1 or by enclosing the register name in parentheses:

```
HEADER CONSTRAINT DATA
       (X1) <= 1
       1*X1 <= 1
```

Constraints may be temporarily suspended via calls to OFFCST (Section 7.13.2), or frozen as equality constraints using HLDCST. The routine RELCST reverses both such actions.

The CSTTAB routine (Section 7.13.2) provides a convenient method for reporting the current status or final values of the constraint variables. Note that constraints are classified in that output routine on the basis of whether they were explicitly named or whether they simply consist of expressions containing registers and/or processor variables (in which case they are termed “unnamed” and at least part of one of the defining expressions is listed for reference).

The CSTNAMES routine (Section 7.13.2) may be used to reveal the internal machine-generated names for unnamed constraints in CSTTAB calls, such that these constraints can be manipulated by OFFCST, HLDCST, and RELCST.

5.7 Solver Support Subroutines

This section lists support subroutines that are useful with the Solver. These abbreviated descriptions are wholly redundant: complete descriptions may be found in Section 7.13.

Refer also to Section 5.10.4 for a similar brief overview of routines intended to facilitate use of the Solver for test data correlations.

5.7.1 Output Routines

REGTAB (no arguments)—Tabulates registers, whether they are integer or real, their current values, and the expressions defining them. Internally calls CALREG to refresh registers prior to tabulation.

DESTAB (no arguments)—Tabulates design variables and their limits, if any. Identifies which limits are active and perhaps violated. Otherwise the tabulation indicates whether they are fixed or formula-based and therefore whether the limits are treated as a "side constraint" or a real constraint. Also contains other information about the current Solver status.

CSTTAB (no arguments)—Tabulates constraint variables and expressions and their limits, if any. Identifies which limits are active and perhaps violated, otherwise it indicates whether they are fixed or formula-based (in either case, they are treated as real constraints). Also contains other information about the current Solver status.

5.7.2 Auxiliary Routines

HLDDES(dv_name)—Temporarily hold the design variable *dv_name* (in single quotes) constant to the *current value*. Upper and lower limits are *internally* set equal to the current value, and external (user) limits are ignored and left unchanged.* There is little computational savings in holding a design variable fixed, so if a variable is never changed during an entire run, it should not be included as a design variable (i.e., it should be commented out temporarily). Or, if different sets of variables are to be permuted at different times within one run, thought should be given to separating the runs for efficiency.

RELDES(dv_name)—Releases a hold on a design variable *dv_name* (in single quotes), restoring attention to current user limits: reverses the action of HLDDES.

CHGDES(dv_name, flag, value)—Changes the value of an upper or lower limit on the design variable *dv_name* (in single quotes) to be *value*. Use *flag* = 'hi' or 'high' to change the upper limit, or *flag* = 'lo' or 'low' to change the lower limit. Caution: If this limit was originally defined as a register-containing expression, this change will be overwritten later unless DEREGC is used to disconnect that update process.

* For an example usage, see Section 5.9.1 on dealing with integer and discrete variables.

HLDCST(cv_name)—Temporarily hold a constraint constant (i.e., make it a temporary equality constraint). External (user) upper and lower limits of *cv_name* (in single quotes) are set equal to the *current value*, and register-based formula for those limits are temporarily disconnected from the update process. This routine can be called multiple times if needed to reset the limits. Note: the value of the constraint variable should continue to be updated in logic such that a departure from the limits can be monitored by the Solver.

RELCST(cv_name)—Releases a hold or the inactivation of the constraint variable *cv_name* (in single quotes) reconnecting limits to the update process if they were defined as formula, *but RELCST does not restoring previous limits* unless they were defined using registers. Otherwise, the CHGCST routine must be used to change the limits after they have been released.

OFFCST(cv_name)—Temporarily disable a constraint (as if "commented out"). RELCST releases this action.

CHGCST(cv_name, flag, value)—Changes the value of an upper or lower limit on the named constraint variable *cv_name* (in single quotes) to be *value*. Use *flag* = 'hi' or 'high' to change the upper limit, or *flag* = 'lo' or 'low' to change the lower limit. Caution: If this limit was originally defined as a register-containing expression, this change will be overwritten later unless DEREGC is used to disconnect that update process.

5.7.3 Diagnostic Utility

SOLCHECK—SOLCHECK is a Solver diagnostic utility intended to help check both connectivity and sensitivity for an initial problem set-up. *Connectivity* means answering the question: "Do changes in design variables cause a noticeable change in the objective and in the constraint functions?" *Sensitivity* means answering the question: "Are these changes significant, not too sensitive, and are they in the right direction?" SOLCHECK (no arguments) should be called from OPERATIONS. It provides a "Solver-eye view:" a snapshot of how the problem initially appears to the Solver.

5.8 How The Solver Works

This subsection describes how the Solver goes about the business of varying the N dimensional design vector (the set of N design variables) as needed to achieve the goal requested by the user. Whether one-dimensional goal seeking or test data correlation is the objective, the internal workings of the Solver are the same: it is seeking to *quickly* find a set of design variables that has the least difference between GOAL and OBJECT and yet does not significantly violate any of the imposed constraints.

This section refers to parameters and routines defined earlier in this chapter. Especially relevant are Figure 5-2 and Table 5-2.

Despite the 3 choices of methods (as selected by METHO) for constrained problems and the 2 choices of methods for unconstrained problems, the underlying strategy of the Solver is similar for all methods. The Solver starts from the initial conditions and moves toward an optimal solution, paying attention to constraints it encounters along the way. This can be likened to climbing a hill (maximizing the objective function or, in the case of minimization, maximizing its negative) where fences (constraints) exist to limit movement (Figure 5-3). The optimum solution might exist at the top of the hill as long as the fences permit the climber to reach the summit. It is possible that a higher summit exists, but that the climber did not start near it and therefore achieves only a local solution. It is also possible that the climber gets trapped by local fences, unable to find the way around them, and therefore does not achieve the highest point possible. It is also possible that the climber gives up if it is in a flat or rough area where it is unable to make significant progress or where it is not clear which way to proceed.

There are two steps in each phase of the Solver. The first step is the determination of the *search direction.* A search direction is an arrow or vector in the N-dimensional design space. To find this direction, the Solver calculates the derivative of the objective function (OBJECT) and any active constraints by perturbing the design variables one by one. Thus, if there are N design variables, the Solver will initially make 1+N*(MDERO+1) calls to PROCEDURE to estimate these derivatives. To continue the hill-climbing analogy, the Solver finds the shape of the local terrain. During this phase, NSTATO will contain -I, where the negative sign indicates that the Solver taking derivatives and that the I^{th} design variable is being perturbed.

If no constraints are initially active (i.e., none are violated or within RCACTO of being violated), then the Solver proceeds in the direction of steepest ascent (see “Search #1” in Figure 5-3). If constraints *are* active, the search direction is modified to follow the fence (as in the case of “Search #2” in Figure 5-3), or at least to avoid excessive transgressions of the fence (depending on the method selected). If the initial design is infeasible (violated constraints), then the search direction focuses instead on moving rapidly to the nearest feasible region, as indicated by the gradients in the constraint functions.

Upon determining this search direction on the basis of the first step, the Solver begins a *one dimensional search* or a movement of all design variables. During this second step, NSTATO will contain 1, indicating the first such search. The Solver will initially move somewhat cautiously (as governed by the PUSHO and RCHGO/ACHGO factors), but may become more ambitious during

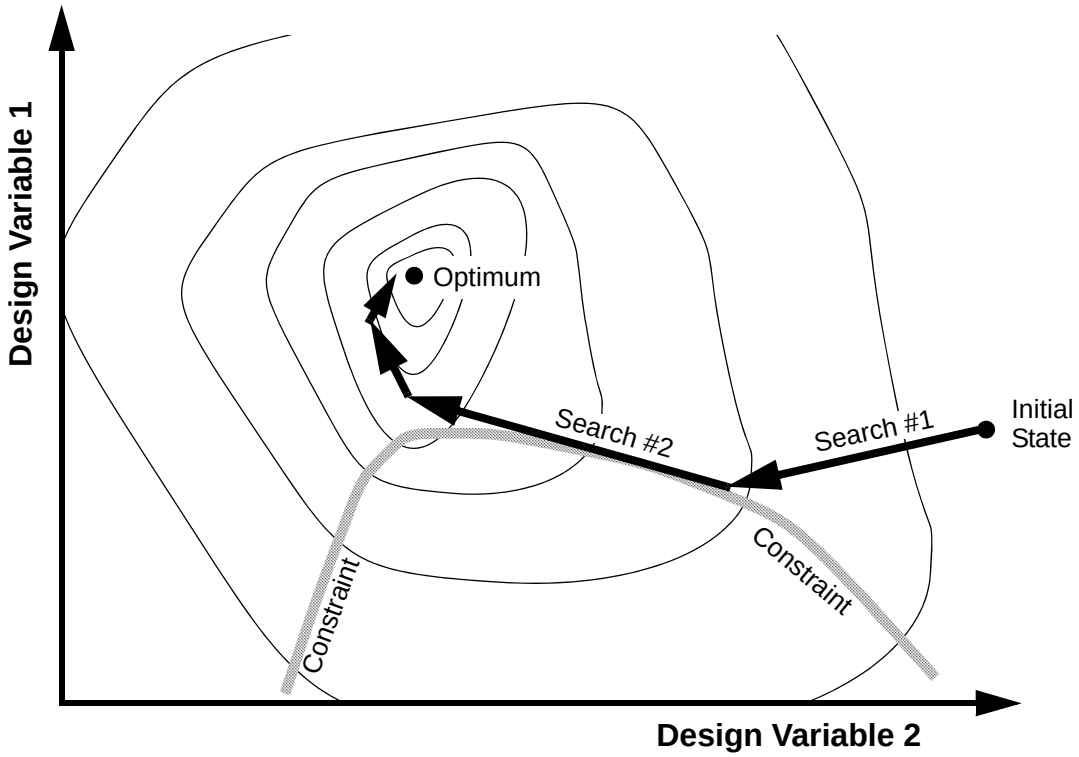


Figure 5-3 Graphical Example of Optimization in Two Dimensions

this search if conditions warrant. Usually about 5 (perhaps 3 to 10) calls to PROCEDURE are made while the Solver is performing this search. The selected methods (per METHO) will differ in how they perform this search.

A search terminates when either a new constraint becomes active (per the RCACTO criterion), or when a constraint is excessively violated (per the RCVIO criterion or per the RCERRO criterion), or when no more improvement is being made in the objective function OBJECT (an indication that the climber has perhaps reached a ridge). This illustrates the cost and consequences of the RCACTO criteria: too large a value might prematurely terminate a valid search direction (the climber is too cautious about nearby fences), yet too small a value might cause excessive transgression of a constraint (i.e., the climber moves through a fence before realizing it was there). This variation in the length of a search is shown graphically in Figure 5-4. Similarly, greater tolerance of a violation during the search (by larger values of RCVIO and RCERRO, indicating a more robust model and/or reduced accuracy requirements) are often rewarded by shorter run times (fewer PROCEDURE calls).

Once the first search direction terminates, the Solver goes back to the first step to find a new search direction by once again perturbing the design variables. The Solver often does not start a new search at the same point at which it ended the last search. Rather, it will back up to a previously determined best point.

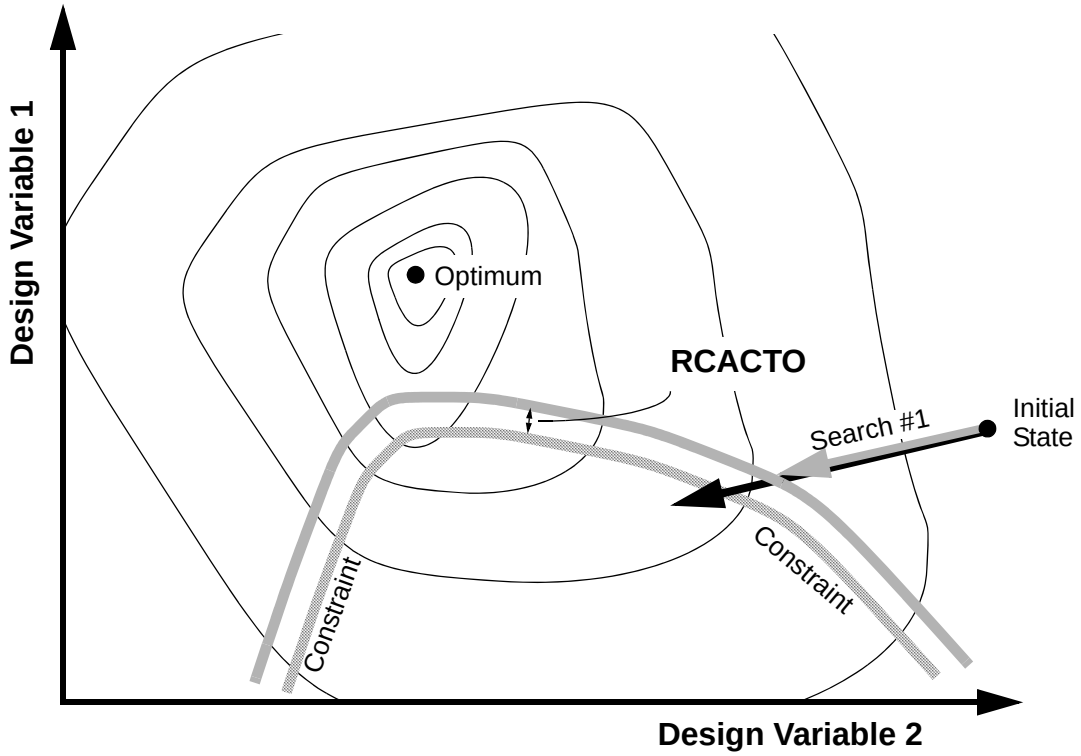


Figure 5-4 Graphical Example Showing Importance of RCACTO

At this point, SOLOUTPUT CALLS is called, but PROCEDURE has not been called so while the design variables and OBJECT are current, the constraints and the rest of the SINDA/FLUINT solution may not correspond to that state. To force them to be current (via the NEWPRO flag) requires an extra, otherwise unnecessary call to PROCEDURE. Also, DELOBJ is updated at the start of a new search direction to reflect progress made since the previous search terminated.

Since it has a known starting point, N*(MDERO+1) calls to PROCEDURE are required (i.e., one fewer than the initial step), and once again NSTATO contains -I as the I^{th} design variable is perturbed.

The second search direction is unlikely to be based on the direction of steepest ascent, even if no constraints are active, since that approach is surprisingly inefficient. Many of the variations in the selected methods occur in the selection of subsequent search directions. Otherwise, the second one dimensional search proceeds as did the first, with NSTATO now containing 2 to indicate the second search. The current status of the Solver is also presented in the header information of DESTAB and CSTTAB, along with the status of the objective function and constraint functions.

The above two steps (find a search direction and move along it) are repeated until either the objective function and/or constraints are no longer responsive to changes in the design variables, or until NLOOPO is reached, or until an optimal solution is detected. RERRO and AERRO are used to discern when changes to OBJECT (since the last search direction) are negligible, and NCONVO is used to determine how many such final search directions are required to permit convergence.

5.9 Extended Solver Usage

5.9.1 Dealing With Discrete Design Variables

Future versions are expected to feature prepackaged options for handling discrete variables. Until then, the user must treat discrete variables (e.g., available line diameters, integer number of components, material selection problems, etc.) as real variables and use the closest discrete choice to the resulting number.

While the above method might be adequate, some refinements are possible. After a successful solution (call to SOLVER), the values of all discrete variables can be changed to their nearest discrete value, fixed using HLDDES, and then a second call to SOLVER can be made to let other continuous variables adjust to the user-imposed movement of the design vector. Picking and freezing one discrete variable at a time, with N+1 calls to Solver where N is the number of discrete variables, yields even better results although at a cost of more calls to PROCEDURE.

As an example, assume that PARALLEL was a design variable describing the number of parallel (manifolded) fluid lines, and must therefore represent an integer value. The following logic might then apply in OPERATIONS:

```
        CALL SOLVER
C ROUND TO THE NEAREST INTEGER:
        PARALLEL       = FLOAT(NINT(PARALLEL))
        CALL HLDDES('PARALLEL')
C LET OTHER DESIGN VARIABLES ADJUST TO THE CHANGE
        CALL SOLVER
```

5.9.2 Handling Multiple Objectives

One of the most difficult parts of using the Solver is differentiating between objectives and constraints. For example, one might wish to maximize the efficiency of a fin while minimizing its mass. Normally, only one of these is an objective, and the other is a constraint. In other words, one should maximize the efficiency while constraining the mass to be less than a certain limit, or one should minimize the mass while constraining the efficiency to be greater than a certain limit. The user may wish to perform both analyses and see how different the results are.

However, it may be that the analyst is willing to sacrifice some efficiency if the weight savings are significant. In other words, there is no fixed lower limit on efficiency, but it may instead be a function of the total mass of the fin. This flexibility might be posed as a constraint to the code, or it might be posed as a composite objective combining the desire for *both* low weight and high efficiency.

To continue the example, the user might be willing to drop the efficiency by as much as 5% if the weight savings associated with that change is 10% or more of the baseline (constant) value "BASE." In other words, in this case the weight savings is twice as important as efficiency. This might be posed to the code as:

```
OBJECT = 2.0*WEIGHT/BASE - 1.0*EFFICIENCY
```

where "2.0/BASE" and "-1.0" are weighting factors to tell the program the relative importance of each subobjective. In the above case, the overall objective (OBJECT) is minimized.

Goal seeking can also be combined with one or more maximizations or minimizations. In the general case, where there are T targets and M maximizations or minimizations, the user might chose to minimize:

$$\text{OBJECT} = \left\{ \sum_{i=1}^{T} [F_i(O_i - G_i)]^2 \right\}^{1/2} + \sum_{j=1}^{M} F_j C_j O_j$$

where:

O_i . . . . . the i^{th} subobjective

G_i . . . . . the target or goal for the i^{th} goal seeking

F_i. . . . . the weighting factor for the i^{th} subobjective, taking into account the need to convert the units of each subobjective and to assign each objective a relative importance factor in order to be able to sum them linearly

C_i. . . . . a rectifying factor: $C_i = 1$ for a minimization and -1 for a maximization

The above suggestions are guidelines only: there is no "best approach" or theoretical basis for the above relationships, and some experimentation is required.

5.10 Test Data Correlation Methods

Although the Solver can be used to help the analyst design a system (e.g., change the design and therefore the model), it can also be used to help calibrate a model to test data (e.g., change the model to fit the data). A correlated model is effectively an intelligent extrapolator of known data points to untestable conditions.

This section contains tips and suggestions for using the solver to automate test data correlation.

5.10.1 Identifying Key Uncertainties

Perhaps the most difficult part of test data correlation is identifying which modeling parameters have the greatest uncertainties and are therefore in need of adjusting, and to what extent they should be allowed to vary (in other words, what are the limits on these uncertainties?). Although prime candidates in most thermal/fluid models are convection coefficients (especially natural convection), as-built insulation performance, contact conductances, flow loss coefficients, and surface optical properties, even dimensional variations and property uncertainties can be important, as can boundary conditions (e.g., input power, sink temperatures, etc.).

If a model has been built making extensive use of registers, then some of these registers can be used as “design variables” (perhaps better referred to as “correlation variables” in this instance). Other factors not originally defined using registers can be made variable perhaps by multiplying by a correlation factor whose initial value is unity. For example, if the electric power into a device is believed to be 100W but is uncertain by 5%, then the user might specify:

```
HEADER DESIGN DATA
       0.95 <= UNKPOW <= 1.05
```

and then apply the value of UNKPOW as a multiplier to the appropriate places, perhaps on a certain node in SOURCE DATA:

```
HEADER SOURCE DATA
       999, 100.0*UNKPOW
```

Similarly, uncertainties in film coefficients can be applied as factors to the DUPL and DUPN of a tie, uncertainties in flow losses can be applied to the FK term, etc.

It is important to identify appropriate uncertainties, and to avoid applying them gratuitously. Too many correlation parameters will result in some not having much effect on the results which will significantly slow the progress of the Solver.

More importantly, it should be realized that correlation may result in more than one seemingly valid answer. For example, assume that a piece of composite material is heated and that it conducts to a sink. Assuming the sink temperature is a known (not uncertain) value. In a traditional SINDA/FLUINT transient analysis, there is *one* solution: an exponential curve in time. However, to correlate to a single data point (say the final steady state temperature), *the reverse problem has multiple solutions*: a modification of either the power input *or* the conductance to the sink can result in the

same “correlated” model. The user might be able to discern better between these two uncertainties by correlating to the entire transient history and not just to the final results, but then uncertainties in size (mass) and perhaps even specific heat must then be taken into account. All of the foregoing discussion assumes that there were no *modeling uncertainties*: that the user did a perfect job of capturing the system response using an appropriate network and logic. In reality, the model itself usually contains many approximations and assumptions, especially when fluid flow is involved.

Therefore, the user cannot leave too much to the computer by making too many parameters uncertain and assuming that enough test data will overcome the problem. Rather, some intelligence must be applied to the selection of the key uncertainties, and to the process by which they will be correlated. For this reason, it is important to understand the modeling uncertainties and to have run parametric analyses (Section 4.9) to better understand how the system (and the model of that system) responds.

5.10.2 Varying the Uncertain Parameters

Once the uncertainties have been identified and their relative uncertainty assessed, the problem can be posed to the Solver. Traditional “manual” test data correlation often involves changing one uncertainty at a time. Improved correlations may be achieved by letting the Solver vary all relevant parameters simultaneously. Nonetheless, the Solver works more efficiently with fewer design variables.

For example, many correlations to thermal test data are performed in two steps. The first step is correlation of conductances to steady-state thermal balance testing based on measured temperatures and perhaps measured duty cycles of thermostatic heaters. Given an improved model, the second step is to correlate uncertainties in lumped capacitances based on transient test data. Although theoretically the Solver could be used to find the best values of all uncertainties given a single PROCEDURE that contained both steady state and transient analyses, this would be much less efficient than separating the correlation into two runs corresponding to the above steps. The first run would identify only conductances and similar uncertainties as design variables, and the PROCEDURE would contain only the relevant steady state comparison (or comparisons for multiple data sets). Saving the results, the second run would identify only capacitances and other relevant uncertainties as design variables, and the PROCEDURE would contain only transient analyses.

Multiple data sets may be available with which to correlate, and some data may be more important or trustworthy than others (or some correlation points more critical than others). Section 5.10.3.4 and Section 5.10.3.5 deal with these issues.

Finally, note that there are many methods available for determining a best fit to a given set of test data, as described next in Section 5.10.3.

5.10.3 Comparison Methods

The Solver ultimately requires a single-valued measurement of the appropriateness of any set of correlation parameters to a given set of test data. Assuming that this value is a measure of the relative error between test data and predictions, it becomes the OBJECT to be minimized. Ultimately, the user has complete control over how the OBJECT is calculated. This section provides guidelines for common correlation methods: methods by which a reasonable OBJECT can be computed.

Since correlation typically deals with a lot of data, data handling and comparison routines are provided to help with these tasks, as described in Section 5.10.4. Refer also to Section 5.14 for examples.

5.10.3.1 Least Squares Error

One of the easiest relationships to apply is a least-squares fit, meaning that the Solver will adjust the model until the predictions P give the least square fit to experimental data E over a total of T points of comparison:

$$\text{OBJECT} = \sum_{i=1}^{T} (P_i - E_i)^2$$

This is easily programmed by setting OBJECT to zero at the start of PROCEDURE, and then in the appropriate place adding (*perhaps*--the following is a programming example only):

```
      DO 1 I = 1, T
1           OBJECT  = OBJECT + (P(I) - E(I))**2
```

If the basis of the comparison (the contents of PROCEDURE) is a single steady state solution, then the “appropriate place” for such logic is at the end of PROCEDURE. This would also be the appropriate place to compare the final results of a transient analysis. However, to compare the results cumulatively--over the entire time history of a transient--then the appropriate place for the above logic is in an OUTPUT CALLS block or some other place of known calling frequency.*

From the perspective of the Solver, least squares fits usually involve fixed limits on design variables, but no constraints. (Recall that fixed limits on correlating parameters in DESIGN DATA are not considered to be true constraints, unlike register-based limits or real constraints applied in CONSTRAINT DATA.) Therefore, it is important to note that METHO often has a different meaning for least squares correlations than for other methods (such as the “minimax” method described later,

* A fake thermal submodel (with an empty NODE DATA block) can be created solely for the purpose of being able to independently specify its output interval. Such a method is also often used for modeling devices such as discrete control systems.

which has extensive constraints), and that changing constraint-related control parameters such as RCACTO, RCERRO, and RCVIO has little or no effect on most least squares fits. However, strong differences have been noted between performance of METHO=1 and METHO=2 for unconstrained least squares fits, so experimenting with both options is recommended, and verifying the final answers (see Section 5.11.3) is also strongly recommended.

To yield the root-mean-square (RMS) error, note that RMS = SQRT(OBJECT/T) where T is the number of comparison points. Since the RMS error is not as volatile as OBJECT itself in terms of changes in magnitude (see Section 5.11.2.1), and since its units are likely to be more intuitive, it is often more suitable to use the RMS error as a variable to be minimized.

The COMPARE routine (Section 5.10.4 and Section 7.13) may be used to assist in the above programming tasks, using the SUMSQR option to return the sum of squared errors, and the RMSERR option to return the RMS error.

5.10.3.2 Least Average and Cubic Errors

Instead of a least squares error, good results are sometimes obtained by minimizing the average error, which is equivalent to minimizing the sum of the absolute errors:

```
      DO 1 I = 1, T
1           OBJECT  = OBJECT + ABS(P(I) - E(I))
```

The above method does not apply as severe a penalty to large errors as does a least squares fit. Alternatively, to apply an even greater penalty on the larger errors than the least squares fit, a “least cubes” fit could be used (remembering the need to take the absolute value of the error):

```
      DO 1 I = 1, T
1           OBJECT  = OBJECT + ABS(P(I) - E(I))**3
```

In the above case, however, note that the magnitude of changes OBJECT can be quite large. Therefore, minimizing the cube of the above factor (e.g., a root mean cube) is often preferred. Alternatively, SOLVER can be called twice in a row.

The COMPARE routine (Section 5.10.4 and Section 7.13) may be used to assist in the above programming tasks, using the SUMABS option to return a sum of absolute values or error, or better still the AVGABS option for an average of absolute error. SUMCUBE option returns the sum of cubed absolute errors, while the preferred RMCERR option to returns the cube root of the mean of the SUMCUBE result.

5.10.3.3 Minimized Maximum Error (Minimax)

An alternative to the least squares method is to minimize the maximum error (maximum deviation from test data) instead of the average error. All predictions must fall within this error band, and the best fit corresponds to the smallest error band. This method tends to spread the error more uniformly over the simulation than does the least-square method, which despite its larger penalty for larger errors, tends to accumulate the error in discrete locations (which is beneficial from the standpoint that it better tolerates erroneous data points). In general, the “minimax” method results

in better correlations. ***However***, it often requires more calls to PROCEDURE, is much more difficult to set up and check than is a least squares method, is more sensitive to erroneous test data, and is more sensitive to control parameters such as RCACTO. Therefore, this method should only be used when the number of comparison points is small, and when the test data is clean. Otherwise, the user might get better results faster by using a "least cubes" fit as explained above.

It might seem at first that this minimax method can be easily achieved in a manner analogous to the least squares method. This will be shown below to be false. However, for comparison purposes that incorrect method is described first:

$$\text{OBJECT} = \text{MAX}\left\{ \sum_{i=1}^{T} |P_i - E_i| \right\}$$

This could be easily programmed by setting OBJECT to zero at the start of PROCEDURE, and then in the appropriate place adding:

```
      DO 1 I = 1, T
1           OBJECT = MAX(OBJECT, ABS(P(I) - E(I)))
```

This is the same as the MAXERR option returned by the COMPARE routine (Section 5.10.4 and Section 7.13).

Unfortunately, in the final answer more than one data point (and perhaps several) will hit this limit. This causes the function OBJECT to be discontinuous, which usually causes the Solver to converge to a suboptimal answer too quickly, or more rarely to oscillate between several solutions.*

There is a safer method to use for performing a minimax curve fit, but it requires extra work and often requires extra solution time over a least squares method. Hence, despite the fact that the two methods might come to very different conclusions, the thought should be given as to when the minimax method is needed, and the least squares method should probably be attempted first for comparison.

To perform a minimax fit, the following steps should be taken:

* Having made appropriate warnings about the mathematical dangers of this simplest "MAXERR" approach, it must be noted that in several test cases this approach worked rather well, and is certainly easier to apply than the "true" minimax method described in this subsection. Therefore, taking precautions to watch for stalled or nonconverging solutions, it is recommended that the user experiment with the MAXERR approach.

1. Add a new register (named “ERRMAX” in this example) in REGISTER DATA, and initialize it to a relatively large value -- approximately the largest initial value of MAX(SUM(ABS(P_i-E_i))). A large positive guess is adequate, but a more efficient value might be available from an earlier run with a little foresight.

2. Make the new register a design variable in DESIGN DATA, but do not place any limits on its value (other than perhaps 0.0 > ERRMAX).

3. In SOLVER DATA, set OBJECT = ERRMAX and minimize (the default). OBJECT, being defined on the basis of a register, need not be updated elsewhere in logic.

4. For each comparison point (at each time in a comparison versus time in a transient), create a new constraint variable in CONSTRAINT DATA and set the limits to be +/- ERRMAX.

   For example:

```
HEADER CONSTRAINT DATA
       -ERRMAX  <=     COM001   <= ERRMAX
       -ERRMAX  <=     COM002   <= ERRMAX
       -ERRMAX  <=     COM003   <= ERRMAX
       ...
       -ERRMAX  <=     COM100   <= ERRMAX
```

   The above can be generated using the GEN command:

```
GEN COM,001,100,001, -ERRMAX, ERRMAX
```

5. In the appropriate place in logic, update the values of the constraint variables (*perhaps-the following is a programming example only*) ***without*** absolute values:

```
COM001  = P(1) - E(1)
COM002  = P(2) - E(2)
COM003  = P(3) - E(3)
...
```

   The above can be automated using the MINIMAX option of the COMPARE routine (Section 5.10.4 and Section 7).

As guidance, note that minimax fits are heavily constrained analyses, and therefore their execution can be very sensitive to the values of constraint-related control parameters, especially RCACTO (Section 5.4). If a run is taking too many procedure calls, try higher and lower values of RCACTO. Also strong differences have been noted between performance of METHO=1 and METHO=2 (or 3) for minimax correlations, so experimenting with both options is recommended, and verifying the final answers (see Section 5.11.3) is also strongly recommended. Even reversing the sign of the constraint variables:

```
COM001  = E(1) - P(1)
COM002  = E(2) - P(2)
COM003  = E(3) - P(3)
...
```

(or by reversing the first two arguments in the COMPARE routine) can yield different solutions.

Spurious or untrustworthy test data can hamper any correlation. However, it is particularly harmful for minimax correlations since these are typically dominated by a few data points, whereas a weighting factor is applied to all comparisons in a more traditional least squares fit.

One of the difficulties in setting up a minimax problem is updating all of the constraints. While the COMPARE routine is very helpful in such instances, sometimes the user must employ advanced techniques. Therefore, it should be remembered that constraint variables are stored internally in the input order. Thus, a user can set a Fortran EQUIVALENCE of a user array (not a SINDA array) to one cell and loop through adjacent cells, or can pass one cell to an internal routine that treats the constraints as an array. Both of these techniques require not only a little programming skill, but also a lot of caution and care during updates of the model.

5.10.3.4 Comparisons at Multiple Times or Multiple Cases

Often, one need only compare model predictions with test data at the end of a steady-state or transient run. However, if the user needs to compare the predictions with the test data over the course of a transient event, or if a single correlation is to be performed on the basis of multiple tests (and therefore multiple comparisons), then the objective and/or the constraints must be cumulative (summed into during the execution of PROCEDURE).

In the least squares method, the value of OBJECT must be zeroed at the start and then summed into over the course of the transient. Often, the points of comparison will be at preset and constant intervals. If no suitable place already exists for such logic, one can be easily generated by creating a fake thermal submodel (with an empty NODE DATA block), adding it to the BUILD command, setting its OUTPUT value to the desired frequency of comparisons, and then using the OUTPUT CALLS for that fake submodel to store the comparison logic.

In the minimax method, the programming is similar except that the constraint variables are updated rather than the OBJECT, and since each point of comparison must have its own constraint variable, one such comparison and variable must exist for each time point compared. Although the COMPARE routine is available to assist in this task, it can still require quite a bit of logic. For example, if there are 4 time histories (say, the temperature of nodes 1 through 4) to compare over the course of 1 hour at 5 minute intervals (12 points, neglecting the starting conditions), then the user must create and update 4*12= 48 constraint variables (perhaps using the GEN command). Furthermore, the logic that updates them must be able to choose which variable to update. In the above example, if COMXYY compared the X^{th} data stream at time YY, the following might be used in an appropriate OUTPUT CALLS block (where time is in units of hours, A1 contains a bivariate array of test results versus time for node 1, etc.):

```
CALL D1DEG1(TIMEN,A1,ATEST)
CALL D1DEG1(TIMEN,A2,BTEST)
CALL D1DEG1(TIMEN,A3,CTEST)
CALL D1DEG1(TIMEN,A4,DTEST)
IF(TIMEN .GT. 59./60.0)THEN
       COM160  = T1 - ATEST
       COM260  = T2 - BTEST
       COM360  = T3 - CTEST
       COM460  = T4 - DTEST
ELSEIF(TIMEN .GT. 54./60.0)THEN
       COM155         = T1 - ATEST
       COM255         = T2 - BTEST
       COM355         = T3 - CTEST
       COM455         = T4 - DTEST

ELSEIF(TIMEN .GT. 49./60.0)THEN
       COM150         = T1 - ATEST
       COM250         = T2 - BTEST
       ...
```

With a little preparation (as explained in Section 5.10.4), this programming can be reduced using prepackaged auxiliary routines:

```
CALL PREPLIST('smn','T',A5,0,A10)
CALL PREPDAT1(TIMEN,'smn',A5,0,0,A20)
IF(TIMEN .GT. 59./60.0)THEN
       CALL COMPARE(A10,A20,0,0,'MINIMAX',COM160)
ELSEIF(TIMEN .GT. 54./60.0)THEN
       CALL COMPARE(A10,A20,0,'MINIMAX',COM155)
ELSEIF(TIMEN .GT. 49./60.0)THEN
       CALL COMPARE(A10,A20,0,'MINIMAX',COM150)
       ....
```

Another difficulty with correlating transient runs, whether the comparison is accumulated or whether only the last value (at TIMEND) is used, is that the initial conditions must be reset in PROCEDURE since all operations in SINDA/FLUINT operate cumulatively upon the data set. Normally, a call to SAVPAR could be made in OPERATIONS before the call to SOLVER, and then a call to RESPAR could be made at the start of PROCEDURE. SVPART might be used instead of SAVPAR to make more selective restores, but note that in either case the user should separately reset the values of TIMEO and TIMEN.

Restoring routines (e.g., RESTAR or RESPAR) automatically detect that they have been called from within the Solver, and so refresh design variables after reading in data so that the design variables are not overwritten by the restore command. Furthermore, when data is restored via the

RESTAR or RESPAR call, the updates invoked by UPREG (as called in the Solver) would be overwritten (Section 4.9.7) without further action by the program. For example, if the temperature of a boundary node is determined by a register-containing formula and those registers are either design variables or a functions of design variables, then this temperature will be reset by a call to RESPAR such that the effects of the formula would be lost. To avoid this problem, the code automatically propagates updates for any expression containing (directly or indirectly) the design variables after restoring the data.

Finally, note that the repeated transient runs that will be performed by the Solver (perhaps 30 to 300 of them) can be quite costly to execute, and that every effort should be made to reduce the model, increase the time step, and/or streamline the integration in order to make multiple transients analyses affordable. Correlated parameters from a reduced model, for example, can be applied to a more detailed original model.

5.10.3.5 Uneven Weightings

The previous examples use equal weighting of all comparison points at all points. This need not be the case: some data points may be more critical than others, and some time points might be more critical than others, or some test cases may be more valued than others.

As an example, correct answers at the end of a transient run might be more critical than those that are predicting during the transient, since errors tend to accumulate in a transient. Each comparison point might be weighted, for example, by the problem time (e.g., TIMEN/TIMEND) or some other factor. Thus, it can be seen that using only the final results to correlate a transient run represents the ultimate in such a weighting strategy.

As another example, some parts of the test data may be more important than are other parts. For example, the temperatures of critical components should be given more weight in the correlation than should the temperatures of support structure or other uncontrolled components.

To apply uneven weightings to the average absolute error, RMS (least squares), or RMC (lease cubes) methods, simply apply a multiplying factor to the relevant $(P_i\text{-}E_i)$ term in the above equations.

To apply uneven weightings to the minimax method, apply a multiplying term to the calculation of the relevant constraint, which again corresponds to adding a multiplying factor to the relevant $(P_i\text{-}E_i)$ term.

5.10.4 Test Data Correlation Support Routines

This section provides a brief overview of routines intended to help deal with the large data requirements typical of test data comparisons. This section is wholly redundant; complete details are available for each of the support routines in Section 7.

These routines help prepare and compare large sets of predictions and test data, in many cases using SINDA/FLUINT arrays (Section 2.11) as the basis for storage. For this purpose, recall that space can be allocated in a SINDA/FLUINT array using the SPACE command.

COMPARE—Given that a set of test data has been stored in one SINDA/FLUINT singlet array and a set of predictions has been stored in another singlet array, COMPARE can be used to produce a comparison or to tabulate and report statistics on the two data sets, which is strongly recommended as part of model debugging. When used to reduce comparison tasks, COMPARE can return single-valued statistics such as RMS error or maximum error for use in the OBJECT variable, or it can be used to store comparison points in constraint variables as needed to support minimax fits.

PREPLIST—This routine prepares a singlet array containing SINDA/FLUINT predictions, perhaps in preparation for a subsequent call to COMPARE. The user provides a list of node, lump, or path identifiers and the name of the parameter (e.g., ‘T’ or ‘PL’) to return.

PREPDAT1—This routine prepares a singlet array containing test data, perhaps in preparation for a subsequent call to COMPARE. PREPDAT1 assumes the data is stored as doublet arrays (time vs. data) or as pairs of singlet arrays (one time array, one or more data arrays of equal length).

PREPDAT2—This routine helps prepare a singlet array containing test data, perhaps in preparation for a subsequent call to COMPARE. PREPDAT2 assumes the data is stored in one large table, such as might be imported from an external spreadsheet.

5.11 Potential Solver Problem Areas

The design philosophy of the Solver recognizes (1) the typically large costs involved in each call to PROCEDURE, and (2) the low accuracy levels typically needed by most engineering applications (e.g., 1% not 0.01%), with almost any improvement in a design or a correlation welcome. It therefore attempts to arrive at a reasonable answer with as few calls to PROCEDURE as possible. The user may need to adjust decisions regarding what is a "reasonable" answer, and may need to consider the problem carefully to make sure it has been posed well.

A well posed problem has as few design variables as possible, with OBJECT and constraints not being noisy or weak functions of those design variables (such that trends are not hard to find and are not lost in the noise). The initial values of OBJECT and the design variables should not be orders of magnitude away from the final values. The underlying SINDA/FLUINT solution should be honed for its purpose, and should accept wide variations in the model as a function of the designated design variables. Adequate accuracy must be demanded from each PROCEDURE solution (and the underlying steady state and/or transient solutions) such that the noise level is small enough for trends to be clearly distinguished.* OBJECT and the constraints should not be discontinuous (not differentiable), especially near a solution.

5.11.1 Large Run Times

To perform its work, the Solver makes repeated calls to PROCEDURE: typically about 30 to 300. Since PROCEDURE is often equivalent to a single call to SINDA/FLUINT (in versions prior to the introduction of the Solver), considerable solution time can accrue as a result. Sometimes, as in the case of repeated steady state runs, there is significant economy since the iterative solutions start from a known and reasonable point.† Still, ***it is strongly recommended that the user expend effort to make sure that a single call to PROCEDURE performs the desired operations correctly and quickly before invoking the Solver***.

Therefore, the user should consider setting up a preliminary run in which NLOOPO=1 (or NLOOPO=2 in the case of PROCEDURE blocks containing transient analyses) to make sure that the Solver is set up correctly and that the desired procedure is being followed successfully. The SOLCHECK utility (Section 7.13.4) is especially helpful for verifying a good initial set-up.

The user should also set the value of NERVUS to as high as is tolerable in order to avoid aborting a run. The default value is safe, but too often causes runs to abort since the "ENERGY STABLE BUT NOT BALANCED" (lack of progress) termination in SINDA steady state solutions is often reached. However, the user must be careful not to accept too much energy imbalance in such solutions, since this results in numerical noise that can mask trends or confuse the Solver.

* If such "noise" is unavoidable, raise RDERO and perhaps set MDERO=1 as well.
† DSCANLH (Section 5.15.2) is helpful for determining a good starting point.

SOLOUTPUT CALLS is intended as a reasonable place to perform calls to CRASH, SAVE, and other operations that allow results to be restored in future runs. However, often just printing out the current set of design variables (using perhaps DESTAB) allows the user to manually reset their initial values in future runs providing the number of design variables is not excessive.

The performance of the Solver can be sensitive to the values of the control parameters, in particular METHO and RERRO but also RCHGO (if it has been reduced) and RCACTO for heavily constrained problems. As with other solution routines, some experimentation may be required to find a set of parameters and methods that works well with any particular problem.

5.11.2 Nonresponsive Solutions: Apparent Stalling

The Solver can sometimes stop and declare completion with little or no movement in the one or all design variables. To the Solver, “I give up” has the same meaning as “I am finished because I am making no more progress:” it cannot distinguish between the two cases and therefore issues no warnings or error messages in such cases.

If only one or two design variables remain constant, the user should investigate the sensitivity of OBJECT and the user-imposed constraints to changes in those design variables. The SOLCHECK utility (Section 7.13.4) is specifically designed for this diagnostic activity, and the routine DVSWEEP (Section 5.15.1) is also helpful for such debugging activities. If almost all of the variables remains unchanged (as indicated by zeroes or very small values in the SOLCHECK output), then it is likely that either OBJECT or one or more constraints is weakly dependent on the design variables, noisy, or discontinuous. These causes and their resolution are described below.

Other causes of lack of progress include poor initial values of design variables, large changes in design variables and/or the objective, or problems with mixed power scaling (extreme nonlinearity).

5.11.2.1 Scaling Problems: Large Changes in OBJECT and Design Variables

Because the Solver deals with a wide variety of problems of difference scales, it internally rescales and normalizes variables, especially at the start of its operation. This fact, combined with its use of control parameters that depend on the *initial* values of inputs, makes the Solver sensitive to large changes (greater than about two orders of magnitude) in OBJECT and any of the design variables over their initial values.

Because of this sensitivity, ***starting with initial design variable values of zero is illegal. (***Note that “the initial value of OBJECT” refers to its value *after* the first call to PROCEDURE, and not to any initial value assigned that variable in SOLVER DATA or before SOLVER is called.)

If such changes are experienced, the program will issue a caution and the user may need to either refine the initial values or consider calling SOLVER twice or more in a row:

```
CALL SOLVER
CALL SOLVER
```

This usage lets the Solver rescale itself in the second call using whatever progress it was able to make in the first call. A change of methods may also be warranted between the calls, as noted in Section 5.11.3.

A more efficient method (especially useful for worst-case searching and calibration to test data) is to prescan the design space with a call to DSCANLH (Section 5.15.2) then start the Solver with the results of that scan. For example:

```
NLOOPO = 20            $ quick scan
CALL DSCANLH
NLOOPO = 200           $ back to full # of PROC calls
CALL SOLVER
```

The call to DSCANLH usually pays for itself in reduced procedure evaluations in the subsequent Solver call, but requires that reasonable upper and lower values of design variables be specified.

Finally, a change in scale may be imposed by the user. For example, the user might choose to minimize the *square root* of squared error instead of just the squared error. While they are mathematically identical in theory, in practice the former will have smaller variations in magnitude than the latter. Similarly, the user might use exponentiations and logarithms to raise or lower the power of a design variable.

For example, if a design is especially sensitive to a single variable such as an orifice diameter and the Solver is therefore neglecting changes to other variables by comparison, then the user could choose to use the logarithm of the orifice diameter as a design variable instead of the diameter itself:

```
HEADER REGISTER DATA
       Dorifice = ln(expD)
       expD    = exp(0.001)
HEADER DESIGN DATA
       expD
```

The above logic “desensitizes” the model to percent changes in this design variable, in effect lowering its power. Squares and square roots can be used as well to raise or lower the power of design variables:

```
HEADER REGISTER DATA
       Dorifice = sqrt(Dsqr)
       Dsqr    = 0.001^2
HEADER DESIGN DATA
       Dsqr
```

Raising or lowering the power of the OBJECT (objective value) is similarly possible. Ideally, all important variables would be at equal strengths.

5.11.2.2 Weak Functions

Just as some design variables have too strong an influence on a design, others can have too weak an influence. Either a change in variables needs to be made, perhaps by rescaling, or the more likely weaker variables should be dropped as design variables since they had no influence on the design. The SOLCHECK utility (Section 7.13.4) is specifically designed for this diagnostic activity.

To raise the power of a variable to make changes to it more noticeable, one could employ an analogous but opposite strategy to the one applied in the previous subsection. The following makes the variable “outerD” have a stronger response:

```
HEADER REGISTER DATA
        outerD      = sqrtD^2
        sqrtD       = sqrt(0.75)
HEADER DESIGN DATA
        sqrtD
```

5.11.2.3 Noisy Functions: Accuracy of the Underlying Procedure

Noise (low accuracy, hysteresis, jitter, etc.) in the underlying SINDA/FLUINT solutions can also prevent the Solver from making intelligent decisions because the trends in the data are masked by the noise in the solution. **Experience has shown this *computational noise* to be one of the most common causes of stalled or incomplete Solver solutions.**

One cause of computational noise is premature convergence or nonconvergence in steady states, which can be caused by being too liberal with the value of NERVUS and accepting incomplete answers from the network solutions. In such cases, if tighter SINDA convergence criteria (e.g., EBALSA, DRLXCA, RERRF) and matrix methods (nonzero MATMET) don’t work, then a modeling change may be required to avoid stalled convergence. As an alternative, consider raising the value of RDERRO (and perhaps MDERO) to force larger perturbations to design variables, in effect making the real trends in the model stand out against a noisy background.

If RadCAD® Monte Carlo solutions are used, they can also be a source of computational noise. Make sure that enough rays have been shot per surface, and consider using the same random seed to start each solution in order to improve comparisons.

To watch for weak or noisy responses, the user can monitor the results of the first calls few calls made to PROCEDURE: the first few Solver iterations. When the Solver starts, the first N calls to PROCEDURE correspond to perturbations in each of the N design variables in input order, so the output can be scrutinized for significant variations, perhaps using DESTAB and CSTTAB inside of SOLOGIC 0 to watch the changes each iteration. Either NSTATO or the header information in DESTAB and CSTTAB can be used to determine which variable is currently being perturbed.

5.11.2.4 Discontinuous Functions

Discontinuous objectives or constraints can present false or conflicting trends to the Solver.

For example, perhaps the user wanted a goal-seeking solution where OBJECT was a lump quality, and a desired quality (GOAL) of 0.1 were imposed. While this might seem to be a perfectly valid approach, the fact that lump quality is limited to the range of (0.0, 1.0) can cause the Solver to sense lack of progress and terminate prematurely. In other words, such a problem could only be solved if the quality of the lump was never 0.0 and never 1.0 during the range of designs investigated by the Solver, such that the quality always responded to changes in design variables. A constraint on such a quality would also cause the Solver to either oscillate or terminate prematurely for similar reasons: the quality may be continuous, but it is not differentiable or always responsive to changes. The user must find other variables or combinations of variables that are responsive and smoothly differentiable. For example, if in the above case the user were trying to specify the quality of a single lump within a boiler or condenser, they might find more success specifying the average quality of all lumps in the component. Or, the user might choose a continuous yet equivalent thermodynamic quantity such as enthalpy instead of quality.

A common problem is to add a discontinuous constraint. For example, assume that the user wanted to prevent an entire component from dropping below -40 degrees at any point within it, and that the component was modeled using many nodes. At first, this might seem like one constraint that is easily posed to the problem as:

```
HEADER CONSTRAINT DATA
       -40.0 <= TMIN
HEADER PROCEDURE
       ....
       TMIN    = MIN(T31, T45, T66, ...)
```

However, this is a discontinuous constraint since two or more temperatures can be limiting the design at the same time. While the above may be attempted and *may* perform satisfactorily, a more consistent and safe approach is to assign each limited temperature to its own constraint. The GEN option in CONSTRAINT DATA is intended to automate such constraints, and the COMPARE (MINIMAX option) and PREPLIST auxiliary routines can be helpful in updating lists of constraint variables.

Other examples of discontinuous problems include minimizing an integer or discrete value, such as number of components (e.g., minimizing the number of parallel fluid legs), or constraining an integer/discrete value (e.g., limiting the number of heater cycles). In the former case, the objective variable needs to be a real (continuous one) with the nearest appropriate value used as a result after the optimization is complete. In the latter case, a different constraint needs to be applied to a continuous variable, or at least one that can be assumed to be continuous.

5.11.3 Double Checking an Answer

As was mentioned above, the intent of the Solver is to find an improved solution in as few calls to PROCEDURE as possible. It therefore may not have found a unique solution, nor the best possible solution. Below are suggestions for double checking the solution to see if a local (versus global) maximum or minimum was found, or to see if other solutions exist, or to see if there is further improvements to be made in the current solution.

1. The user can reduce the values of RERRO and/or AERRO to force the solver to achieve greater accuracy in the final solution. This will improve the results of the previous solution, but will not find new solutions. The user should be warned that accuracy is asymptotic--that many more calls to PROCEDURE will be required for each order of magnitude reduction in the error tolerance variables. Furthermore, the error tolerance of the underlying SINDA/FLUINT solutions (as governed by EBALSA, RERRF, REBALF, etc.) should be also reduced if error tolerance at the Solver level is to be reduced, and this makes each call to PROCEDURE more expensive.

2. By default, the Solver stops when it has been unable to improve the solution after two (NCONVO) consecutive search directions. By changing NCONVO to 3, or in extreme cases to 4, the user will have more assurance that the returned solution is optimal. However, like RERRO and AERRO, this method is unlikely to find new solutions, but rather to better confirm previous solutions.

3. Because the Solver uses search methods, it can be sensitive to the choice of initial conditions of design variables. Therefore, to search for potentially new and perhaps improved solutions, try perturbing the initial conditions as much as possible and rerunning the analysis. *This is the recommended approach* for finding global minima or maxima, or at least for quantifying the uncertainty in any one optimization. An automated way of scanning for initial conditions is by using a prior call to DSCANLH (Section 5.15.2), and the DSCANFF (Section 5.15.3), with appropriate limits on design variables, can be used to validate a solution. For example:

```
NLOOPO = 20     $ quick scan
CALL DSCANLH
NLOOPO = 200    $ back to full # of PROC calls
CALL SOLVER
```

4. Similarly, not all search methods will return the same result. To confirm a solution or to look for new solutions, rerun the Solver with a new value of METHO. To look for new solutions, start with the same or different initial conditions as before, perhaps by repeating a run with different values of METHO. It is also possible to call the Solver twice (consecutively in one run):

```
METHO   = 1
CALL SOLVER
METHO   = 2
CALL SOLVER
```

   but since the second call starts with the solution of the previous call, the above method is unlikely to return a novel solution. However, the above method is a good check on the validity of the previous solution.

5.11.4 Solution With Violated Constraints

If the Solver reports convergence, yet some constraints are violated (as listed in DESTAB or CSTTAB), the Solver was “stuck” and unable to make significant progress towards a valid solution. See also Section 5.11.2 for causes of stalled solutions.

The Solver can return an invalid solution, for example, during a MINIMAX curve fit if the ERRMAX (maximum error constraint variable) was initially specified too low, and too many constraints are violated. Perhaps no valid solution exists, or perhaps a new set of initial conditions need to be attempted after reviewing the problem set-up. This problem might also be a result of a excessively large value of RCACTO.

5.11.5 Solution Worse Than Before

Occasionally, the Solver will return a solution that is somewhat worse than a previously discovered solution (either within the same call to SOLVER as reported in the SOLOUTPUT CALLS, or as found by a previous call to SOLVER). This is often an indication of excessive error (noise) in the network solution routines used in PROCEDURE, which can prevent the Solver from making intelligent decisions. After improving the accuracy or convergence of the PROCEDURE solutions (Section 5.11.2.3), if the problem still persists the user can increase RDERO/ADERO, perhaps using MDERO=1, and should avoid limiting changes that can be attempted by the Solver (via RCHGO, PUSHO, etc.). This problem might also be a result of a excessively large value of RCACTO.

5.11.6 Solution on the Edge of Infeasibility

The underlying model must be tolerant of some degree of violation of constraints (and formula-based design variable limits, which are treated similarly). If a model would fail if such a constraint were violated (e.g., zero sum of conductors, negative diameters, etc.) then the user must impose tighter restrictions than initially desired on the constraints.

These added restrictions can take the form of tighter limits, but can also be regulated via the RCVIO option (which controls the degree of violation acceptable on formula-based design variable limits), or via the RCHGO and ACHGO options (which limit the changes that a design variable can undergo), probably at the expense of more calls to PROCEDURE.

This problem is particularly relevant when the desired solution itself lies near a region of infeasibility and perhaps model failure. For example, the user might be trying to size a condenser to achieve zero subcooling at the exit, but the model may not be able to tolerate incomplete condensation, which during the course of iterations is almost sure to occur. Therefore, the user may have to either make the model more tolerant of such excursions, or design instead for a nonnegligible degree of subcooling with perhaps limits on the Solver as described above to prevent excessive excursions.

5.12 Brief Solver Example: Constrained Optimization

Refer also to the Sample Problem Appendix for different examples.

The three node bar problem from Section 1.5.3 will be extended. First, five nodes will be used instead of three, and the cross section (e.g., the thicknesses at 5 points) will become design variables X1 through X5. The purpose of the analysis is the same: to find the shape of the bar that has the smallest mass while preventing the temperature of the first node from exceeding 500 degrees under steady-state conditions. However, unlike the example in Section 1.5.3, no constraints on the shape of the bar are imposed.

While the underlying analysis that is required is steady-state, the capacitances of each node will be faithfully described on the basis of the new dimensions and design variables. This allows the OBJECT to be defined simply in terms of the capacitance of the nodes, which will be updated by SINDA/FLUINT automatically. (Alternatively, a single complex formula for OBJECT, based on the design variables X1 through X5 and the registers DENS, LONG, and WIDE could have been written in SOLVER DATA, eliminating the need to update OBJECT at the end of PROCEDURE.)

The conductance from one cross section to another, and the radiation off the last face are all described algebraically in terms of the dimensions and design variables. A constraint is imposed such that the final answer (although perhaps not intermediate ones) do not permit the temperature of the node where the heater is applied to exceed 500 degrees.

The PROCEDURE block is then simply a steady state solution, followed by an update of the OBJECT value. SOLVER is called from OPERATIONS after building the configuration.

A SOLOUTPUT block is added to track progress, as well as some customized tracking written in SOLOGIC 1, which is called more often.

The complete input file is as shown:

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM, USING SOLVER
        OUTPUT          = baropt.out
        USER1           = baropt.usr
        MODEL           = TEST
HEADER REGISTER DATA
        DENS            = 0.3
        CP              = 0.2
        CON             = 0.5/12.0
        MPER            = DENS*WIDE*LONG/RESOL
        EMIS            = 0.1
        WIDE            = 1.0
        X1              = 1
        X2              = 2
        X3              = 3
        X4              = 4
        X5              = 5
        LONG            = 3.0
        HR2MIN          = 60.0
        RESOL           = 5.0
HEADER SOLVER DATA
        METHO           = 3
        NERVUS          = 1
        NLOOPO          = 300
HEADER CONTROL DATA,GLOBAL
        EBALSA          = 0.001
```

```
HEADER NODE DATA, SUB1
       1,500.0,(0.5*X1)*CP*MPER
       2,500.0,(0.5*(X1+X3)+X2)*CP*MPER
       3,500.0,(0.5*(X2+X4)+X3)*CP*MPER
       4,500.0,(0.5*(X3+X5)+X4)*CP*MPER
       5,500.0,(0.5*X5)*CP*MPER
       -99, -460., 0.0
HEADER CONDUCTOR DATA, SUB1
C FORMULA FOR WEDGE
       12,1,2,CON*(X1-X2)*WIDE/(LONG/RESOL)/LN(X1/X2)
       23,2,3,CON*(X2-X3)*WIDE/(LONG/RESOL)/LN(X2/X3)
       34,3,4,CON*(X3-X4)*WIDE/(LONG/RESOL)/LN(X3/X4)
       45,4,5,CON*(X4-X5)*WIDE/(LONG/RESOL)/LN(X4/X5)
       -599, 5, 99, EMIS*X5*WIDE*sbcon/(HR2MIN*144.)
HEADER SOURCE DATA, SUB1
       1, 10.0/(btuhrwat*HR2MIN)
HEADER DESIGN DATA
       0.0010 <= X1
       0.0011 <= X2
       0.0012 <= X3
       0.0013 <= X4
       0.0014 <= X5
HEADER CONSTRAINT DATA
       SUB1.T1        <= 500.0
HEADER OPERATIONS
BUILD TEST, SUB1
       NLOOPS         = 10000
       CALL SOLVER
       CALL TPRINT('SUB1')
HEADER SOLOGIC 1
       WRITE(NUSER1,99) LOOPCO,X1,X2,X3,X4,X5,OBJECT,SUB1.T1
99     FORMAT(I3,1P,7(1X,G12.5))
HEADER SOLOUTPUT
       CALL DESTAB
       CALL CSTTAB
HEADER PROCEDURE
DEFMOD SUB1
       CALL STDSTL
       OBJECT         = (C1+C2+C3+C4+C5)/CP
       CALL TPRINT('ALL')
       CALL GPRINT('ALL')
END OF DATA
```

Alternative, the definition of OBJECT could have been moved from PROCEDURE to SOLVER DATA, although submodel prefixes would then have been required:

```
HEADER SOLVER DATA
       OBJECT         = (SUB1.C1+SUB1.C2+SUB1.C3+SUB1.C4+SUB1.C5)/CP
       ...
```

For each of the three methods, using default settings of control parameters, the results (exact values of which vary from version to version!) are as follows:

Value	METHO = 1	METHO = 2	METHO = 3
PROC calls	112	84	51
OBJECT (mass)	7.80	7.83	7.83
THEAT	501.37	500.11	499.86
X1	0.56	0.67	0.64
X2	0.83	0.86	1.43
X3	0.86	0.75	0.89
X4	0.85	0.86	0.63
X5	38.5	38.7	38.0

Although not a universal conclusion, METHO=3 (sequential quadratic method) clearly performed the best in the above case. The trend is clear though: the optimum shape is a thin bar with a very wide face where radiation, a relatively weak heat transfer mode, dominates. If this shape is deemed unmanufacturable, some limits on the shape could be imposed as was done in Section 1.5.3.

For example, to prevent the cross section from changing abruptly (perhaps limiting the resulting design to a twofold change in thickness from one cross section to the next), the DESIGN DATA section could have been replaced with:

```
HEADER DESIGN DATA
       0.0010 <= X1
       0.0011 <= X2 <= 2.0*X1
       0.0012 <= X3 <= 2.0*X2
       0.0013 <= X4 <= 2.0*X3
       0.0014 <= X5 <= 2.0*X4
```

The above upper limits represent formula-based limits. Effectively, the represent unnamed constraints and therefore, they might be violated slightly during the course of the solution (unlike the fixed lower limits). This means that the above formulation is treated differently from the following, seemingly equivalent formulation since the lower limits are now variable:

```
HEADER DESIGN DATA
       MAX(0.0010, 0.5*X2) <= X1
       MAX(0.0011, 0.5*X3) <= X2
       MAX(0.0012, 0.5*X4) <= X3
       MAX(0.0013, 0.5*X5) <= X4
       0.0014 <= X5
```

While this effect is of no consequence in the above model, it might cause problems in other models if the violation of such limits results in an invalid network.

Before leaving this example, note that the initial conditions for X1 through X5 were not equal, and neither were the lower limits for those variables. This was intentional to avoid any adjacent values (e.g., X1 and X2, X3 and X4) from being equal. If that had happened during the course of the solution, then the “wedge” formula for conductance would have experienced an arithmetic fault, and the alternate KA/L formula would need to be substituted. While SOLOGIC 1 is used above for customized outputs, one of its other uses is to enable the user to detect and correct such occurrences before the Solver fails in an internal call to UPREG. In that block, the user could have updated the conductances directly (using Fortran) on the basis of incoming design variables, or could have changed a variable within each expression. For example, if instead the conductances had been defined using new registers TERM12, TERM23, etc.:

```
23,2,3,CON*TERM23*WIDE/(LONG/RESOL)
```

then the following logic in SOLOGIC 1 would have solved the problem:

```
if(abs(1.0-x3/x2) .le. 1.0E-5)then
       term23  = x2
else
       term23  = (x3-x2)/alog(x3/x2)
endif
```

Since SOLOGIC 1 represents an unique opportunity to adjust the registers after the Solver has updated the design variables but before a call to UPREG is made, other uses will surely arise.

5.13 Brief Solver Example: Goal Seeking

Refer also to the Sample Problem Appendix for different examples.

Goal seeking is essentially one-dimensional optimization, with the difference between the OBJECT and the target value (GOAL) minimized.

The capillary pumped loop (CPL) presented in the Sample Problem Appendix (sample problem E) was a variable conductance device, meaning that the evaporator temperature remained relatively constant because the reservoir temperature was controlled, and the reservoir was never completely full of liquid. If the reservoir had been smaller, such that as the condenser emptied it had become completely full of liquid, then the reservoir would no longer control the system pressure, which would now rise or fall as needed to balance energy in and out of the loop.

Such a constant-volume, fixed conductance device represents a problem to codes such as FLUINT. (A similar problem occurs in vapor compression cycles and unless either the high pressure or the low pressure is specified.) By itself, FLUINT is able to calculate the system pressure only if tanks are used to model the system such that there is a basis for predicting pressure changes: conservation of mass. If STEADY is used or if junctions are used instead of tanks, then a plenum must exist somewhere to provide a reference pressure. The user must add additional logic to adjust the pressure of that plenum as needed to balance energy.

The Solver can be used to perform such adjunct solutions, at least in steady-state analyses. The idea is to let the Solver find the system pressure that balances energy and perhaps meets other constraints.

The purpose of the following analysis is to find the pressure that balances energy such that the condenser is 80% empty (contains 20% liquid), or at least preserves a 5K subcooling at the condenser exit. The CPLGAS problem (the full model, not the second reduced model CPLPDQ) was modified as follows:

1. The model was converted to run under steady state conditions with the plate sink fixed at 6.6°C, and a constant 400W applied to the evaporators. The start-up logic was removed, and the call to SPRIME was left operating continuously, and the wick capillary radius was lowered to eliminate spurious deprimes that might occur as the Solver tested different configurations during its iterations.
2. The temperature of the plenum was redefined as a register called TSAT instead of being fixed at 29°C.
3. TSAT was named as the single design variable, with a lower bound of 6.6+5.0=11.6°C (corresponding to minimum subcooling in the limit of an infinite heat exchanger).
4. A PROCEDURE block consisting of a single call to STEADY was added, along with a call to SOLVER as the sole operation in OPERATIONS (after initializing FK factors).
5. A minimum subcooling of 5K was added via the following constraint expression (where TL9 is the condenser exit temperature):

```
5 <= TSAT - CPL.TL9
```

6. The GOAL was set to 0.8, meaning 80% void in the condenser. At the end of PROCEDURE, OBJECT was set to measure the current void:

```
OBJECT  = (AL1+AL2+AL3+AL4+AL5+AL6+AL7+AL8+AL9)/9.0
```

Thus, the Solver adjusts the plenum TL (via the design variable TSAT) and therefore its saturation pressure until the goal is met and the constraint is not violated, or until the goal is achieved as close as possible without violating the constraint.

Method 1 (METHO=1) took 28 calls to PROCEDURE to return a result of TSAT=14.5°C, with the subcooling at 5K (i.e., and active constraint) and the target void fraction met. During the course of the solution, one or more of the STEADY solutions failed to converged, so the NERVUS value was set to 3 and the final answers were accepted as reasonable.

Note that the above problem set-up is slightly dangerous since each value of void fraction is actually discontinuous, and therefore the resulting summation is not continuous over the entire range of possible solutions. However, the resulting summation is continuous near the answer, and hence no difficulties were encountered.

5.14 Brief Solver Example: Test Data Correlation

Refer also to the Sample Problem Appendix for different examples.

To demonstrate data correlation methods without delving into the details of a specific model, consider a dummy curve of "test data" to be correlated at 13 points as shown in Figure 5-5. This data is to be correlated using a third order polynomial of the form $y = A+Bx+Cx^2+Dx^3$. In other words, the task of the Solver is to find the values of A, B, C, and D that best fit the fake test data at the 13 points shown in Figure 5-5.

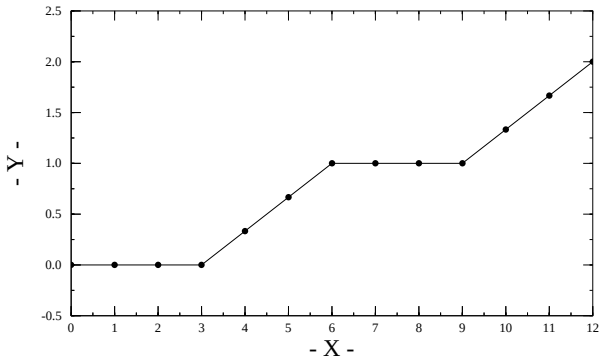


Figure 5-5 Fake Test Data to Correlate Against

5.14.1 Least Squares (RMSERR and SUMSQR) Examples

The COMPARE routine will be used to generate comparisons and reports of those comparisons. To employ this routine, both the "test data" and the "predictions" must be stored as SINDA/FLUINT arrays. While arranging data into these arrays is somewhat contrived for this non-SINDA example, in real cases data would likely either already be available in this form or could be easily cast into that form using routine such as PREPLIST, PREPDAT1, or PREPDAT2.

To use SINDA/FLUINT and therefore the Solver, a dummy thermal submodel named *fake* is generated. The "test data" is stored in array #1 as follows:

```
1 =     0.,     0.0
        3.,     0.0
        6.,     1.0
        9.,     1.0
        12.,    2.0
```

To generate the 13 comparison points a DO LOOP is added to OPERATIONS to look up the 13 points along the curve using the one-dimensional interpolation routine D1DEG1 (Section 7). The test data is stored in array 10 of submodel "fake."

Four registers, all of which are design variables, are declared: *able*, *baker*, *charlie*, and *delta*, corresponding to A, B, C, and D in the equation $y = A+Bx+Cx^2+Dx^3$ presented above. These design variables are all declared to be initially equal to 0.01.*

* Performance of the Solver is fairly sensitive to the initial values of these constants. In most real correlation cases, of course, the initial values are usually not very far from final values. In this case, of course, no knowledge of the general form of the final curve fit is known a priori.

To use the Solver, a PROCEDURE block must be defined. In this case, the procedure consists of generating and storing the 13 “prediction points” using the current values of the four design variables. The predictions are stored in array 20 of submodel “fake.” Then, the COMPARE routine is used to produce the data needed for the Solver: in this case the root-mean-square error RMSERR, which will be stored as the OBJECT of the minimization.

Finally, in OPERATIONS after the call to SOLVER, the values of the design variables are tabulated using DESTAB, and COMPARE is called once more as an output routine to summarize the results of the comparison.

The complete input file is as follows:

```
header options data
title fake fitting: least squares via RMS
        output  = fakefit.ls
header array data,fake
 1 =    0.,     0.0
        3.,     0.0
        6.,     1.0
        9.,     1.0
        12.,    2.0
C “TEST DATA”
        10 = SPACE,13
C “PREDICTIONS”
        20 = SPACE,13
header node data,fake
header register data
        able    = 0.01
        baker   = 0.01
        charlie = 0.01
        delta   = 0.01
header design data
        able
        baker
        charlie
        delta
header operations
build ff,fake
defmod fake
        do 1 itest=0,12
                xtest   = float(itest)
                call d1deg1(xtest,fake.a1,fake.a(10+itest+1))
1       continue
        call solver
        call destab
        call compare(a20,a10,0,0,’report’,-1)
header procedure
defmod fake
        do 1 itest=0,12
                xtest   = float(itest)
                a(20+itest+1) = able + baker*xtest
     .                        + charlie*xtest**2
     .                        + delta*xtest**3
1       continue
        call compare(a20,a10,0,0,’rmserr’,object)
end of data
```

Adding HEADER SOLVER DATA and applying METHO=1 (which could equivalently have been set in OPERATIONS before the call to SOLVER) yields similar results using a slightly different method on this unconstrained problem.

Replacing ‘rmserr’ with ‘sumsqr’ in the call to COMPARE in PROCEDURE yields a true least square fit, which should be mathematically identical to the RMSERR fit. After all, the only difference is whether the RMS error or the square of the RSS error is minimized.

In practice, however, the Solver often has more difficulty with the SUMSQR problem because of the magnitude of changes in OBJECT over the initial value. This can be seen in Table 5-3, which lists the summary of several runs made with various least squares method variations (these results will vary somewhat from version to version!).

Table 5-3 Results of Various Least Squares Methods

Value	METHO = 1 RMS Error	METHO = 2 RMS Error	METHO = 1 Least Square	METHO = 2 Least Square	METHO = 1 LS (Double)
PROC calls	68	41	40	41	40+46
ERRMAX (K)	0.38	0.53	0.54	0.54	0.40
STD DEV (K)	0.17	0.21	0.21	0.22	0.17
able	0.0105	0.00999	0.00998	0.00998	0.0118
baker	0.0126	0.00985	0.00980	0.00980	0.0189
charlie	0.0177	0.00840	0.00794	0.00793	0.0144
delta	-4.57E-4	4.36E-4	4.80E-4	4.81E-4	-2.07E-4

While all variations found reasonable solutions, the improvement in the least squares method when SOLVER is called twice (“Double,” the last column in Table 5-3) demands an explanation. In that case, the Solver was able to find a very good fit (as good as a single call to SOLVER with RMS error and METHO=1: the first column in Table 5-3). The difference between calling SOLVER once and calling it twice is that, in the second run, the Solver rescaled the problem starting from a closer answer, and was therefore better able to make progress: the changes to OBJECT and the design variables were not as large in the second call. (Calling SOLVER twice in a row is a method recommended in Section 5.11.3 for double checking an answer; calling DSCANLH instead of the first call to Solver is often preferred.)

5.14.2 Maximum Error (MAXERR) Example

To minimize the maximum error instead of the RMS error, the ‘rmserr” argument can be replaced by ‘maxerr’ in the call to COMPARE made within PROCEDURE in the previous case. As noted in Section 5.10.3.3, it is theoretically a bit dangerous to directly minimize the maximum error. The reason is that at the final solution the objective OBJECT will be met at two distinct points, causing a discontinuity. However, the Solver tolerates this condition in many cases, and setting up such a correlation is easy.

The results of such a correlation are presented in Table 5-4, which also shows the best result from the previous subsection for comparison purposes. Interestingly in this case, there was very little distinction between METHO=1 and METHO=2, although this is not a general conclusion.

Table 5-4 Results of Simplified MAXERR Minimizations

Value	METHO = 1 RMS Error (comparison)	METHO = 1 Max Error	METHO = 2 Max Error
PROC calls	68	12	12
ERRMAX (K)	0.38	0.483	0.483
STD DEV (K)	0.17	0.276	0.276
able	0.0105	0.00999	0.00999
baker	0.0126	0.00989	0.00989
charlie	0.0177	0.00860	0.00860
delta	-4.57E-4	6.38E-4	6.38E-4

5.14.3 MINIMAX Example

To perform a “true” minimized maximum error per the methods outlined in 5.10.3.3, a new design variable called *errmax* is added, and is used as the objective in SOLVER DATA. A CONSTRAINT DATA section is then added to generate the 13 constraint variables needed for each of the 13 comparison points, and the call to COMPARE is modified to update these constraint variables.

Because more iterations will be needed, the NLOOPO value is also increased about the default of 100. Otherwise, the input file is similar to that presented above:

```
header options data
title fake fitting: minimax
        output  = fakefit.minimax2
header array data,fake
 1 =    0.,     0.0
        3.,     0.0
        6.,     1.0
        9.,     1.0
        12.,    2.0
C “TEST DATA”
        10 = SPACE,13
C “PREDICTIONS”
        20 = SPACE,13
header node data,fake
header register data
        able    = 0.01
        baker   = 0.01
        charlie = 0.01
        delta   = 0.01
        errmax  = 1.0
header design data
        able
        baker
        charlie
        delta
        errmax
header solver data
        nloopo = 500
        object = errmax
header constraint data
        gen const,1,13,1,-errmax,errmax
header operations
build ff,fake
```

```
defmod fake
       do 1 itest=0,12
              xtest   = float(itest)
              call d1deg1(xtest,fake.a1,fake.a(10+itest+1))
1      continue
       call solver
       call destab
       call compare(a20,a10,0,0,'report',-1)
header procedure
defmod fake
       do 1 itest=0,12
              xtest   = float(itest)
              a(20+itest+1) = able + baker*xtest
     .               + charlie*xtest**2
     .               + delta*xtest**3
1      continue
       call compare(a20,a10,0,0,'minimax',const1)
end of data
```

Table 5-5 presents the results of the MINIMAX method, again presenting the previous RMS error case for comparison. Note that, being a constrained optimization, three possible values of METHO exist instead of 2, and variations of constraint control parameters such as RCACTO can have an effect on the performance of the Solver.

Table 5-5 Results of Formal MINIMAX Correlations

Value	**METHO = 1 RMS Error (comparison)**	**METHO = 1 Minimax**	**METHO = 2 Minimax**	**METHO = 2 Minimax RCACTO=0.3**	**METHO = 3 Minimax**
PROC calls	68	192	98	112	71
ERRMAX (K)	0.38	0.28	0.43	0.42	0.62
STD DEV (K)	0.17	0.17	0.26	0.25	0.34
able	0.0105	0.0108	0.0113	0.0121	-0.00019
baker	0.0126	0.0128	0.0113	0.0121	0.00098
charlie	0.0177	0.0245	0.0113	0.0121	0.00262
delta	-4.57E-4	-1.14E-3	3.78E-4	3.02E-4	1.28E-3

In this case, METHO=1 produced a very good fit, but took many PROCEDURE calls to do so. The sensitivity to RCACTO is also demonstrated in Table 5-5 for the METHO=2 case.

The curve fits for the best of both RMS and MINIMAX methods are presented in Figure 5-6.

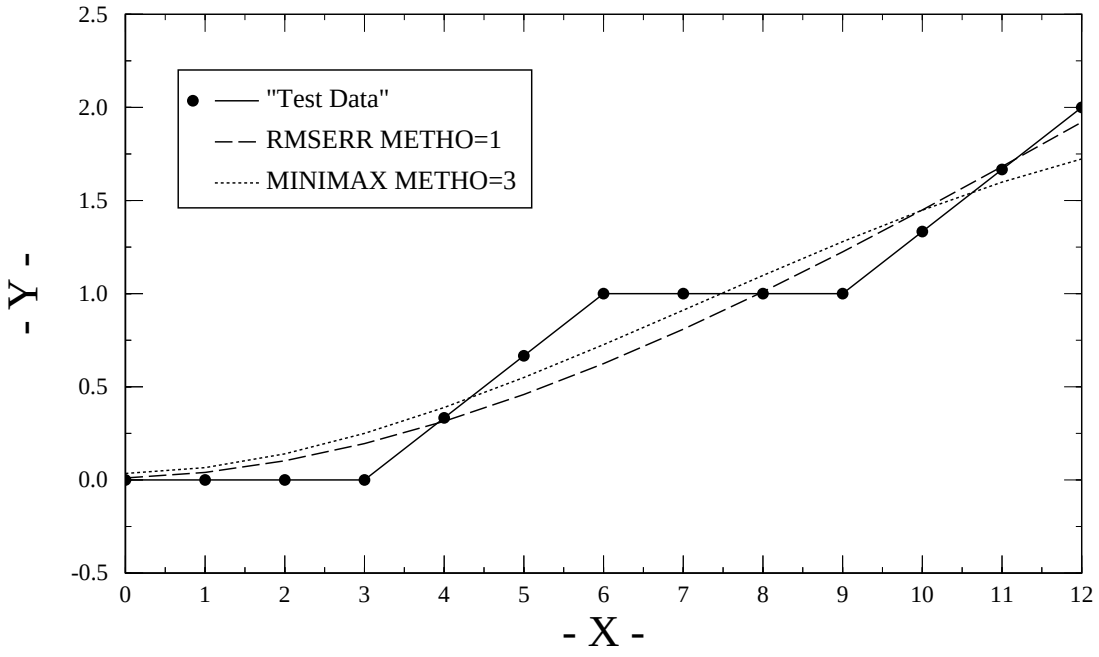


Figure 5-6 Summary of Best Methods

5.15 Design Space Scanning

This section describes utilities for sampling a design space (range of valid design variable values), usually in order to determine a good starting point for a subsequent optimization using the SINDA/FLUINT Solver or to investigate the design space parametrically.

The Solver is a gradient-based optimization method: it starts with the user-supplied initial values of design variables and then begins seeking a better solution based on local gradients measured at that starting point. It can be therefore be fooled by "local minima:" good solutions that are near the starting point, but that are perhaps not the best possible solution ("global minimum"). This problem is rarely experienced during practical design optimization. Such optimizations are normally better described as "fine tuning" an already good design that is heavily constrained. However, multiple solutions (and often equally good ones) *are* fairly common when the Solver is used to calibrate a model to test data. Multiple solutions might also exist when using the Solver to seek the worst case design scenario.

Three methods are described: a parametric sweep of a single design variable, a full (and therefore costly) scan, and a partial scan based on statistical sampling techniques.

All of the subsequent methods are sensitive to tight (equality or near-equality) constraints, since the chances of randomly hitting a valid point become diminishingly small. Refer to Section 5.15.4 for guidance on overcoming this issue.

5.15.1 Parametric Sweep of a Design Variable

This section describes a simple routine that performs a parametric sweep of a single design variable. This is not intended as a replacement for the more complete routines (e.g., SOLVER, DSCANLH) introduced elsewhere. Rather, this feature is useful for either debugging a problem or investigating trends for a reduced problem space.

Calling Sequence:

```
CALL DVSWEEP(dvnam)
```

where:

dvnam . . . . . Design variable name, in single quotes

DVSWEEP makes NLOOPO calls to PROCEDURE using even increments of the design variable from the lower to the upper limit. SOLOUTPUT CALLS is called after each evaluation. Both such limits must be specified in DESIGN DATA for that variable in order to invoke this routine.

Upon completion, DVSWEEP returns the best viable value it tested (though *not* the best possible design). The "best viable design" means the design with the optimum OBJECT that did not violate *any* constraints. If all trial designs violated one or more constraints, then a caution is issued and the original variable value is returned. (If equality constraints are used, or if so many constraints exist that the a scan is prevented from finding a viable design, see Section 5.15.4 for more guidance.)

The last PROCEDURE call will rarely correspond to the best design found. To force an extra (and final) PROCEDURE call using the best design, set NEWPRO=1 in OPERATIONS or SOLVER DATA. Alternatively, use a separate CALL PROC following the DVSWEEP call in OPERATIONS, perhaps setting an integer control parameter to identify this special run (perhaps to turn on output operations):

```
CALL DVSWEEP('power_in')
LAST_ONE= 1 $ Flag for output operations
CALL PROC   $ Repeat procedure using best-found design
```

Restriction: The design variable must have *both* lower and upper limits to use this utility. If those limits are defined by expressions, and if the values of those limits changes significantly during the run, then it is possible that situations will arise in which the algorithm cannot easily pick a valid set of design variable values. If this happens, the code will skip that evaluation point: less than NLOOPO points will be tested in this rare case.

Example:

```
CALL DVSWEEP('length')
```

Guidance: This routine ignores design points if trouble is encountered during the solution procedure. Use the NERVUS control constant in SOLVER DATA to adjust tolerance of nonconvergence and other problems: lower NERVUS values cause more design points to be ignored, but this routine does not abort if trouble is encountered (unlike SOLVER).

5.15.2 Latin Hypercube Scanning

The routine DSCANLH might be called before the Solver to search the design space for a good starting point. DSCANLH uses the same latin hypercube descriptive sampling algorithm used in the reliability estimation routine DSAMPLE (Section 5.20), except that design variables are used instead of random variables and the "probability distribution function" of such design variables is assumed to be uniform from the lower to upper limit. In other words, the likelihood of any one design variable value being picked is equal over the range of valid values.

A major restriction of DSCANLH is that all design variables **must** *have both an upper and lower limit. The scan is most efficient when these limits are fixed, or at least are not strong functions of results or of other design variables. In other words, a finite design space must be specified, and the boundaries of that design space should be relatively constant.*

In pseudo-code, a simplified rendition of the routine DSCANLH is as follows:

```
SUBROUTINE DSCANLH
[initialization]
[for NLOOPO iterations:]
       CALL SLLOG0    $ Optional initialization
       [select new trial design variables]
       CALL SLLOG1    $ Optional customization
       CALL UPREG     $ update registers and formulas
       CALL PROC      $ call for a new evaluation:
                      $ new OBJECT, new constraints
       CALL SOLOUT    $ Optional outputs
       [save if best viable design so far]
[restore best viable design, CALL UPREG]
if NEWPRO=1, CALL PROC $ repeat evaluation of best
CALL SOLOUT    $ Optional outputs
```

Simply put, if the user can afford NLOOPO evaluations (calls to PROCEDURE), DSCANLH maximizes the exploration of the design space within that computational budget. The latin hypercube method means that if NLOOPO=10, then ten values of each design variable are tested in 10 unique combinations that are chosen randomly. Because of this underlying randomness, two successive calls to DSCANLH with the same value of NLOOPO will seldom return the same best design point. Furthermore, it is more efficient to make one call with twice the resolution (twice the value of NLOOPO) than two such identical calls.

To illustrate, consider a case with two design variables, A and B, that vary between A_L and A_H, B_L and B_H, respectively. If NLOOPO is 5, then 5 total designs will be evaluated at various random combinations of A and B. Five different values of A will be evaluated, one in each trial design:

$A_N = A_L + (2*N-1)*(A_H-A_L)/(2*NLOOPO)$
where N = 1, 2, ... NLOOPO, N in random order.

The variation in the design variable B is calculated analogously, except that the value of N might be different. Graphically, Figure 5-7 depicts two possible samplings of the above case with NLOOPO=5.

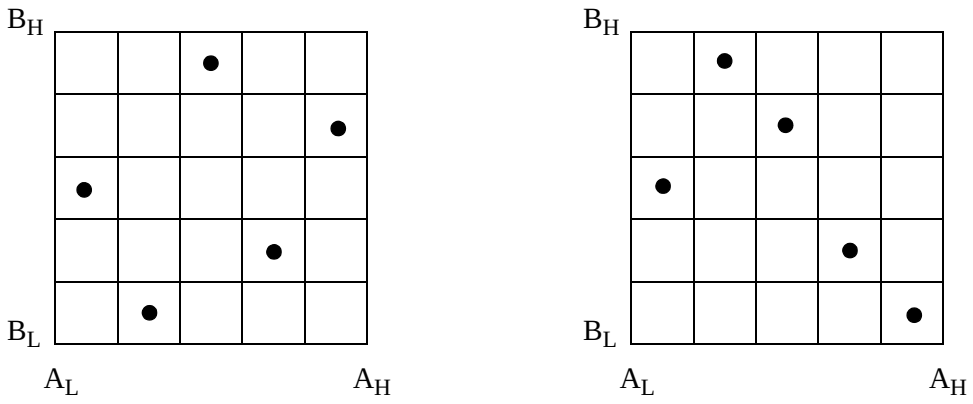


Figure 5-7 Two Possible Latin Hypercube Samplings of a Two Dimensional Design Space with a Discretization Level of Five

To illustrate example usage, assume that a user has decided (based on model execution speed) that 200 total calls to PROCEDURE is an appropriate budget for the current optimization. The user might "spend" 10 to 50 of those PROCEDURE executions in a preliminary call to DSCANLH, followed by the remainder (150 to 190) in a call to Solver. For example, the following lines might appear in OPERATIONS:

```
NLOOPO = 25
CALL DSCANLH   $ Perform exactly 25 evaluations
NLOOPO = 175
CALL SOLVER    $ Perform up to 175 evaluations,
               $ starting with the best point
               $ found by DSCANLH
```

Note that the number of PROCEDURE calls in DSCANLH is user-selected, but that the number of calls in Solver are unknown, and in fact Solver may not complete its task in the allotted number of calls. In general, a call to DSCANLH reduces the calls that the Solver will require to complete its task.

Finally, note that DSCANLH is not formally an "optimization algorithm," even though it returns the best design found during its searches. In other words, it shouldn't be used by itself, but in combination with the Solver.

The DSCANLH Routine

DSCANLH has no arguments, and should only be called from either OPERATIONS or REL-PROCEDURE:

```
CALL DSCANLH
```

DSCANLH performs NLOOPO statistical evaluations of PROCEDURE, returning the best viable design it tested (though *not* the best possible design). The “best viable design” means the design with the optimum OBJECT that did not violate *any* constraints. If all trial designs violated one or more constraints, then a caution is issued and the original design point is returned. (If equality constraints are used, or if so many constraints exist that the a scan is prevented from finding a viable design, see Section 5.15.4 for more guidance.)

The last PROCEDURE call will rarely correspond to the best design found. To force an extra (and final) PROCEDURE call using the best design, set NEWPRO=1 in OPERATIONS or SOLVER DATA. Alternatively, use a separate CALL PROC following the DSCANLH call in OPERATIONS, perhaps setting an integer control parameter to identify this special run (perhaps to turn on output operations):

```
CALL DSCANLH
LAST_ONE= 1 $ Flag for output operations
CALL PROC   $ Repeat procedure using best-found design
```

Restriction: All design variables must have both lower and upper limits to use this utility. If those limits are defined by expressions, and if the values of those limits changes significantly during the run (especially if they are strong functions of other design variables), then it is possible that situations will arise in which the algorithm cannot easily pick a valid set of design variable values. If this happens, the code will skip that evaluation point: less than NLOOPO points will be tested in this rare case.

Guidance: This routine ignores design points if trouble is encountered during the solution procedure. Use the NERVUS control constant in SOLVER DATA to adjust tolerance of nonconvergence and other problems: lower NERVUS values cause more design points to be ignored, but this routine does not abort if trouble is encountered (unlike SOLVER).

5.15.3 Full Factorial Scanning

While computationally efficient, the latin hypercube technique is more likely to miss the best starting point than is a full scan of every possibility (subject to finite design space discretization, of course). Therefore, a alternate method is also available: the “full factorial scan.” Simply put, a full factorial scan checks every point in the discretized design space: a dot would appear in every box in Figure 5-7.

To be consistent with both DSCANLH and the Solver, the DSCANFF full factorial method retains the meaning of NLOOPO as “the total number of PROCEDURE calls” rather than “the discretization level of each design variable” (whereas these are equivalent concepts in DSCANLH).

However, unlike latin hypercube methods, with full factorial scanning the user can specify the number of variations each design variable independently, with an ultimate limit of NLOOPO points being evaluated. Also, some design variables can be excluded from the scan (i.e., their current value will be used). This degree of customization is provided by specifying a SINDA array containing integer values.

The user can also choose to let the program choose the number of values for one or more variables such that the total number of evaluations is less than NLOOPO. For example, if there were four design variables and NLOOPO were equal to 100, then each design variable would be tested at three different values (3**4 = 81 < 100). If in the same situation NLOOP were equal to 110, then the first design variable would be tested at four values and the rest at three (4*3**3 = 108 < 110).

The DSCANFF Routine

DSCANFF should only be called from OPERATIONS.* DSCANFF has a single argument, which should be an integer SINDA array reference:

```
CALL DSCANFF(smn.NAn)
```

where:

smn.NAn. . . . . A reference to integer array “n” in submodel “smn.” The first entry in the array describes the number of variations in the first design variable, the second describes the number of variations in the second design variable, etc. A positive value directly specifies the number of values to use, evenly spaced between the upper and lower limits, and exclusive of them. A value of zero means to hold that design variable constant. A value of one means to use a single value for that design variable: the average of the upper and lower limits. A value of two means to use just two limiting values. The code will return immediately with an error message if the number of requested points exceeds NLOOPO.

A input value of -1 means to apply as many variations as can be afforded within the limits of NLOOPO total evaluations, potentially zero. Input negative one (“-1”) as this argument as a single integer value or as an integer variable to apply this scheme to all design variables (the default if this argument is omitted).

* Although unlikely, it may also be called from RELPROCEDURE.

Example:

```
HEADER DESIGN DATA
      1.5 <= LENGTH <= 6.0
      0.03 <= WIDTH <= 0.04
      thickmin <= THICKNESS <= width/10
      0.0 <= cost <= 0.99
HEADER ARRAY DATA, MARGE
      101 = 5, 3, -1, 0 $ five lengths, three widths, as many
                        $ thicknesses as can afford, hold cost constant
HEADER OPERATIONS
BUILD ALL
BUILDF ALL
      CALL DSCANFF(MARGE.NA101)     $ Use array to select resolution
      CALL DSCANFF      $ Up to NLOOPO even evaluations, all des. var.
```

DSCANFF performs up to NLOOPO evaluations of PROCEDURE, returning the best viable design it tested (though *not* the best possible design). The “best viable design” means the design with the optimum OBJECT that did not violate *any* constraints. If all trial designs violated one or more constraints, then a caution is issued and the original design point is returned. (If equality constraints are used, or if so many constraints exist that the a scan is prevented from finding a viable design, see Section 5.15.4 for more guidance.)

The last PROCEDURE call will rarely correspond to the best design found. To force an extra (and final) PROCEDURE call using the best design, set NEWPRO=1 in OPERATIONS or SOLVER DATA. Alternatively, use a separate CALL PROC following the DSCANFF call in OPERATIONS, perhaps setting an integer control parameter to identify this special run (perhaps to turn on output operations):

```
      CALL DSCANFF(-1)
      LAST_ONE= 1 $ Flag for output operations
      CALL PROC   $ Repeat procedure using best-found design
```

Restriction: All design variables must have both lower and upper limits to use this utility. If those limits are defined by expressions, and if the values of those limits changes significantly during the run (especially if they are strong functions of other design variables), then it is possible that situations will arise in which the algorithm cannot easily pick a valid set of design variable values. If this happens, the code will skip that evaluation point: less than the designated number of points will be tested in this rare case.

Guidance: This routine ignores design points if trouble is encountered during the solution procedure. Use the NERVUS control constant in SOLVER DATA to adjust tolerance of nonconvergence and other problems: lower NERVUS values cause more design points to be ignored, but this routine does not abort if trouble is encountered (unlike SOLVER).

5.15.4 Scanning in the Presence of Equality Constraints

One of the primary purposes of the DSCANLH and DSCANFF routines (and to a lesser extent, the DVSWEEP routine) is to prescan the design space: to find a good starting point for a subsequent optimization (or calibration task or worst-case scenario seeking task, etc.) using the SOLVER.

This purpose can be subverted if no viable design can be found, where "viable" means "violating no constraints." In cases in which no viable design is found, the routines in this subsection will return the original design point, rendering them useless for prescanning purposes unless further actions are taken. Excessive constraints are tolerated or even helpful to the Solver, which can be initialized using an inviable design as long as sensible results and trends are returned from PROCEDURE. This is not true for design space scanning routines, since they are performing a sampling operation, not a search. In other words, they do not have the ability to discern which designs violate a constraint yet are "good enough to serve as a starting point."

In fact, if *any* equality constraints are used, then all of the routines in this subsection (DVSWEEP, DSCANLH, and DSCANFF) will faithfully perform a scan of the design space but no viable design will be found. An equality constraint means the upper and lower limits are identical, but even tight limits on the acceptable range can preclude a design space scanning routine from finding and returning a viable design. For example:

```
HEADER CONSTRAINT DATA
      pi <= circum/diam <= pi       $ equality constraint
      1.044 <= voltage <= 1.044     $ another equality constraint
      1.04 <= voltage2 <= 1.05      $ nearly an equality constraint
```

The chances that a design will be sampled that happens to return a value of *voltage2* between 1.04 and 1.05 is very small, and the chances that the scan will happen upon a design with a value of *voltage* that is identically 1.044 is essentially zero.

If the purpose of calling the scanning routine is to prescan for the Solver, then perhaps some of these constraints should be temporarily disabled using OFFCST,* and returned to active status prior to a call to Solver. For example:

```
HEADER OPERATIONS
BUILD ALL
      CALL OFFCST('voltage')        $ temporarily disable this constraint
      NLOOPO = 15
      CALL DSCANLH
      CALL RELCST('voltage')        $ re-inactivate the constraint
      NLOOPO = 150
      CALL SOLVER
```

* For unnamed constraints, call CSTNAMES in a prior run and use CSTTAB to reveal the internal machine-generated name, which can then be used in routines such as HLDCST, CHGCST, and OFFCST.

The above action (using OFFCST and RELCST) completely eliminates the constraint during scanning, which may be excessive. As an alternative, some tolerance might instead be added to tight or equality constraints as needed to increase the chances that a viable design will be encountered. To continue the above example, if three real registers (ctol1, ctol2, and ctol3) were added, then the above expressions could be expanded to:

```
HEADER REGISTER DATA
      ctol1 = 0.3       $ same units as “pi” (radians, perhaps)
      ctol2 = 0.1       $ unitless, 10% (see below)
      ctol3 = 0.3       $ unitless, 30%
HEADER CONSTRAINT DATA        $ toleranced constraints
      pi-ctol1 <= circum/diam <= pi+ctol1
      1.044*(1.0-ctol2) <= voltage <= 1.044*(1+ctol2)
      1.04*(1.0-ctol3) <= voltage2 <= 1.05*(1+ctol3)
```

These three tolerances (ctol1, ctol2, ctol3) would be chosen to temporarily render a larger viable design space, perhaps choosing each tolerance according to the criticality of the corresponding constraint.

Then, following a design space scan, these tolerances can be zeroed before starting the Solver:

```
HEADER OPERATIONS
      ...
      NLOOPO = 20
      CALL DSCANLH      $ prescan with a temporarily expanded viable space
      NLOOPO = 200
      CTOL1 = 0.        $ tighten or eliminate constraint tolerancing
      CTOL2 = 0.
      CTOL3 = 0.
      CALL SOLVER       $ optimize
```

5.16 Introduction to Reliability Engineering

The subsections comprising the second half of Section 5 describe the design reliability estimation (statistical design) features in SINDA/FLUINT, collectively referred to as the Reliability Engineering module. This subsection provides an introductory overview.

5.16.1 Reliability Engineering Terms

A basic understanding of statistics is a prerequisite for this module: the reader should understand the meanings of *mean* (or average) and *standard deviation*. It is also helpful to be familiar with the basic terms used in statistical design. These terms are defined as follows, and their interrelationship is depicted in Figure 5-8:

Random Variables
: The inputs that are uncertain. There needs to be at least one random variable to use the Reliability Engineering module, and most problems will have from one to ten of them even though many more are supported. The user must also specify the range of possible values for each random variable, perhaps in the form of a distribution function: a curve that describes the relative probability that any one value will occur.

Reliability Constraints
: The upper and/or lower limits on any response that distinguish a failed design from a successful one, or at least the identification of responses of interest even if no limits are applied. This can alternatively be called “Failure Limits.” Although most engineering analyses will have many reliability constraints, it is possible to have zero reliability constraints, and to rely entirely on postprocessing to investigate the reliability of the design.

Evaluation Procedure
: The analytic operations that are required to evaluate a given instance of a design, as defined by a set of specific values for each random variable. This procedure might be a simple steady state, a transient simulation, or some complex mixture of the two. Ultimately, to “evaluate” means to predict the responses implicit in the statement of the reliability constraints, and to see if any of them violated their limits, and if so by how much.

As an example, consider a simple rectangular heat transfer fin (extended surface) with a given length, width, and thickness, each of which is subject to manufacturing tolerances. The conductivity of the fin itself varies from unit to unit depending on the material lot. The fin must achieve 80% fin efficiency under a variety of root temperatures (or perhaps power inputs), environmental temperatures, and air flow rates (convection coefficients).

In this example, are many *random variables*: the length, the thickness, the width, the material conductivity, the root temperature (or power input), the sink or ambient temperature, and the air flow rate (or the corresponding convection coefficient). Nonetheless, a single *reliability constraint* exists: the fin efficiency is required to be greater than 0.8 (80%).

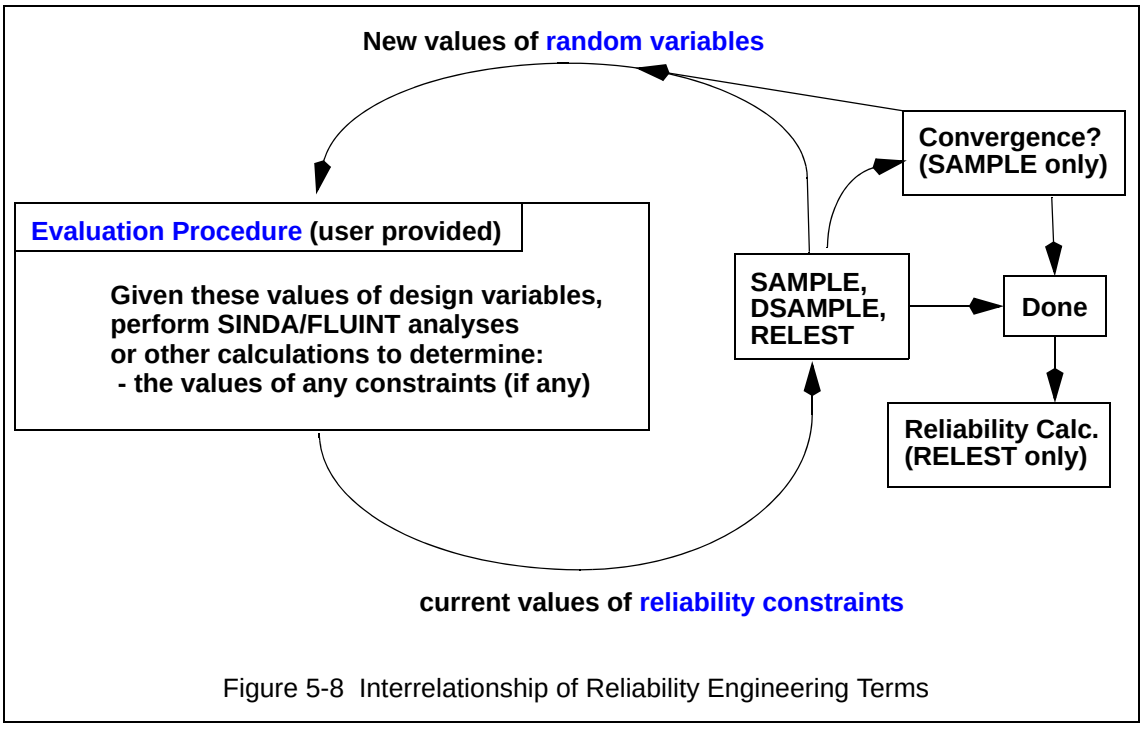


Figure 5-8 Interrelationship of Reliability Engineering Terms

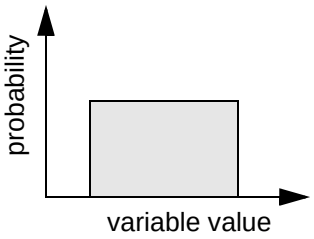
The *evaluation procedure* is as follows: Given a specific value for each of the random variables (i.e., a specific length, a specific sink temperature, etc.), find the steady state for that configuration using a SINDA/FLUINT solution routine, and then calculate the fin efficiency (and thereby update the reliability constraint).

In this example, a single SINDA/FLUINT steady state solution is needed to evaluate a single reliability constraint. In other cases, there may be many reliability constraints, some of which are trivial and do not require a SINDA/FLUINT solution. Any such combinations are valid, including the lack of any SINDA/FLUINT solution altogether, as long as the evaluation procedure updates the reliability constraints for a given set of random variables. The full range of SINDA/FLUINT functionality is available to help the user perform this evaluation.

5.16.2 Distribution Functions

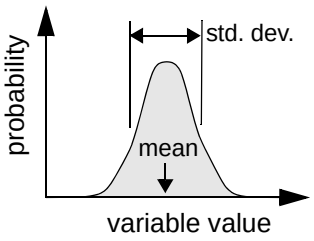
It is insufficient just to label a parameter as a random variable; the range of possible values must also be defined along with a probability of any one value occurring. This probability (or “frequency of occurrence”) is relative: what matters is how likely one value is compared to another.

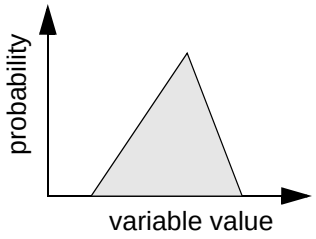
The simplest such *distribution function* is a uniform one: the variable may assume any value between a *lower limit* and an *upper limit*, and any value within this range is equally probable. The distribution function for such a variable is a rectangle as shown at the right. The



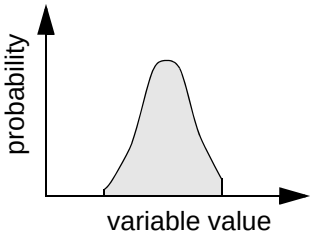
horizontal axis is the range of possible values for the variable, and the vertical axis is the relative frequency or probability for each of those values. The units of the vertical axis are irrelevant, all that matters is the shape.

Perhaps the most common nontrivial distribution is the *normal* or *Gaussian* distribution (shown at the right, not to scale), which can be entirely defined by two numbers: a *mean* and a *standard deviation*. The *coefficient of variation* is another common means of describing the breadth of the distribution as a unitless fraction of the mean: it is defined as the standard deviation normalized by (divided by) the mean. However, if the mean is zero then the coefficient of variation is meaningless.





Many other distributions are possible, including asymmetric ones (in which the mean, the median value, and the most likely point are all different from each other).



Sometimes, for example, a normal (Gaussian) distribution is appropriate, but a theoretical range between negative and positive infinity is nonphysical or would cause numerical problems: a truncated normal distribution is required (shown at right). Another possibility is a triangular (witch's hat) distribution, useful when all that is known is a most likely value plus a lower and upper bound (shown at left).

In fact, there are many types of distributions available (e.g., log normal, Weibull, Chi-square, etc.), each suited for a different purpose. It is also possible that a distribution function is produced from test or manufacturing data or from a previous analysis.

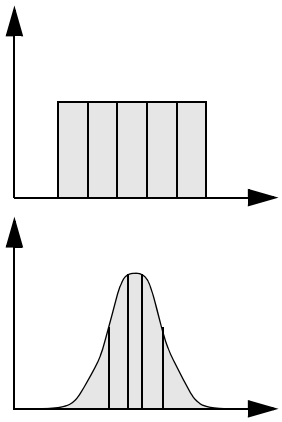
5.16.3 Sampling Techniques

Distribution functions for inputs (random variables) can be randomly sampled, monitoring the resulting responses (reliability constraints) and thereby developing a distribution function for those responses. Given a distribution for these responses, the reliability (probability of not violating a reliability constraint or failure limit) can be estimated as described in Section 5.16.4.1. Note that sampling is not the only technique for reliability estimation, as will be discussed in Section 5.16.4.2.

The simplest approach is that taken by the SAMPLE routine: a Monte Carlo method in which values of random variables are selected randomly according to their probability distribution functions. As an example, for a uniform distribution any value within the valid range is selected using a uniform random number generator. For normal distributions, random values are selected, but values near the center (the mean) will be generated more frequently than those at the extremes. This approach requires many samples (on the order of 1000: 100 to 10,000) and is therefore expensive. However, it yields the most information. Furthermore, the accuracy of the estimation can be controlled at least

relatively if not absolutely: the SAMPLE routine detects convergence as defined by negligible change in the selected responses and their associated limits (i.e., the reliability constraints) between any two consecutive samples.

A faster alternative is descriptive sampling, which is used in the DSAMPLE routine. This approach has a known cost: the user specifies the number of samples to be made (based on what they can afford). This number becomes the resolution with which the distributions in the random variables are subdivided. For example, if 100 samples are to be used, each input profile will be divided into 100 regions of equal probability. For uniform distributions, one hundred equal regions will be used. For normal distributions, the region near the mean will be more finely subdivided than the extremes such that each region is equally probably and therefore contains the same area (integral of probability over the random variable values: the *cumulative distribution function*). This subdivision is illustrated at the right using five subdivisions.



Once the distributions of the random variables have been subdivided, only one value from each subdivision (the center of the corresponding region in the cumulative distribution function) is sampled, since each of these values is as probable as any of the others. There is still randomness involved for more than one random variable: each cell, while sampled only once, is selected at random. For example, the 5^{th} cell of variable #1 might be combined with the 88^{th} cell of variable #2 in one run, but the 5^{th} cell of variable #1 might be combined with the 42^{nd} cell of variable #2 in a second run.

For the same number of samples, descriptive sampling will yield more accurate results than will Monte Carlo sampling. Typically, descriptive sampling takes about 10 to 20% of the number of samples that Monte Carlo sampling does to achieve the same accuracy. However, Monte Carlo sampling retains certain advantages, the most important of which is a measure of confidence that enough samples have been taken for the given problem. In other words, there is no convergence test possible in descriptive sampling. Furthermore, Monte Carlo sampling is more readily cumulative (repeated runs can be combined for more accuracy than can repeated runs of descriptive sampling), and it can yield a more accurate prediction of the overall reliability than can descriptive sampling.

5.16.4 Reliability Estimation

5.16.4.1 Based on Sampling

When a design is sampled, whether by Monte Carlo or descriptive methods, the program collects statistics on the desired responses (as specified by the reliability constraints). These statistics include a simply tally of the number of times a reliability constraint was violated, in addition to a mean and standard deviation for the underlying response.

The tallies can be used directly as probabilities by dividing them by the total number of counts. In fact, if parallel limits are not used, then the overall reliability can be estimated as the total number of samples that didn't violate *any* constraints divided by the number of samples. (In the limit of a single constraint with only an upper or only a lower limit, the overall reliability is the same as the reliability for that constraint.)

An alternate method of estimating reliability is to assume a normal (Gaussian) distribution for the responses. Then, using the mean and standard deviations that have been independently calculated while the tallies were being performed, the probability of exceeding any limit can be calculated using the assumed profile.

Both estimation methods are available in SINDA/FLUINT when *either* sampling technique is used.

5.16.4.2 Based on Gradients

A method for estimating reliability is available that is even faster than DSAMPLE, but has even more limitations: RELEST. This technique is not a sampling technique at all. Rather, it estimates reliability by measuring gradients in the responses with respect to the random variables, and by assuming (but not requiring) that all distributions (both input and response) are normal (Gaussian). It further assumes that the mean of the responses can be predicted using the mean values of the random variables, and that response variations from that point are linear with respect to changes in inputs.

RELEST requires only N+1 evaluations, where N is the number of random variables. This is often an order of magnitude smaller than what DSAMPLE requires, which is an order of magnitude smaller than SAMPLE requires: RELEST is comparatively cheap to call.

The first evaluation uses the mean values of random variables, and assumes that the resulting responses are the means of those functions. The next (and final) N evaluations perturb each random variable (in input order) such that the gradients of each response with respect to each input variable can be estimated using finite differences. RELEST then assumes a first order Taylor series of variance (the square of standard deviation) can be applied to estimate the variance (and therefore standard deviation) of each response given the variance of each random variable, whether those variables are normal or not. Now the code has enough information to predict reliabilities: it has an estimate for the mean and standard deviation of each response, and can therefore predict the likelihood that a response will assume any given value.

RELEST cannot predict overall reliability much less the tallied estimate of reliability that a sampling routine can, and should be used with caution in cases with nonlinear responses and non-normal random variables. It also cannot handle variable failure limits. Furthermore, unlike sampling techniques, the accuracy of RELEST is not cumulative: repeated calls do not affect the accuracy of the results. However, because it is so inexpensive, RELEST is often plays an important role in robust design (reliability optimization, Section 5.27).

5.16.4.3 Foresight vs. Hindsight

The above discussion would imply that the user must know all possible failure limits and desired responses, and must define them in advance to use the reliability engineering module.This would be a terrible restriction given the typically high cost of performing a statistical analysis. Therefore, the user should note that postprocessing tools are available that allow the information gathered during a SAMPLE, DSAMPLE, or RELEST run to be used to investigate other responses or even other limits on existing responses.

5.16.5 Usage Overview

Reliability Engineering consists of several high-level "solution routines" that invoke an arbitrary analysis procedure to evaluate a design or model.

In its simplest form this reliability evaluation procedure, input in a header block called RELPROCEDURE (Section 5.22), might consist of a single call for a steady state solution ("CALL STEADY"). However, the procedure can be an arbitrarily complex process with one or more steady state or transient runs, complete with BUILD/BUILDF commands, internal loops or iterations, and internal parametric restarts or other I/O operations. In fact, the only distinction between RELPROCEDURE and OPERATIONS is that the logical instructions and analysis sequences in RELPROCEDURE are normally invoked many times, whereas OPERATIONS remains a one-pass set of instructions (i.e., executed once). The call to SAMPLE, DSAMPLE, or RELEST is made in OPERATIONS, and these routines then internally call RELPROCEDURE any time it needs to *evaluate* the user's design or model. Using the Reliability Engineering module is therefore like setting up an automatic sequence of SINDA/FLUINT runs, wherein each single run is now a self-contained procedure.

What is meant by the term "evaluate a design or model" is ultimately controlled by the user according to reliability constraints supplied: what analysis operations are required to update the reliability constraints? The evaluation procedure starts with specific values for the random variables, and must (before it returns) update the current values of the response functions, as specified by the reliability constraints. This evaluation process might therefore simply be the calculation of a single steady-state followed by the calculation of the current reliability constraints (if they have not been defined via expressions and therefore automatically updated).

To use the Reliability Engineering module, as a minimum the user:

1. Builds a model that can be varied using registers (Section 4.9) as needed to make the model respond to changes in those registers.

   Although it is a good idea to use registers and register-containing expressions extensively throughout a model, it is also possible to retroactively add registers (perhaps as multiplying factors initialized to unity).

2. Chooses a subset of the registers that will participate in the solution. These are the *random variables*: one or more uncertain parameters that will be perturbed by the reliability routines during their operation. These special registers and their probability distribution functions are specified within RANDOM DATA (Section 5.17).

3. Defines the desired responses and failure limits. The user should identify the threshold temperatures, pressures, or other criteria that distinguish a failed case from a successful case. These are defined in RELCONSTRAINTS DATA (Section 5.18).

4. Defines the solution procedure by which the module should evaluate a design to calculate desired responses (defined above in step #3) for the current design instance (as dictated by the current program-selected set of values of random variables defined above in step #2) against the goal. This evaluation procedure is defined in HEADER RELPROCEDURE (Section 5.22), a logic block similar to OPERATIONS.

6. Invokes a sampling technique or reliability estimation technique in OPERATIONS by calling SAMPLE (Section 5.19), DSAMPLE (Section 5.20), or RELEST (Section 5.21), adding perhaps a call to RANTAB and/or RCSTTAB (all described in Section 7, with an overview in Section 5.24) afterwards to print final values of the reliabilities and other run information.

Actually, it is only strictly necessary to define reliability constraints when using RELEST, and even in that case hindsight may be applied using postprocessing tools. However, forethought in defining these failure limits will result in the most productive use of the code.

Many more tools exist, such as the ability to make cumulative runs, or to nest reliability estimations within the Solver's evaluation procedure (PROCEDURE, analogous to but distinct from Reliability Engineering's RELPROCEDURE). These additional tools are documented in the rest of this section.

5.16.6 A Brief Example

The operation of the Reliability Engineering module may at first appear complex because it involves many options and input blocks. However, it can be quite simple to employ. As an example, assume that a user needed to find the reliability of a stainless steel pipe of a given diameter and length that transports liquid propane, where the reliability is defined as the probability that the pressure drop will not exceed some threshold such as 10kPa, perhaps as needed to prevent the liquid from flashing. The upstream conditions are known, but the actual diameter can vary due to manufacturing tolerances, as can the length due to variations during installation.

The user would start by constructing a SINDA/FLUINT model of the pipe using registers to define the diameter and length, and perhaps other key parameters as well. The diameter and length, perhaps called DIAM and LENGTH, would then be declared as a random variable in RANDOM DATA.* Their degree of uncertainty would also be defined, perhaps as a fixed range of equal probabilities (high and low values), a Gaussian distribution (mean and standard deviation), or some more complex function. Determining these distributions is often the most difficult part of using this module, requiring perhaps tests or other investigations.

* Alternatively, one could define DIAM as the nominal value and define DTOL as the tolerance in the diameter, where in FLOW DATA DH=DIAM+DTOL, DIAM is fixed, and DTOL is a random variable whose mean is zero.

An expression defining the reliability constraint (the allowed pressure drop in the pipe) would be defined in RELCONSTRAINT DATA. If, for example, the difference between the pressure of the upstream lump #10 and the pressure of the downstream lump #20 in submodel PIPE should not exceed 10kPa, then the expression might appear as "pipe.pl20 <= pipe.pl10 + 10000.0" or preferably* "pipe.pl20-pipe.pl10 <= 10000."

The user would then BUILD and BUILDF the appropriate submodels in OPERATIONS, and make a call to a reliability estimation routine (SAMPLE, DSAMPLE, or RELEST) followed perhaps by a call to the output routine RCSTTAB. The evaluation procedure supplied in HEADER REL-PROCEDURE would then be a call to a steady state solution ("CALL STEADY").

Given that the default parameters in SOLVER DATA were appropriate for this problem, SINDA/FLUINT would then internally call RELPROCEDURE iteratively as needed to estimate the probability that the candidate design does not violate the constraint. Although the user may have required a call to STEADY to calculate the current pressure drop, from the perspective of the Reliability Engineering module the only data that was relevant was the returned values of pipe.pl10 and pipe.pl20, since those are the values required to calculate the reliability.

* The first example is a variable (relative) failure limit, the second is a fixed (absolute) failure limit. Fixed limits are preferred because they enable the use of the fast routine RELEST, and also because they provide additional data in SAMPLE and DSAMPLE: estimated reliability based on presumed normal response distributions.

5.17 HEADER RANDOM DATA

Of the registers defined in REGISTER DATA (Section 2.8), some or all may be selected to act as *random variables*. If the Reliability Engineering module is to be used, at least *one* random variable must be selected. These random variables will be automatically perturbed (within their defined ranges) by the Reliability Engineering module as needed to determine the distributions of any defined response variables and therefore to estimate the probability that any failure limits will be violated, using internal calls to UPREG and RELPROC (Section 5.22). (Responses and failure limits are defined as *reliability constraints*, described in Section 5.18.) Therefore, *the user should not independently modify the random variables while the Reliability Engineering module is executing.*[*]

In the current version, only real registers may be used as random variables, and only continuous values are supported. There are three types of distributions accepted:

UNIFORM . . . A flat distribution between a lower and upper value defining the range. Any value of the random variable is equally probably within this range, but the random variable will never exceed this range.

NORMAL. . . . A normal or Gaussian distribution defined by a mean value (μ) and either a standard deviation (σ) or a coefficient of variation (σ/μ). Values near the mean are most probably, but a finite possibility exists that any value from negative to positive infinity can occur. Realistically, a value below μ-6σ or above μ+6σ will probably never occur.

ARRAY . . . . . An arbitrary distribution defined by a table of probabilities versus value. This table can be used to input almost any type of distribution, including a truncated (limited) normal distribution, and data from test or manufacture.

The format is:

```
HEADER RANDOM DATA
       reg_name, UNIFORM, lower_limit, upper_limit
       reg_name, NORMAL [,mean=R] [,sd=R or cv=R]
       reg_name, ARRAY, array_ref
```

where:

reg_name . . . . name of valid real register to be used as a random variable

lower_limit . . . smallest value for UNIFORM variables

upper_limit . . . largest value for UNIFORM variables (upper_limit must be greater than or equal to lower_limit)

mean . . . . . the mean value (μ) for NORMAL variables (defaults to the *initial* value defined in REGISTER DATA, but will not change later during execution if register is changed)

sd. . . . . the standard deviation (σ) for NORMAL variables

cv . . . . . the coefficient of variation (σ/μ) for NORMAL variables

* Random variables can, however, be changed *between* calls to the estimation routines.

array_ref. . . . . a complete SINDA/FLUINT array reference including submodel prefix,* such as “smn.a100.” This real bivariate array should contain a table of relative frequencies (probabilities) versus random variable value: V1, P1, V2, P2 ... etc. where V is monotonically increasing. The first and last values of V become the limits on the variable: no extrapolation past the defined range is performed, although the probabilities at the end points need not be zero.

Only one random variable and its distribution description may be entered per line.

Either SD or CV can be input, but not both, and one of them is required. Do not use CV to define a distribution for a NORMAL variable whose mean is or can be zero: use SD instead in such cases.

Example:

```
HEADER RANDOM DATA
       fred, normal, sd = 0.5
       wilma, array, flint.a404
       barney, uniform, -0.1, 0.1
```

For example, if the emissivity, the absorptivity, and the incident flux on a surface were uncertain, and if they were defined as registers (say: EMIS and ALPHA and FLUX), then they could also be defined as random variables. Perhaps the emissivity may be any value between 0.8 and 1.0, while test data defines a triangular probability for the absorptivity: zero probability below 0.03 or above 1.0, with a peak near 0.2. This data is placed in array #400 of submodel RADIATOR. Finally, the variation in flux is Gaussian, with a mean of 1400.0 and a standard deviation of 58. This could be expressed as follows:

```
header random data
       emis, uniform, 0.8, 1.0
       alpha, array, radiator.a400
       flux, normal, mean=1400., sd=58.
header array data, radiator
       400    =       0.03,   0.0
                      0.2,    1.0
                      1.0,    0.0
```

The RANTAB routine (Section 7.13) provides a convenient method for listing the random variables and describing their distributions for input verification purposes. This data includes mean and standard deviations for all types of variables.

Complex expressions may be used to define the limits on UNIFORM variables, the mean and standard deviation (or coefficient of variation) of NORMAL variables, or even the elements within arrays used to define ARRAY variables. However, these expressions should be used with caution.

* There is no equivalent of DEFMOD in RANDOM DATA: submodel names must be used explicitly.

It is legal to change the distribution of a random variable ***before or between*** *calls to reliability estimation routines (SAMPLE, DSAMPLE, RELEST), but it is illegal to change them* ***during*** *such a run*. The code will expect constant distributions, but will not check to assure they have not changed.

The units of probability are irrelevant, and this is why it is sometimes called *relative* frequency. This is especially relevant to ARRAY variables: any value of probability* can be used to define the *shape* of the profile. However, probabilities cannot be less than zero. They need not be nonzero at the ends of the profile (the lowest and highest values of random variables at the beginning and end of the array, which form the fixed limits of variation), and they can become zero within an array.

When describing a complex distribution for which a mathematical formula exists, a preliminary run might be made in which the distribution is printed to a user output file (comma delimited). This file can then be INSERTed as an array in a later run. For example, assume a truncated normal distribution is required to avoid excessively high and low values. To be specific, assume the variable's mean is 50 (stored in MEAN), with a standard deviation of 10 (stored in SDEV), but values above 100 or below 0 must be excluded to avoid numerical problems or impossible values. Assuming the value of π has been brought into the processor using the REGISTER DATA statement "PIE=PI," the logic to create such an array might be:

```
        do 10 vtest = 0.0, 100.0, 1.0
              ytest   = -0.5*((vtest-mean)/sdev)**2
              ptest   = exp(ytest)/(sdev*sqrt(2*pie))
10            write(nuser1,*) vtest,',',ptest
```

The above logic will create a bivariate array input breaking the normal distribution into 100 points, with cut-offs below 0 and above 100. More resolution could be used if needed.

To temporarily turn off a UNIFORM variable within the processor (but not during the execution of a reliability routine), the upper and lower limits may be set equal. Similarly, a NORMAL variable may be temporarily turned off by setting its SD or CV (depending on which was used to originally define it) to zero.

Caution on Using Non-NORMAL Responses: ***RELEST assumes NORMAL variables.*** Refer to Section 5.21 for more information.

* Within reason: values too high ("1.0E20") may cause numerical problems.

5.18 HEADER RELCONSTRAINT DATA

In order to calculate reliability, the Reliability Engineering module must first know which responses are critical, and what limits on those responses determine success or failure. Responses and their failure limits are collectively referred to as *reliability constraints*. While many reliability constraints can be specified as expressions of registers and processor variables, more complex constraints requiring logical operations, function calls, etc. can be defined using arbitrarily-named *reliability constraint variables.*

If used, these reliability constraint variables are real (noninteger) values that are intended to be updated during or near the end of the RELPROCEDURE (Section 5.22). The values of the reliability constraint variables may therefore be arbitrarily complex, based on system parameters (temperatures, pressures, etc.) or derived data. The user may optionally define an upper and/or a lower limit on these reliability constraint variables. These limits determine the threshold for success or failure, and may be themselves defined by registers or expressions, including expressions containing random variables and other registers.

The basic format is similar (*but not identical!*) to CONSTRAINT DATA as described in Section 5.6. The one important difference is that a reliability constraint can be defined as a response to be tracked *without specifying a limit*. The format is:

```
HEADER RELCONSTRAINT DATA
       [ lower_limit <=] rcst_name [<= upper_limit]
       [ lower_limit <=] expression [<= upper_limit]
```

where:

lower_limit. . . . optional lower failure limit or expression

rcst_name. . . . . Response to track. Unique name (8 characters or less) for a real reliability constraint; *cannot be a register name (Section 2.8), nor an optimization constraint name, nor a global user data name (Section 2.9) nor any reserved word (Section 6.18.5). In logic block references, all reliability constraint variables are of type real, independent of their names.*

upper_limit. . . . optional upper failure limit or expression

expression . . . . Response to track. An expression based on registers and/or processor variables, effectively replacing a reliability constraint variable (and therefore called an “unnamed reliability constraint”). Cannot be a register alone: use “(register)” or “1*register” to avoid an apparent conflict between a reliability constraint name and a register name.

Only one reliability constraint and its limits may be entered per line. Limit definitions may include expressions containing registers or processor variables. If a named reliability constraint is used, then the selected name must be unique, and the value for that named constraint must be updated by the user by the end of RELPROCEDURE (“RELPROC”). If both limits are defined, the upper and lower limit may be equal, but the lower limit cannot exceed the upper limit.

Example:

```
HEADER RELCONSTRAINT DATA
       0.01 <= deltaP         $ "deltaP" is a rel. const. var.
       -20.0 <= 0.5*(rod.T15+rod.T20) <= 40.0
       0.0 <= mass <= 10.0
       checkit         $ no limits, just need mean and std. dev.
       side1 <= 0.5*thick             $ OK but ...
       0.0 <= side1 - 0.5*thick       $ this is preferred
       -pi <= angle <= pi
       flow.fr20 <= flow.fr30         $ OK but ...
       flow.fr20 - flow.fr30 <= 0.0   $ this is preferred
```

For example of how to use a reliability constraint variable, consider a system in which the pressure drop were not allowed to exceed 10,000 Pa. To accommodate this constraint, a constraint variable named "deltaP" could be created in RELCONSTRAINT DATA:

```
deltaP <= 10000.0
```

At the end of HEADER RELPROCEDURE, a single calculation might be added to update the *current* value of deltaP, where lumps 10 and 23 define the pressure drop to be constrained:

```
deltaP = pl10 - pl23
```

Alternatively, the above constraint can be imposed directly in RELCONSTRAINT DATA with a single statement that avoids the creation of a reliability constraint variable that must be updated later. The following impose the above constraint (assume the lumps are in submodel* "water"):

```
water.pl10 <= water.pl23 + 10000      $ acceptable
water.pl10 - water.pl23 <= 10000      $ preferred†
```

The calculation of the reliability constraint variable may be complex, perhaps performed cumulatively during the course of the RELPROCEDURE solution. For example, if a certain temperatures in the system should never drop below 100 degrees during the course of a transient, this could be achieved by defining:

```
100 <= Tmin
```

in RELCONSTRAINT DATA, setting Tmin to be huge (say, 1.0E30) at the start of RELPROCEDURE, and then adding the following line in the appropriate VARIABLES 2 blocks for whichever nodes are of interest:

```
Tmin    = min(Tmin,T101,T304,T504)
```

* There is no equivalent of DEFMOD in RELCONSTRAINT DATA: submodel names must be used explicitly.
† Refer to cautions regarding variable (relative) failure limits later in this subsection.

In the current version, reliability constraint variables are all real, and only continuous values are supported.

Upper and lower limits on reliability constraints may be set equal if desired, but no more information is returned than if a single limit had been placed.

The Reliability Engineering module must have a recent value of the reliability constraint variable in order to compare it with the limits, or to develop statistics that describe its distribution. ***Reliability constraints do not prevent failed designs from being sampled during the course of the reliability estimation. The underlying model must be robust enough to handle any possible combination of random variable values, including those that result in failed cases.***

Multiple limits on the same constraint may be input, if for example the user desires to know the reliability of the design at various levels of margin. However, if the reliability constraint is named, a new name must be used each time and all such named variables must be updated by the end of RELPROCEDURE. The same restriction does not apply to unnamed constraints. For example:

```
left.t401 <= 100.0
left.t401 <= 90.0
left.t401 <= 80.0
```

Guidance: The is no equivalent of DEFMOD in RELCONSTRAINT DATA. Submodel prefixes must be used explicitly.

A “bare” register (that is, a register that is not part of an expression such as “X1”) may cause apparent conflicts since the program cannot allow a reliability constraint variable to be defined that conflicts with a register name. Thus, the following is illegal:

```
HEADER RELCONSTRAINT DATA
C ILLEGAL IF X1 IS A REGISTER
       X1 <= 1
```

To make sure the program can discern an expression from the declaration of a reliability constraint name, the user must “assure” the program that an expression is being specified by multiplying by 1 or by enclosing the register name in parentheses:

```
HEADER RELCONSTRAINT DATA
       (X1) <= 1
       1*X1 <= 1
```

The RCSTTAB routine (Section 7.14) provides a convenient method for reporting the current status or final values of the reliability constraint variables, along with statistics describing the response and the reliabilities: the probabilities that a failure limit will not be exceeded. ***RCSTTAB is the main output for the Reliability Engineering module, and should normally immediately follow a call to SAMPLE, DSAMPLE, or RELEST.*** Note that reliability constraints are classified in that output routine on the basis of whether they were explicitly named or whether they simply consist of expressions containing registers and/or processor variables (in which case they are termed “unnamed”).

The RCGET and RCGETNO routines (Section 5.24.2) can alternatively be used to dynamically fetch the results of the RCSTTAB routine in case those results are needed within logic.

Caution on Using Variable Failure Limits: When possible, it is preferable to state failure limits ("upper_limit" and "lower_limit") as fixed quantities instead of expressions that can vary during the reliability estimation routine. For example, ***RELEST cannot tolerate variable limits***. SAMPLE and DSAMPLE *can* handle them, but they must then restrict the reliability predictions to tallies. Refer to the estimation routines for more information.

Caution on Using Discontinuous Responses: ***RELEST cannot tolerate discontinuous responses***, and linear responses are assumed in that routine. Refer to Section 5.21 for more information.

5.19 SAMPLE Routine

There are three main drivers for the Reliability Engineering module: SAMPLE, DSAMPLE, and RELEST. This section describes SAMPLE along with a general discussion of other features that will not be repeated in the subsequent sections on DSAMPLE (Section 5.20) and RELEST (Section 5.21). Table 5-6 is a summary of the options available.

Table 5-6 Comparison of Reliability Estimation Routines

	SAMPLE	DSAMPLE	RELEST
Method	Monte Carlo sampling	Descriptive sampling	Gradient method
Speed	Slow	Intermediate	Fast
Convergence Detected?	Yes	No	No
Fixed Execution Cost?	No	Yes	Yes
Overall Reliability?	Yes	Yes	No
Cumulative?	Yes	Somewhat	No
Applicability?	Unlimited	Unlimited	Limited. Assumes: - NORMAL variables - Linear responses - Continuous responses - Fixed failure limits
Incomplete Runs Tolerated?	Yes	No	No
Notes		Can't change NLOOPR while executing.	NLOOPR .GE. N+1, where N is the number of random variables.

SAMPLE uses Monte Carlo random sampling, as described in Section 5.16.3.

Like the other Reliability Engineering main drivers, *SAMPLE may only be called from OPERATIONS or from PROCEDURE, but not from RELPROCEDURE.* Also like other drivers, SAMPLE internally calls an evaluation procedure defined by the user separately in RELPROCEDURE (Section 5.22) according to the control parameters defined in SOLVER DATA (Section 5.4). SAMPLE randomly perturbs *random variables* defined in RANDOM DATA (Section 5.17) while tracking the responses and failure limits (*reliability constraints)* defined in RELCONSTRAINT DATA (Section 5.18).

This sampling continues until either NLOOPR calls to RELPROCEDURE have been made, or until the defined responses and their associated reliabilities converge. Convergence is defined by the mean and standard deviation of all responses as well as the tallied individual and overall reliabilities not changing by more than RERRR (relative check, 0.1% by default.) and/or AERRR (absolute check for small values) between consecutive RELPROCEDURE calls. RERRR and AERRR are specified in logic or in SOLVER DATA (Section 5.4). If convergence is detected, SAMPLE will stop. Otherwise it will take up to NLOOPR but will not issue nonconvergence warnings.

Just as the instructions in OPERATIONS are converted to a Fortran (or C) routine named OPER, the instructions in RELPROCEDURE are converted into a routine named RELPROC. SAMPLE changes all random variables, then propagates the changes throughout the rest of the model via an internal call to UPREG (Section 7.12.1),* and then calls for a new evaluation via a call to RELPROC

to get new values of reliability constraint variables and expressions (unnamed variables). SAMPLE keeps a running tally of failed cases along with statistics (mean and standard deviation) describing each reliability constraint.

An additional global logic block is available that is analogous to the submodel-dependent OUTPUT CALLS blocks. This block, RELOUTPUT CALLS, is described in Section 5.19.1. It is converted in to RELOUT.

In pseudo-code, a simplified rendition of the routine SAMPLE is as follows:

```
SUBROUTINE SAMPLE
[initialization: reset if cumulative not detected]
[check for changes in random variable distributions]
[for a maximum of NLOOPR iterations:]
       [for each random variable:]
              [select random value per distribution]
       CALL UPREG     $ update registers and formulas
       CALL RELPROC   $ call for a new evaluation:
                      $ new reliability constraints
       CALL UPREG     $ update registers and formulas
       [for each reliability constraint:]
              [tally successes or failures]
              [update mean and standard deviation]
              [watch for convergence per RERRR/AERRR]
       [tally overall reliability]
       CALL RELOUT
       [if converged, return]
[return]
```

In many ways, each SAMPLE step is analogous to a single steady-state relaxation step. Of course, since RELPROCEDURE can be as arbitrarily complex as a separate SINDA/FLUINT run, SAMPLE clearly works at a higher level than a traditional steady-state or transient solution routine.

Example:

```
HEADER OPERATIONS
BUILD ALL
       CALL RANTAB
       CALL SAMPLE
       CALL RCSTTAB
```

* Unlike other automatic internal calls to UPREG, calls from within SAMPLE cannot be turned off using the NOUPDATE feature in OPTIONS DATA, though they *will* comply with any disconnections performed using the DEREG family of routines (Section 7). However, NOUPDATE *does* apply to UPREG calls made within solution routines, and these are normally called from within SAMPLE (see the RELPROCEDURE block).

Guidance on NLOOPR: SAMPLE is a complete but expensive routine. NLOOPR is typically on the order of 1000 or more.

Cautions on Using Overall Reliability: OVEREL, which is printed by RCSTTAB and is available in logic or expressions, is the probability that no failure limit will be violated. If all reliability constraints are independent, then this corresponds to the overall reliability. However, if multiple reliability constraints are placed in parallel (failure constitutes exceeding all and not just one limit) then OVEREL will underestimate overall reliability.

Cautions on Using Variable Failure Limits: Often, a failure limit (the upper and/or lower limit on a response) does not change during the course of a SAMPLE run. For example, a temperature may be specified not to exceed 100 degrees. However, if the limit is relative (e.g., "do not exceed 10 degrees more than an average temperature") or can otherwise change during a SAMPLE run, then the limit is variable. If the limit is variable, *use only tallied reliabilities and not the reliabilities calculated assuming a Gaussian (normal) response.*

Guidance on Cumulative Runs: If SAMPLE is call more than once per run (with no intervening call to DSAMPLE or RELEST), then the results are by default cumulative. Similarly, if a RESTDB call is made (Section 5.23), results may be cumulative. This allows a small run (small NLOOPR) to be made to make sure the problem is set up correctly, and then later runs to continue reusing the results of former runs. To avoid accumulation, use the RESAMP routine (Section 5.24). If making cumulative runs, do not change the distribution functions (any RANDOM DATA inputs) between calls to SAMPLE.

Repeatable Runs: If for some reason a run needs to be completely repeatable, the user can set the NSEED (Section 5.4) value to either be the same as a previously reported run, or perhaps a user-selected large (>10000) integer. An 8 digit limit applies. To exceed this limit (subject to a maximum value of 2^{31}), initialize NSEED at the beginning of OPERATIONS, but use an F in column 1 to turn off translation. For example:

```
F       NSEED   = 142843216
```

5.19.1 RELOUTPUT CALLS

A global logic block is available that is analogous to the submodel-dependent OUTPUT CALLS blocks. Like those blocks, output operations (particularly RCSTTAB, as described in Section 7.13) are commonly placed in the HEADER RELOUTPUT block, but few restrictions exist as to what can be placed in this block.

Since this block is global (i.e., not dependent on any submodel), submodel prefixes and/or the DEFMOD command must be used when referring to any submodel-dependent keyword.

RELOUTPUT CALLS is converted into a routine named RELOUT.

RELOUT is called by SAMPLE, by DSAMPLE (Section 5.20), and by RELEST (Section 5.21) upon the completion of each iteration. Between the end of RELPROC and the start of RELOUT, reliabilities and response statistics have been calculated by SAMPLE and by DSAMPLE (but not

by RELEST). Therefore, RELOUT is a good place to track progress using RCSTTAB or perhaps calls to RCGET or RCGETNO for SAMPLE and DSAMPLE. However, it is specifically designed to be a suitable location for calling SAVEDB in any Reliability Engineering module driver, generating a database containing sampled runs.

5.20 DSAMPLE Routine

DSAMPLE uses descriptive sampling, as described in Section 5.16.3.

Like the other main Reliability Engineering drivers, *DSAMPLE may only be called from OPERATIONS or from PROCEDURE, but not from RELPROCEDURE*. DSAMPLE internally calls an evaluation procedure defined by the user separately in RELPROCEDURE (Section 5.22) according to the control parameters defined in SOLVER DATA (Section 5.4). DSAMPLE perturbs *random variables* defined in RANDOM DATA (Section 5.17) while tracking the responses and failure limits (*reliability constraints)* defined in RELCONSTRAINT DATA (Section 5.18).

DSAMPLE always performs exactly NLOOPR calls to RELPROCEDURE (which is turned into subroutine “RELPROC”). Unlike SAMPLE, convergence is not detected. In fact, if for some reason DSAMPLE is unable to complete the specified NLOOPR samples, intermediate results cannot be reliably used.

DSAMPLE changes all random variables, then propagates the changes throughout the rest of the model via an internal call to UPREG (Section 7.12.1),* and then calls for a new evaluation via a call to RELPROC to get new values of reliability constraint variables and expressions (unnamed variables). DSAMPLE keeps a running tally of failed cases along with statistics (mean and standard deviation) describing each reliability constraint, but until it completes the planned NLOOPR samples, these intermediate results are inaccurate.

An additional global logic block is available that is analogous to the submodel-dependent OUTPUT CALLS blocks. This block, RELOUTPUT CALLS, is described in Section 5.19.1. It is converted in to RELOUT. It represents a location for calling SAVEDB, which stores results for use in postprocessing or restarts.

In pseudo-code, a simplified rendition of the routine DSAMPLE is as follows:

```
SUBROUTINE DSAMPLE
[initialization: reset if cumulative not detected]
[check for changes in random variable distributions]
[determine variable values: discretize distributions]
```

* Unlike other automatic internal calls to UPREG, calls from within SAMPLE cannot be turned off using the NOUPDATE feature in OPTIONS DATA, though they *will* comply with any disconnections performed using the DEREG family of routines (Section 7). However, NOUPDATE *does* apply to UPREG calls made within solution routines, and these are normally called from within SAMPLE (see the RELPROCEDURE block).

```
[for a total of NLOOPR iterations:]
        [for each random variable:]
                [select value from discretized list]
        CALL UPREG      $ update registers and formulas
        CALL RELPROC    $ call for a new evaluation:
                        $ new reliability constraints
        CALL UPREG      $ update registers and formulas
        [for each reliability constraint:]
                [tally successes or failures]
                [update mean and standard deviation]
        [tally overall reliability]
        CALL RELOUT
[return]
```

Since RELPROCEDURE can be as arbitrarily complex as a separate SINDA/FLUINT run, DSAMPLE works at a higher level than a traditional steady-state or transient solution routine.

Example:

```
HEADER OPERATIONS
BUILD ALL
        CALL RANTAB
        CALL DSAMPLE
        CALL RCSTTAB
```

Guidance on NLOOPR: DSAMPLE takes roughly 10-20% of the number of steps that SAMPLE takes to achieve the same level of accuracy, although this number is problem dependent. Set NLOOPR to the maximum number of evaluations that can be tolerated, but expect to use at about 100 or more evaluations. (NLOOPR defaults to 100.)

Cautions on Using Overall Reliability: OVEREL, which is printed by RCSTTAB and is available in logic or expressions, is the probability that no failure limit will be violated. If all reliability constraints are independent, then this corresponds to the overall reliability. However, if multiple reliability constraints are placed in parallel (failure constitutes exceeding all but not one limit) then OVEREL will underestimate overall reliability. DSAMPLE is not as accurate as is SAMPLE in predicting overall reliability for more than one failure limit.

Cautions on Using Variable Failure Limits: Often, a failure limit (the upper and/or lower limit on a response) does not change during the course of a DSAMPLE run. For example, a temperature may be specified not to exceed 100 degrees. However, if the limit is relative (e.g., “do not exceed 10 degrees more than an average”) or can otherwise change during a DSAMPLE run, then the limit is variable. If the limit is variable, *use only tallied reliabilities and not the reliabilities calculated assuming a Gaussian (normal) response.*

Guidance on Cumulative Runs: If DSAMPLE is call more than once per run (with no intervening call to SAMPLE or RELEST), then the results are by default cumulative. Similarly, if a restart is made (Section 5.23), results will be cumulative. This allows a small run (small NLOOPR) to be made to make sure the problem is set up correctly, and then later runs to continue reusing the results of former runs. To avoid accumulation, use the RESAMP routine (Section 5.24). If making cumulative runs, do not change the distribution functions (any RANDOM DATA inputs) between calls to SAMPLE.

Caution on Cumulative Runs: Combining two consecutive NLOOPR=100 samples in SAMPLE is completely equivalent to a single run with NLOOPR=200. ***This is not true for DSAMPLE:*** a single NLOOPR=200 run will be much more accurate than two NLOOPR=100 runs, despite the fact that the code will combine them. The NLOOPR=200 run used twice the resolution in the distribution functions of the random variables, whereas a combination of NLOOPR=100 runs is effectively an average between them, with only slightly better results than a single NLOOPR=100 run. In fact, if there is only one random variable *no* increase in accuracy results from cumulative DSAMPLE runs.

Despite the fact that they use the same resolution, any two consecutive DSAMPLE runs will not yield the exact same results since they used a different combination of each random variable. Although the *values* used in DSAMPLE are not random, the *order* and the *combination* of these values will be random. The order in which values are sampled ideally shouldn't matter, but sometimes does due to secondary factors such as slight order dependence in underlying steady state solutions. The combination of values will definitely matter if more than one random variable is applied.

Repeatable Runs: If for some reason a run needs to be completely repeatable, the user can set the NSEED (Section 5.4) value to either be the same as a previously reported run, or perhaps a user-selected large (>10000) integer. An 8 digit limit applies. To exceed this limit (subject to a maximum value of 2^{31}), initialize NSEED at the beginning of OPERATIONS, but use an F in column 1 to turn off translation. For example:

```
F       NSEED   = 381301923
```

5.21 RELEST Routine

RELEST uses approximations and gradient methods, as described in Section 5.16.4.2. The methods used in RELEST are completely different from those used in SAMPLE (Section 5.19) and DSAMPLE (Section 5.20).

Otherwise, like the other main Reliability Engineering drivers, *RELEST may only be called from OPERATIONS or from PROCEDURE, but not from RELPROCEDURE.* RELEST internally calls an evaluation procedure defined by the user separately in RELPROCEDURE (which is turned into subroutine "RELPROC," as described in Section 5.22) according to the control parameters defined in SOLVER DATA (Section 5.4). RELEST perturbs *random variables* defined in RANDOM DATA (Section 5.17) while tracking the responses and failure limits (*reliability constraints)* defined in RELCONSTRAINT DATA (Section 5.18).

RELEST always performs exactly N+1 evaluations, where N is the number of random variables. NLOOPR must be greater than or equal to this value or the call to RELEST will fail.

The first call that RELEST makes to RELPROC uses the mean values of all random variables. This call is used both to estimate the mean values of all responses and to set the baseline case for the gradient calculations that follow. The subsequent N calls (where N is the number of random variables) perturb one random variable at a time (in input order) according to RDERR and ADERR (Section 5.4), leaving all other random variables at their mean value.

When RELEST changes random variables, it propagates the changes throughout the rest of the model via an internal call to UPREG (Section 7.12.1),* and then calls for a new evaluation via a call to RELPROC to get new values of reliability constraint variables and expressions (unnamed variables). Unlike SAMPLE and DSAMPLE, no running tallies nor partially accumulated results are available until RELEST has completed its run.

Therefore, when RELOUT (the results of RELOUTPUT CALLS, see Section 5.19.1) is called from within RELEST, calls to RCSTTAB, RCGET, or RCGETNO in that location will not return useful results. Rather, a call to SAVEDB can be made in RELOUT to support postprocessing estimations of responses not originally listed in RELCONSTRAINTS. Refer to Section 5.16.4.3 and Section 5.23.

In pseudo-code, a simplified rendition of the routine RELEST is as follows:

```
SUBROUTINE RELEST
[initialization: reset]
[check for changes in random variable distributions]
[set random variables to their mean values]
```

* Unlike other automatic internal calls to UPREG, calls from within SAMPLE cannot be turned off using the NOUPDATE feature in OPTIONS DATA, though they *will* comply with any disconnections performed using the DEREG family of routines (Section 7). However, NOUPDATE *does* apply to UPREG calls made within solution routines, and these are normally called from within SAMPLE (see the RELPROCEDURE block).

```
CALL UPREG     $ update registers and formulas
CALL RELPROC   $ call for a new evaluation:
               $ new reliability constraints
CALL UPREG     $ update registers and formulas
[set the mean of responses to the returned values]
[for each random variable in input order:]
       [perturb according to RDERR/ADERR]
       CALL UPREG     $ update registers and formulas
       CALL RELPROC   $ call for a new evaluation:
                      $ new reliability constraints
       CALL UPREG     $ update registers and formulas
       CALL RELOUT
       [for each reliability constraint:]
              [calculate and store gradients]
[for each reliability constraint:]
       [estimate standard deviation of responses]
[return]
```

Since RELPROCEDURE can be as arbitrarily complex as a separate SINDA/FLUINT run, RELEST works at a higher level than a traditional steady-state or transient solution routine.

Example:

```
HEADER OPERATIONS
BUILD ALL
       CALL RANTAB
       CALL RELEST
       CALL RCSTTAB
```

Assumptions: RELEST employs the following assumptions:

1. All random variables are normal (Gaussian): their distributions can be entirely estimated based on their mean and standard deviation. This does not preclude RELEST from being used for other distributions, but it does adversely affect the accuracy of the resulting predictions in such cases.
2. All responses are linear with respect to changes in random variables. Again, this does not preclude RELEST from being used in nonlinear systems, but does adversely affect the accuracy of the resulting predictions in such cases.
3. (Implicit in the above two assumptions:) All responses are normal (Gaussian).
4. All responses are continuous, and their derivatives with respect to random variables are similarly continuous.

Restrictions on Using Overall Reliability and Tallied Reliabilities: OVEREL, which is printed by RCSTTAB and is available in logic or expressions, is the probability that no failure limit will be violated. ***RELEST does not calculate OVEREL***, leaving its value at -1.0. Furthermore, RELEST does not calculate tallied reliabilities for each constraint, leaving them also at -1.0 as a signal that no valid value is available. RELEST produces only Gaussian (normal) estimates of reliability for each failure limit.

Restrictions on Using Variable Failure Limits: The failure limit (the upper and/or lower limit on a response) must not change during the course of a RELEST run. For example, a temperature may be specified not to exceed 100 degrees. However, if the limit is relative (e.g., “do not exceed 10 degrees more than an average”) or can otherwise change during a RELEST run, then the limit is variable and RELEST should not be used. Most variable or relative reliability constraints can be easily modified to be equivalent fixed constraints (see Section 5.18).

Restrictions on Cumulative Runs: RELEST is not cumulative: consecutive calls result in the same results if not changes have been made between calls, hence there is no advantage to calling RELEST repeatedly for the same design.

Guidance Using Variables with Small or Zero Mean: If a random variable (such as a tolerance) is zero or otherwise has a very small value (less than RDERR), ADERR will be used instead as an absolute (not relative) perturbation, so the value of ADERR should be selected carefully in such cases.

Caution: Computational Noise: Whereas SAMPLE and DSAMPLE are not as sensitive to problems in the underlying PROCEDURE solution (as long as they are not systematic), RELEST, being a gradient-based approach, has Solver-like sensitivities to problems such as computational noise. Refer to Section 5.11.2 in general and Section 5.11.2.3 in particular for more details.

5.22 HEADER RELPROCEDURE

In addition to HEADER OPERATIONS and HEADER PROCEDURE, the user may add another block called HEADER RELPROCEDURE. All rules and capabilities associated with OPERATIONS apply to the RELPROCEDURE block as well, and so that description (Section 4.1.3.1) will not be repeated here.

RELPROCEDURE is converted into a routine named RELPROC.

Although the primary purposes of the HEADER RELPROCEDURE block is to define the evaluation sequence to be performed iteratively by the Reliability Engineering module drivers (SAMPLE, DSAMPLE, and RELEST), it can also be used as a convenience to replace repetitive procedures in OPERATIONS. The user can call for these procedures to be performed by inserting "CALL RELPROC" in OPERATIONS. By default, the logic placed in RELPROCEDURE is not called unless the user either invokes the Reliability Engineering module or calls the subroutine RELPROC.

Caution: Commands such as BUILD, BUILDF, DRPMOD, ADDMOD, HTRNOD, HLDLMP, etc. that are performed in OPERATIONS remain in effect upon the call to RELPROCEDURE, and any such commands made in RELPROCEDURE remain in effect upon returning to OPERATIONS.

Guidance: Calls to RESPAR or RESTAR within RELPROCEDURE are common when the evaluation procedure involves transient runs and initial conditions must be reset. If registers were saved in the original SAVPAR, SVPART, SAVE, or RESAVE call (i.e., the '-R' option was *not* invoked), then restoring those values would overwrite random variable manipulations made by the Reliability Engineering modules. To avoid this problem, RESPAR and RESTAR automatically propagate updates for any expression containing (directly or indirectly) the random variables after reading back in the data. This allows the user to save and reset other (non-design variable) registers via a prior SAVPAR (for example) call. See also Section 4.9.6.3 and Section 4.9.7.

Example:

```
HEADER OPERATIONS
        CALL RELPROC
HEADER RELPROCEDURE
BUILD CON,MYMOD
        CALL STEADY
DEFMOD HOTR
        delT = t44-t55
```

If PROCEDURE (Section 5.3) is the same as RELPROCEDURE, then one can simply be called from the other. For example, if PROCEDURE has already been input and is identical to RELPROCEDURE:

```
HEADER RELPROCEDURE
        CALL PROC
```

If on the other hand RELPROCEDURE has already been input and is identical to PROCEDURE:

```
HEADER  PROCEDURE
        CALL  RELPROC
```

5.23 Database Options and Cumulative Runs

This section describes special files that can be used to store results of Reliability Engineering module predictions. Typically, these results are used either to (1) accumulate results from several runs and therefore improve accuracy, or (2) to be able to apply hindsight for calculating the distributions of additional responses and/or additional failure limits using postprocessing software.

5.23.1 Reliability Engineering Databases and other Binary Formats

NOTE: Binary input and output are undergoing a transition from an older and soon-to-be-obsolete single "SAVE file" format to a newer format: a set of files stored within a compressed subdirectory, or "CSR folder." The term **file** *in phrases such as "SAVE file" or "REDB file" can refer to a* **CSR folder** *too. In other words, several different CSR folders may be named in OPTIONS for the purposes of output (SAVE, RESAVE, SAVEDB, CRASH) or restarting (RSO, RSI). We apologies for confusion during this transition.*

SINDA/FLUINT uses a variety of binary files to perform parametric and restart operations. For example, the SAVE routine writes cumulatively to the file or CSR folder named as the SAVE entry in OPTIONS DATA, RESAVE writes cumulatively to the RSO file or folder, and CRASH writes and overwrites a single record to a locally named file or folder. Despite the differences in names (which are employed to encourage but not enforce structured usage), all of these files use the same format, and hence all of them can be restored using RESTAR or postprocessed using external codes such as Sinaps®, EZXY®, or Thermal Desktop®.

The Reliability Engineering module introduces yet another binary results data set that *again* uses exactly the same format as the SAVE and RSO files and folders, and therefore can be used for either restarting or postprocessing. It is named in OPTIONS DATA using the following format:

```
REDB = [rel. engr. database file or folder name]
```

Uniquely, the REDB file (or CSR folder if USECSR has been invoked) can also be used like SVPART/RESPAR to perform saves and retrievals within the same run. More importantly, *it can be used to add new data to the results of an old run.*

Thus, while the remainder of this section describes the usage of the REDB data set as accessed by the SAVEDB and RESTDB routines, nothing prevents a user from more creative usage such as adding to a previous SAVE or RSO or CRASH data set created in a previous run.

5.23.2 SAVEDB and RESTDB

SAVEDB is the primary routine for adding new records into the REDB file or folder. The calling sequence is as follows:

```
CALL SAVEDB('ARGS', NREC)
```

Like SAVE and SVPART and RESAVE (Section 7.2), the amount of data that is to be stored is controlled by alphanumeric arguments ('ARGS'). Since the primary purpose of this routine is to support the Reliability Engineering module, failing to store register, Solver, and Reliability Engineering module data via the 'R' (included in the 'ALL' flag) would be an aberrant case.

Like SVPART, this routine returns a record number (NREC) for the last record written, such that this record can be restored later in the same run using RESTDB.

Like SAVE or RESAVE, SAVEDB will overwrite an existing REDB file or folder if it is called without a prior call to RESTDB.

RESTDB has many similarities to both RESPAR and RESTAR. It is called as follows:

```
CALL RESTDB(NREC)
```

The actions of RESTDB depend on the sign of NREC:

1. If NREC is positive, then that record (creating during the current or previous run) is restored, *and subsequent calls to SAVEDB begin overwriting after that record*.
2. If NREC is zero, then the last record (creating during the current or previous run) is restored, *and subsequent calls to SAVEDB begin appending to the file after that record (i.e, at the end of the file)*. This is the most common usage of RESTDB.
3. If NREC is negative, this signals the code that an existing REDB file or folder is present and should be appended instead of overwritten. However, the last state will not be restored.

These routines are fully documented in Section 7. The purpose of this section is to introduce this database and to describe its usage with respect to the Reliability Engineering module.

5.23.3 Using REDB Files with SAMPLE and DSAMPLE

SAMPLE and DSAMPLE are both cumulative routines: accuracy improves with increase samples (though not as much in DSAMPLE as in SAMPLE).

To take advantage of this fact, place a call to SAVEDB in RELOUTPUT CALLS, such as:

```
HEADER RELOUTPUT CALLS
        CALL SAVEDB('ALL',NTEST)
```

To add to this file or folder in later runs, add a call to RESTDB using NREC=0 in OPERATIONS, such as:

```
HEADER OPERATIONS
BUILD ALL
        CALL RESTDB(0)
        CALL SAMPLE
```

The use of NREC=0 is critical to yield improved accuracy from repeated runs. Otherwise SAMPLE and DSAMPLE will not begin where the previous run left off.

Otherwise, using an REDB file or folder with SAMPLE and DSAMPLE allow hindsight to be applied in the investigation of results using postprocessors (Section 5.23.5).

5.23.4 Using REDB Files and Folders with RELEST

RELEST is not cumulative. Therefore, the only reason to use REDB with that routine is to be able to apply hindsight in the investigation of results using postprocessors (Section 5.23.5).

5.23.5 Hindsight: Using Postprocessing

RELCONSTRAINT DATA provides a mechanism for defining responses and failure limits in advance. This data allows SAMPLE to determine convergence, as well as provides mechanisms for printing out important results using RCSTTAB or fetching them dynamically using RCGET or RCGETNO.

However, this usage requires that the desired responses and their limits all be known ahead of time, and yet estimating reliability can be quite expensive, especially when SAMPLE is employed. Therefore, extensive postprocessing tools exist to allow a user to investigate other responses and other limits after a run has been made by postprocessing the REDB results data.

Specific tools include histograms and scatter plots. A histogram is a bar plot that allows a user to visualize the results of RCSTTAB. More importantly, it allows users to tally new limits and to estimate the mean and standard deviation for new responses. In effect, it allows users to apply new reliability constraints dynamically as afterthoughts. To have the maximum ability to apply hindsight, the ‘ALL’ option should be used in the SAVEDB call, to the limit that large files can be tolerated.

A scatter plot allows the user to plot one variable (register, response, etc.) against another as a series of points, not lines. This is especially useful when plotting results from SAMPLE and DSAMPLE. It allows a user to see if any two effects are linked, and if so what trends exist: does an increase in one factor lead to an increase or a decrease in the other?

Results from RELEST runs (assuming SAVEDB was applied in RELOUTPUT CALLS) are not plotted. However, they can once again be interrogated to find the predicted reliability of new responses and new constraints.

5.24 Reliability Support Subroutines

This section lists support subroutines that are useful with the Reliability Engineering module. These abbreviated descriptions are wholly redundant: complete descriptions may be found in Section 7.13.

5.24.1 Output Routines

REGTAB (no arguments)—Tabulates registers, whether they are integer or real, their current values, and the expressions defining them. Internally calls CALREG to refresh registers prior to tabulation.

RANTAB (no arguments)—Tabulates random variables and prints data to describe their distributions, including mean, standard deviations, and some sample points within the distribution. This routine can be called to check inputs before a call to SAMPLE, DSAMPLE, or RELEST, but it also prints other information about the current status of those runs in case it is called during or after such a run.

RCSTTAB (no arguments)—Tabulates reliability constraint variables and expressions and their limits, if any. Identifies the reliability of each limit as using both tally methods and methods assuming normal (Gaussian) responses of fixed limits. Also contains other information about the current status of the Reliability Engineering module, including predictions (if any) regarding overall reliability (OVEREL). ***This is the primary output routine for the Reliability Engineering module.***

The tallied reliabilities (including OVEREL) are *always* valid for SAMPLE and DSAMPLE, but are *never* valid for RELEST. These tallies simply represent the number of times a limit was not violated divided by the total number of samples.

Otherwise, reliabilities may also be estimated by assuming a normal (Gaussian) response and predicting the likelihood of a response being above or below a limit based solely on the mean and standard deviation of the response. All reliability estimation routines produce the mean and standard deviation of all responses, although their internal methods differ. Therefore, all routines produce reliability predictions based on this method, and this is in fact the primary output of the RELEST routine. For SAMPLE and DSAMPLE routines, these “normal reliabilities” should only be used when the limits themselves are fixed (i.e., they do not change during the SAMPLE or DSAMPLE run).

5.24.2 Auxiliary Subroutines

CURMEAN(‘NAME’)—That takes the current value of a register and applies it as the mean value for one or more NORMAL random variables. ‘NAME’ is the name of the NORMAL random variable in single quotes, otherwise use ‘ALL’ to change the means of all NORMAL random variables. CURMEAN might be used, for example, to evaluate the reliability of a design which was just produced using the Solver, or in any case where the mean values are not known ahead of time.

Otherwise, the mean values or any other distribution parameter (e.g., UNIFORM range limits, even the array data underlying an ARRAY random variable) can always be set to registers or other expressions and changed at any time between (but not during) calls to SAMPLE, DSAMPLE, and RELEST.

Caution: If a NORMAL random variable was defined using CV instead of SD, changing the mean will change the value of the standard deviation as well.

RESETM('NAME')—Resets one or more random variables to their mean values. 'NAME' is the name of the NORMAL random variable in single quotes, otherwise use 'ALL' to change *all* random variables (not just NORMAL ones) to their mean values. RESETM is somewhat the opposite of CURMEAN, except that it applies to any or all random variables, not just NORMAL ones. It is useful to return to mean values after having called SAMPLE, DSAMPLE, or RELEST, which will otherwise leave the random variables at their last perturbed value upon return.

RESAMP (no arguments)—By default, SAMPLE and DSAMPLE are cumulative: they will append their results to the results of a previous call to the same routine (as long as no other type of Reliability Engineering module driver was called in between). To avoid this cumulative behavior, a call to RESAMP can be made before starting a second SAMPLE or DSAMPLE run. RESAMP does not affect whether or not SAVEDB overwrites or appends to the REDB file, since that is controlled independently. RESAMP is useful when SAMPLE or DSAMPLE are being called within PROCEDURE as part of using reliability as an objective or constraint for the Solver (see also Section 5.27).

RCGET('NAME', RM, RSD, RTLO, RNLO, RTHI, RNHI)—RCGET is used to return the same results as does the routine RCSTTAB, but in a Fortran callable fashion such that the resulting reliabilities can be used by the user in logic (perhaps as a constraint or objective for the Solver, see also Section 5.27). 'NAME' is the alphanumeric name of the named reliability constraint response in single quotes. The rest of the arguments are returned by the routine, and represent the columns output by RCSTTAB:

RM . . . . . Mean of the response
RSD . . . . . Standard deviation of the response
RTLO . . . . . Tallied reliability of the lower limit (may be -1.0 if not available or not applicable)
RNLO . . . . . Reliability of the lower limit assuming a Gaussian (normal) response (may be -1.0 if not available)
RTHI . . . . . Tallied reliability of the upper limit (may be -1.0 if not available or not applicable)
RNHI . . . . . Reliability of the upper limit assuming a Gaussian (normal) response (may be -1.0 if not available)

RCGETNO(NUMRC, RM, RSD, RTLO, RNLO, RTHI, RNHI)—RCGETNO represents the same functionality as RCGET, but is used for unnamed reliability constraints. To refer to them, the first argument is replaced by an integer identifier instead of an alphanumeric name in quotes such as RCGET uses. This identifier, NUMRC, is simply the number of the constraint in input order (including named constraints), as printed in the first column of RCSTTAB. For example, to return information on the fifth input reliability constraint, use “CALL RCGETNO(5, ...).”

NORMVAL(P, A, S, X)—A normal (Gaussian) distribution utility routine that returns the value X that corresponds to the input probability P, where P is the probability that a random value between negative infinity and X will occur. A is the mean, and S is the (positive) standard deviation. P should be between 0.0 and 1.0. For example, if P=0.01 then a value less than X will occur 1% of the time, and a value greater than X will occur 99% of the time. If P=0.5, the mean A (which is equal to the median for normal distributions) will be returned.

NORMVAL can be used with either the mean and standard deviations input with NORMAL random variables, or with the resulting means and standard deviations resulting from RCGET or RCGETNO (or RCSTTAB from a previous run) to calculate additional failure limits other than those posed in RELCONSTRAINT DATA, assuming that the response is Gaussian.

NORMPROB(X, A, S, P)—The reverse of NORMVAL: a normal distribution utility routine that returns the probability P that a random value between negative infinity and the input value X will occur. A is the mean, and S is the (positive) standard deviation. For example, if P is returned as 0.8, then a value less than X will occur 80% of the time, and a value greater than X will occur 20% of the time.

NORMVAL can be used with either the mean and standard deviations input with NORMAL random variables, or with the resulting means and standard deviations resulting from RCGET or RCGETNO (or RCSTTAB from a previous run) to calculate the reliabilities for additional failure limits other than those posed in RELCONSTRAINT DATA, assuming that the response is Gaussian.

5.25 Brief Reliability Engineering Example

Refer also to the Sample Problem Appendix for different examples.

This example expands upon the example presented in Section 1.5.4, with the following changes:

1. To illustrate ARRAY variables, the uncertainty in POWER is changed from NORMAL to an asymmetric triangular distribution such that the probability of POWER values less than 9.0 or more than 12.0 is zero, but the most likely point is at 10.0.
2. To demonstrate the difference between all three reliability estimation routines, all three are called within one run.

The complete input file is as shown:

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM, USING RELIABILITY ENGINEERING
        OUTPUT          = barrel.out
        MODEL           = TEST
HEADER REGISTER DATA
        DENS            = 0.3
        CP              = 0.2
        CON             = 0.5/12.0
        MPER            = DENS*WIDE*LONG/RESOL
        EMIS            = 0.1
        WIDE            = 1.0
        POWER           = 10.0
        X1              = 8.5
        X2              = 17
        X3              = 34
        LONG            = 3.0
        HR2MIN          = 60.0
        RESOL           = 3.0
HEADER RANDOM DATA
        EMIS,UNIFORM,0.08,0.12
        WIDE,NORMAL,SD=0.01
        POWER,ARRAY,SUB1.A100
HEADER ARRAY DATA, SUB1
        100 =         9.0,       0.0
                      10.0,      1.0
                      12.0,      0.0
HEADER CONTROL DATA,GLOBAL
        EBALSA          = 0.001
HEADER NODE DATA, SUB1
        1,500.0,(0.5*X1)*CP*MPER
        2,500.0,(0.5*(X1+X3)+X2)*CP*MPER
        3,500.0,(0.5*X3)*CP*MPER
        -99, -460., 0.0
HEADER CONDUCTOR DATA, SUB1
C FORMULA FOR WEDGE
        12,1,2,CON*(X1-X2)*WIDE/(LONG/RESOL)/LN(X1/X2)
        23,2,3,CON*(X2-X3)*WIDE/(LONG/RESOL)/LN(X2/X3)
        -399, 3, 99, EMIS*X3*WIDE*sbcon/(HR2MIN*144.)
HEADER SOURCE DATA, SUB1
```

```
        1, POWER/(btuhrwat*HR2MIN)
HEADER RELCONSTRAINT DATA
        SUB1.T1  <= 500.0
HEADER OPERATIONS
BUILD ALL
        NLOOPS         = 100
        NLOOPR         = 100
        CALL RANTAB
        CALL DSAMPLE
        CALL RCSTTAB
        NLOOPR         = 10000
        CALL SAMPLE
        CALL RCSTTAB
        CALL RELEST
        CALL RCSTTAB
HEADER RELPROCEDURE
        CALL STEADY
END OF DATA
```

The results (which can vary from version to version, and even run to run given the random nature of the sampling!) are as follows:

Value	SAMPLE	DSAMPLE	RELEST
Number of Samples	1002	100	4
Tallied Reliability	0.404	0.44	N/A
Normal Reliability	0.376	0.377	0.455
Response Mean	510.0	510.2	508.5
Response Std. Dev.	31.7	32.6	74.8

It took SAMPLE about 1000 iterations to converge, and it achieved comparable results to DSAMPLE, which took 10% as many iterations (by choice). The tallied reliability from DSAMPLE is intrinsically accurate (*at most*) to the nearest 1% because only 100 samples were taken.

RELEST provided nearly the same estimate in only 4 calls to RELPROC, despite being off considerably in the estimate of the standard deviation of the response (sub1.t1). Because of the nonlinearities in the response (due to radiation heat transfer) and due to the fact that the random variables were not Gaussian, RELEST was not strictly applicable to this problem, but performed adequately for an inexpensive first order estimation.

However, before the results of DSAMPLE and SAMPLE are considered final, consider the results of a second “identical” run (repeated run on same machine, same version, etc.):

Value	SAMPLE	DSAMPLE	RELEST
Number of Samples	1003	100	4
Tallied Reliability	0.408	0.39	N/A
Normal Reliability	0.371	0.370	0.455
Response Mean	510.5	510.1	508.5
Response Std. Dev.	31.7	30.5	74.8

The RELEST results are unchanged as expected, but both the SAMPLE and DSAMPLE results differ, especially the tallied reliability of DSAMPLE. The reason the SAMPLE results differ is that it started with a different sequence of random numbers and hence sampled 1000 different design instances than the first run. The fact that it converged does not mean an accurate answer, merely that the answers were asymptotically approaching a limit. Tighter criteria (RERRR) would have required many more samples (perhaps as many as 10,000), but would have approached an answer that did not change significantly from run to run.

The reason the DSAMPLE results differ is analogous. Despite the fact that the same values of random variables were used in both runs, the *combinations* of those values were chosen at random and hence different design instances were sampled in the second run than in the first run. In other words, if only one random variable existed, repeated DSAMPLE runs would yield identical results (subject to tolerances in the underlying SINDA solutions) since the same values would be sampled, albeit in different orders.

Notice that the *tallied* reliability (the fraction of total samples that don't exceed a reliability constraint) is more volatile in DSAMPLE than SAMPLE since about ten times more samples are made when SAMPLE is used. Nonetheless, the "*normal* reliability" (calculated based on a curve fit to a normal or Gaussian curve) of the DSAMPLE result is stable and is in effect the same answer as that arrived at by SAMPLE ... at one tenth of the computational cost. The tallied reliability of SAMPLE is arguably the best value to use, whereas for DSAMPLE the normal reliability will often be a more valuable result.

5.26 Parametric Sweep of a Random Variable

This section describes a simple routine that performs a parametric sweep of a single random variable. This is not intended as a replacement for the more complete routines (e.g., DSAMPLE) introduced earlier. Rather, this feature is useful for either debugging a problem or investigating trends for a reduced problem space.

Calling Sequence:

```
CALL RVSWEEP(rvnam)
```

where:

`rvnam` ..... Random variable name, in single quotes

RVSWEEP makes NLOOPR calls to RELPROCEDURE using equal cumulative probability increments of the random variable from the lower to the upper limit. For example, if NLOOPR=10 then values of rvnam corresponding to cumulative probabilities of 5%, 15%, ... 95% will be chosen. (RVSWEEP is therefore identical to DSAMPLE applied to a problem with a single random variable, except that the values are invoked in a known, nonrandom order.) RELOUTPUT CALLS is called after each evaluation.

Upon completion, RVSWEEP updates statistical information in the same manner as does DSAMPLE. However, since this statistical information is based upon a single variable, it is of limited usefulness: judging the relative effects of a single random variable.

Example:

```
CALL RVSWEEP('future')
```

Guidance: This routine ignores evaluated points if trouble is encountered during the solution procedure. Use the NERVUS control constant in SOLVER DATA to adjust tolerance of nonconvergence and other problems.

5.27 Robust Design: Reliability-based Optimization

The Reliability Engineering module enables a user to estimate the reliability of a point design based on uncertainties in the dimensions, properties, and boundary conditions. The Solver enables a user to size or select dimensions, properties, etc. such that mass is minimized, or such that performance is maximized, etc. This section lists ways in which these two modules can be combined to yield even more powerful design tools, and provides suggestions and cautions for combined usage.

Listed below are a few possible combinations of these modules:

1. a design can be selected using the Solver, and then (in the same or later run) the reliability of that design can be estimated
2. the reliability of a design can be used as an objective (“maximize reliability” or “minimize the chances of failure”)
3. the reliability of a design can be used as an optimization constraint (“find the minimum mass design that achieves a reliability of at least 99%”)
4. the range or variance of a random variable can be used as a design variable (“what variation can be tolerated: how tight must tolerances be?”)

In the first case, the Solver and Reliability Engineering modules are not combined so much as executed in series. In that case, few restrictions or suggestions exist: application is largely straightforward. Note, however, that it might be best to apply a random variable as the uncertainty in a parameter rather than the parameter itself. Consider for example the definition of a hydraulic diameter, where the diameter itself is design variable, while the tolerance is a random variable:

```
HEADER REGISTER DATA
        DIAM    = 0.12  $ will also be the mean value
        DTOL    = 0.0
        DSD     = 0.01
HEADER FLOW DATA ...
        ... DH = DIAM + DTOL ...
HEADER DESIGN DATA
        0.08 <= DIAM <= 0.5
HEADER RANDOM DATA
        DTOL, NORMAL, SD = DSD
```

A first run might size DIAM using the Solver. A later run might predict the reliability of the design letting the final value of DH (= DIAM+DTOL) vary within a machining tolerance.

The last three cases listed above involve nesting the Reliability Engineering module as a subset of the Solver. In other words, calling SAMPLE, DSAMPLE, or RELEST within the PROCEDURE block in order to evaluate the reliability of a particular design being attempted by the Solver. An example of this nesting was briefly discussed in Section 1.5.5.

The cost associated with such nested combinations can be large, usually untenably large for realistic problems if SAMPLE or even DSAMPLE is used inside of PROCEDURE. The Solver might require between 30 and 300 evaluations, and if each of these required evaluations of the same type (in other words, assuming the cost of PROC were about the same as the cost of RELPROC), then the resulting cost of reliability optimization might require on the order of 10,000 evaluations for DSAMPLE and 100,000 for SAMPLE.

This is one of the purposes for RELEST, which is inexpensive enough to be nested. In the above case, only about 100 to 500 evaluations would be required, depending on the number of random variables and the number of Solver evaluations (which itself is dependent on the number of design variables as well as other factors). However, RELEST cannot be used universally. Effort must be made to avoid strongly nonlinear problems and problems with highly non-Gaussian distributions of random variables. Effort must also be made to phrase reliability constraints using fixed limits if possible, per the examples in Section 5.18.

To use RELEST as the basis for predicting reliability within the Solver, it is recommended that the applicability of RELEST first be tested by comparisons with the results of DSAMPLE and/or SAMPLE using the initial design. It is also recommended that the reliability of the final design be checked using DSAMPLE or SAMPLE to assure that the RELEST predictions are conservative if not accurate.

If either DSAMPLE or SAMPLE are used instead of RELEST within PROCEDURE, then the user must call RESAMP (Section 5.24.2) before the call to SAMPLE or DSAMPLE to make sure that the reliability estimation is restarted each time (for each design instance being evaluated).

Furthermore, if RELEST or another reliability estimation routine is called before any other optimization evaluations of the same design, a call to RESETM (Section 5.24.2) is probably required in between, such that those subsequent analysis are made on the basis of mean instead of perturbed or random values.

When nesting Reliability Engineering calculations within the Solver, note that design variables cannot be the same as random variables. (This restriction does not otherwise exist if the modules are not nested, and therefore is not checked by the code.) Otherwise, the reliability engineering modules would change a design variable's value during the call to PROCEDURE, and this is prohibited when using the Solver. A simple separation of a variable into a basis and tolerance (as shown in the example above using DH=DIAM+DTOL) is one means of avoiding such conflicts. Otherwise, there is no reason why a reliability can't be used as an objective (case #2 above) or as an optimization constraint (case #3), or why a parameter describing a random variable distribution (such as DSD in the above example) can't be used as a design variable (case #4), presuming the engineer has some control over the tolerancing of the design.

In cases #2, #3, and #4 above, the results of the RCSTTAB routine must be fetched within the PROCEDURE block such that OBJECT or an optimization constraint can be updated. This is one purpose of the RCGET and RCGETNO routines (Section 5.24.2), which should be called (repeatedly if needed) after the call to RELEST, SAMPLE, or DSAMPLE. If SAMPLE or DSAMPLE are affordable, the overall reliability (OVEREL) may be accessed in logic or expressions, but being a tallied reliability it is discontinuous.

5.27.1 Problems Using Reliability as OBJECT or Optimization Constraint

There are several problems that can be encountered when using reliability estimations as objectives or constraints.

First, note that tallied reliabilities (OVEREL, and the RTHI and RTLO results of a RCGET or RCGETNO call) are actually discontinuous and therefore should not be used per the rules of operation for the Solver. This is only a concern if SAMPLE or DSAMPLE is used, since RELEST does not produce tallied reliabilities. This discontinuity is especially problematic with DSAMPLE because of the coarse resolution of its tallies: if NLOOPR=100, for example, then the tally can only assume values such as 95%, 96%, 97%, etc. *If SAMPLE or DSAMPLE is used, use RNHI or RNLO instead if they are applicable.*

Second, there can be problems caused by a design that initially or excessively returns 0.0 or 1.0 as a reliability, including numbers very close to zero or to one. To the Solver, such a design appears to be unresponsive to its changes and so the Solver will either abort or finish prematurely (giving up).

The initial design should therefore have a reasonable reliability, avoiding values either too small or too close to unity, and the user should be aware that if an intermediate design is too poor (perpetually small, nearly zero reliability) or too good (perfect reliability) then the reliability is locally constant and again the Solver might either abort or finish prematurely. Using a scaled reliability (raised to a power, as described below) can avoid problems at either the high end, or the low end, but not both.

Third, target reliabilities are usually very nearly unity: 0.99 or 0.99999 etc., and yet reliability never exceeds unity. This can cause trouble when used as an objective or constraint, since (from the Solver's perspective) no significant change in reliability is noticed when design variables are perturbed. To overcome this problem, consider using probability of failure (one minus reliability) as an objective to be minimized, or as an upper limit on an optimization constraint.

Another approach is to consider scaling the reliability. For example, instead of:

```
OBJECT  = RELIABILITY
```

or

```
0.99 <= REL
```

consider

```
OBJECT  = RELIABILITY**10
```

or

```
0.99**10 <= REL10 $ where in logic REL10 = REL**10
```

Note that the above treatment would only worsen unresponsiveness problems described above if the reliability is initially or perpetually very low. In other words, the above rescaling of reliability works well as long as the reliability of the initial design is reasonable.

5.27.2 Reliability Constraints vs. Optimization Constraints

Confusion often exists between reliability constraints and optimization constraints (which are usually referred to simply as “constraints”). An optimization constraint defines a rule that should not be violated by a feasible (and usually deterministic) design. A reliability constraint defines a rule that should not be violated by a successful instance of an uncertain design.

Generally, an optimization constraint should include margin or factors of safety if the Robust Design features outlined in this subsection are not or cannot be employed. On the other hand, a reliability constraint should use the actual failure limit (assuming the underlying model is 100% accurate, which of course is unrealistic).

For example, if a chip fails when the junction temperature exceeds 125°C, then during optimization (preliminary design) the junction temperature should not be allowed to exceed perhaps 104°C:

```
HEADER CONSTRAINT DATA
       Tjunc <= 125.0-21.0
```

This allows a margin of 21°C (using certain U.S. military standards* for passive thermal designs as an example) for uncertainties in inputs plus uncertainties in analysis. But when the reliability is estimated, uncertainties in inputs are presumably already taken into account, so a smaller margin (11°C per U.S. military standards) should be applied as a reliability constraint:

```
HEADER RELCONSTRAINT DATA
       Tjunc <= 125.0-11.0
```

The design would then be checked to assure that it meets the reliability requirements levied upon the thermal subsystem. If it exceeds that required reliability, then over-design exists. If the design is not reliably enough, then it must be redesigned (reapplying the original optimization constraint but applying even greater margin). Either way, a truly optimal design (using Robust Design methods outlined in this subsection) will achieve exactly the required reliability and thus be neither over- nor under-designed.

To use Robust Design methods, only the reliability constraint need be applied. Also applying the optimization constraint is legal and perhaps even advisable, but that derived limit may become the limiting factor instead of the less-derived reliability constraint, thereby resulting in over-design.

Finally, it should be noted that even the 125°C limit levied upon the thermal designer is uncertain: it contains margins and/or a hidden reliability predictions. A truly optimal multidisciplinary design would factor in the reliability of the chip directly, rather than indirectly as an inflexible limit.†

---

* Margins for many commercial designs are typically lower by about half. The actual numbers used in this subsection are for illustration purposes only and do not imply what margins or factors of safety are adequate for any specific application.

† Commercial tools exist such as Engineous’ iSIGHT® that can perform optimization, reliability, and robust design methods above the level of what can be accomplished within a thermal/fluid analyzer such as SINDA/FLUINT. SINDA/FLUINT provides links to iSIGHT to encourage such high-level design processes.

5.27.3 Advanced Hint for Further Reducing Run Times

Often, the optimization evaluation procedure (PROC) is exactly the same as the reliability evaluation procedure (RELPROC). In fact, they are often both simply steady state solutions, although this fact is not a requirement for exploiting the following technique.

If PROC and RELPROC as essentially the same and if RELEST is used within PROC, then the user can exploit the fact that the first RELPROC call from within RELEST is based upon mean values of random variables, and can therefore be used to update OBJECT and named optimization constraints.

For example, the following logic could be used:

```
HEADER RELOUTPUT CALLS
       IF(LOOPCR .EQ. 1)THEN
              OBJECT  = ...
              NAMCONST = ...
       ENDIF
```

Then the PROCEDURE would simply consist of:

```
HEADER PROCEDURE
       CALL RELEST
```

If there are 3 random variables, the cost savings would be 1/(3+1) or 25%.

If OBJECT is not updated in logic (i.e. if it is defined completely by an input expression in SOLVER DATA) or if any unnamed optimization constraints are used, then the first iteration of RELEST must be saved and later retrieved. This internal parametric can be performed using SAVPAR and RESPAR, or by using SAVEDB with RESTDB. For example:

```
HEADER RELOUTPUT CALLS
       IF(LOOPCR .EQ. 1)THEN
              OBJECT  = ...
              NAMCONST = ...
              CALL SAVPAR(MTEST)
       ENDIF
```

Then the PROCEDURE block would consist of:

```
HEADER PROCEDURE
       CALL RELEST
       CALL RESPAR(MTEST)
```

The above usage will not violate restrictions on overwriting design variables using RESTAR or RESTDB because the design variables were unchanged between the storage and retrieval steps.

6 REFERENCE SUMMARY

This section is intended to serve as a fast look-up list of formats for experienced users. Figure 6-1 displays all possible input blocks that comprise an input file for SINDA/FLUINT. Note that the blocks may appear in any order with the exceptions of the OPTIONS DATA block, which must appear first, and the global CONTROL DATA and USER DATA blocks (if any) which must appear before submodel-specific CONTROL DATA and USER DATA blocks (if any).

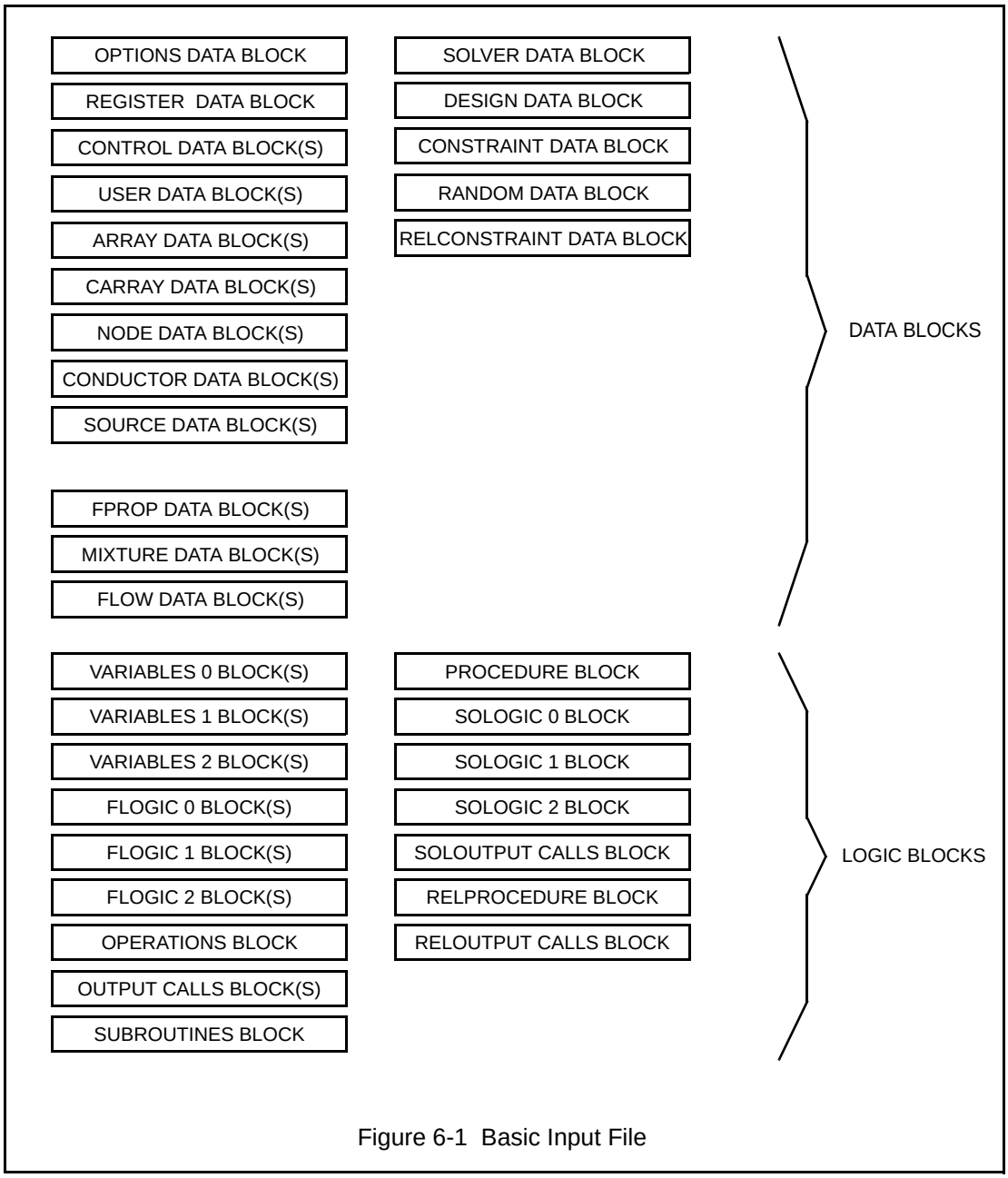


Figure 6-1 Basic Input File

6.1 Basic Conventions - Data Blocks

Data Types:

INTEGER . . . . . .up to 10 numeric characters
FLOATING POINT. . . . . .up to 20 numeric characters
EXPRESSION. . . . . .up to 1000 characters, treated as FLOATING POINT
CHARACTER. . . . . .up to 32 characters (this doesn't include CARRAY data, which may be 1000 characters long)

Comments:

```
      code    data-values $ comments
C     comment
```

Built-in Constants in Expressions:

Variable Name	Value	Description
pi	3.141592653589739	
sbcon	0.1712E-08 BTU/h-ft^2-R^4	Stefan-Boltzmann constant
sbconsi	5.6693E-08 W/m^2-K^4	sbcon in SI units
dtor	0.017453292519943	Degrees to radians
rtod	57.2957795131	Radians to degrees
cmtoin	0.3937007874	Cm. to inches
mtoft	3.280839895	Meters to feet
intocm	2.54	Inches to cm.
fttom	0.3048	Ft. to meters
inhgtoat	3.342E-02	In. Hg to atmospheres
psitoat	6.804E-02	PSI to atmospheres
jtobtu	9.480E-04	Joule to BTU
btuhrwat	2.930722E-01	BTU/Hr to Watt
lbf3kgm3	16.01846	lb/ft^3 to kg/m^3
lbf2n	4.44822	lb$_f$ to Newton
lb2kg	0.4535924	lb. to kg.
kgs2lbh	7.936641E+03	kg/sec to lb/hr
psi2pa	6.894757E+03	psi to Pascals
grav	32.1725*3600.0**2 ft/hr^2	gravity in FLUINT ENG units
gravsi	9.80621 m/sec^2	gravity in SI units
gasr	1545.0 ft-lb$_f$/lb$_{mol}$-R	gas constant
gasrsi	8314.23 J/kg$_{mol}$-K	gas constant in SI units
ft3tom3	2.831685E-02	cubic feet to cubic meters
wmk2bhfr	1.73073	watt/m-K to BTU/hr-ft-R

Built-in Functions in Expressions:

Function	Description	Example
abs	absolute value	abs(float)
acos	arccosine	acos(float)
asin	arcsine	asin(float)
atan	arctangent	atan(float)
ceil	round-up (to next larger integer)	ceil(float)
cos	cosine (given radians)	cos(radians)
exp	base-e exponentiation (opposite of “ln”)	exp(float)
floor	round-down (to next smaller integer)	floor(float)
ln	natural (base-e) logarithm	ln(float)
log	base-10 logarithm	log(float)
max	return larger of two values	max(float1,float2)
min	return smaller of two values	min(float1,float2)
sin	sine (given radians)	sin(radians)
sqrt	square root	sqrt(float)
tan	tangent (given radians)	tan(radians)

6.2 Options Data

Basic Format:

```
HEADER OPTIONS DATA
        options data statements
```

Options Data Example:

```
HEADER OPTION DATA
TITLE   SAMPLE OPTION DATA
    RSI     =   SAMPLE.RSI
    RSO     =   SAMPLE.RSO
    OUTPUT  =   SAMPLE.OUT
    SAVE    =   SAMPLE.SAV
    QMAP    =   SAMPLE.QMP
    PLOT    =   SAMPLE.PLT
    USER1   =   SAMPLE.USR
   MODEL     =    SAMPLE
   NOLIST
   MLINE     =    66
   SPELLOFF
```

Options Data Summary

Variable Name	Description and Options	Default
TITLE	Problem title- up to 72 alphanumeric characters	None - mandatory input
PPSAVE*	Input save file	None - optional input
RSI	Restart input file or folder	None - optional input
RSO	Restart output file or folder	None - optional input
OUTPUT	Processor print file	Same as preprocessor print file
SAVE	File or folder for SAVE data	None - optional input
REDB	File or folder for Rel. Engr. data	None - optional input
QMAP	File for QMAP and FLOMAP data	None - optional input
USER1 USER2	User auxiliary files	None - optional input
MODEL*	Model name, up to 32 characters	'ROOT'
PPOUT*	Input data print/save control flag. ALL1, ALL2, ACTIV1, ACTIV2	ALL2 - Save all input, expanded
NOLIST*	No printout flag	None - optional input
MLINE	Number of printed lines per page (integer)	60
SPELLOFF	User logic spell checker flag	*Will* check spelling
DIRECTORIES*	Print flag for node, conductor, array, lump, path, tie, and user constant directories.	Off
OLDEXPR*	Directs that expressions in data blocks be evaluated using old (pre Version 3.2) methods.	Modern expression syntax
NOUPDATE*	Turns off the automatic updating of expressions using registers and processor variables	Automatic update
MIXARRAY	Turns off checks of mixed real and integer types within arrays	W*ill* check for mixed types
NODEPAIR*	Accepts multiple node pair conductors, disables HR	Multiple node pairs illegal, accepts conductor HR
ENDCHAR	Custom in-line comment character	$ (dollar sign)

* These options are outdated or rarely used

6.3 Register Data

Block Format:

```
HEADER REGISTER DATA
        [{INT:}regnam = express]
        [{INT:}regnam2 = express2]
```

where:

`INT:`	Integer identifier prefix (for integer variables in logic blocks)
`regnam`	an arbitrary register name
`express` ...	arbitrary expression, perhaps including other register names and/or processor variables

Once defined, register names may be used alone or within an expression anywhere in the input file that a floating point value or expression is required. They may be changed in logic and the changes can be propagated throughout the model. Real registers may also be defined as design variables or correlation variables for use in design optimization, goal seeking, and test data correlation. Real registers may also be defined as random variables for use in reliability estimation and robust design.

Restrictions:

a. Each register must be defined on one line not to exceed 1000 characters.
b. No more than one register defined per line.
c. Register names must be unique alphanumeric strings not to exceed 32 characters.
d. The defining expression for a register can refer to other registers, but not to itself. Circular references that result in infinite loops must be avoided, including those involving processor variables.

Register Data Input Options Example:

```
HEADER REGISTER DATA
        ANGLE   = ACOS(SIDE2/SIDE1) + PI
        SIDE1   = 2.0*SIDE2
        SIDE2   = 1.523
        DELT    = MOD.T44 - MOD.T55
        PIPE_DELTA_PRESS = FLOW.PL3310 - FLOW.PL4410
```

6.4 Node Data

Block Format:

```
HEADER NODE DATA, smn [,fac] [,fac,tcon] [,NWORK = I]
       node data cards
```

where:

smn. . . . . thermal submodel name
fac . . . . . capacitance conversion factor (cannot be changed in logic)
tcon . . . . . temperature conversion flag

Node Types:

Diffusion . . . . . have a thermal capacitance, can store energy
Arithmetic . . . . have a zero capacitance
Boundary . . . . . have a constant temperature
Heater. . . . . have a constant temperature

Restrictions:

a. All transient *thermal* problems must include at least one diffusion node.
b. Node ID numbers should not exceed 6 digits.

Reference Forms:

```
T(n),Tn  =  temperature
Q(n),Qn  =  source
C(n),Cn  =  capacitance
```

where n is an actual node number, positive even when referencing boundary nodes.

Diffusion nodes have a T, Q and C.
Arithmetic and Heater nodes have a T and Q.
Boundary nodes have only a T.

Within expressions, a node can refer to its own data (T, Q, C) using “#this” instead of its ID.

Units:

Temperature . . . degrees
Source . . . . . energy/time
Capacitance . . . energy/degree

Node Data Input Options:

```
N#,Ti,C                                  $ Single node*

GEN N#,#N,IN,Ti,C                        $ Group of nodes*

SIV N#,Ti,AP,F                           $ C vs. T (interp)
SPV N#,Ti,AC,F                           $ C vs. T (poly)

SIM N#,#N,IN,Ti,AP,F                     $ GEN and SIV
SPM N#,#N,IN,Ti,AC,F                     $ GEN and SPV

DIV N#,Ti,AP1,F1,AP2,F2                  $ 2 mat'l (interp)
DIV N#,Ti,SUB,F1,AP2,F2                  $ 2 mat'l,C1=const
DIV N#,Ti,AP1,F1,SUB,F2                  $ 2 mat'l,C2=const

DPV N#,Ti,AC1,F1,AC2,F2                  $ 2 mat'l (poly)
DPV N#,Ti,SUB,F1,AC2,F2                  $ 2 mat'l,C1=const
DPV N#,Ti,AC1,F1,SUB,F2                  $ 2 mat'l,C2=const

 DIM N#,#N,IN,Ti,AP1,F1,AP2,F2           $ GEN and DIM
DIM N#,#N,IN,Ti,SUB,F1,AP2,F2
DIM N#,#N,IN,Ti,AP1,F1,SUB,F2

 DPM N#,#N,IN,Ti,AC1,F1,AC2,F2           $ GEN and DPV
DPM N#,#N,IN,Ti,SUB,F1,AC2,F2
DPM N#,#N,IN,Ti,AC1,F1,SUB,F2

BIV N#,Ti,AB,F                           $ C vs. T & time
```

* Note:

Diffusion. . . . . positive node number, positive capacitance
Arithmetic. . . . positive node number, negative capacitance
Boundary . . . . negative node number, non-negative capacitance
Heater . . . . . negative node number, negative capacitance

where:

N# . . . . . Actual node number (non-zero integer)

T_i . . . . . Initial temperature (floating point or expression)

C. . . . . Capacitance (floating point or expression)

#N . . . . . Number of nodes (non-zero integer)

IN . . . . . Increment to the node number (non-zero integer)

AP_n . . . . . References to arrays of temperature vs. capacitance *points* (Integer count form)

AC_n . . . . . References to arrays of polynomial *coefficients* for C = P(T) (Integer count form), [P_O, P_1, P_2, ... P_N]

F_n. . . . . Multiplying factors (floating point data values or expressions)

AB . . . . . Reference to a bivariate array of temperature and time vs. capacitance (Integer count form)

SUB . . . . . Constant value substitute for an array reference (floating point value or expression)

Summary of NODE DATA Input Options

OPTION (CODE)	NODE TYPE D	A	B	H	DESCRIPTION
3 blanks	♦	♦	♦	♦	To input a single node where the capacitance is given as a single, constant value
CAL	♦				Available for backwards compatibility, but undocumented
GEN	♦	♦	♦	♦	To generate a group of nodes, each having the same initial temperature and the same capacitance
SIV SPV	♦				To input a single node where the capacitance varies with temperature. For SIV, the capacitance is found by interpolation using an array of temperature vs. capacitance. For SPV, the capacitance is found by computing an Nth order polynomial function of temperature
SIM SPM	♦				To generate a group of nodes, each having the same initial temperature and the same temperature-varying capacitance. For SIM, C is found by interpolation using an array of T vs.C. For SPM, C is found by computing a polynomial in T
DIV DPV	♦				To input a single node consisting of two materials which have different temperature-varying capacitances. For DIV, C1 and C2 are taken from arrays of T vs. C. For DPV, C1 and C2 are computed from polynomials in T
DIM DPM	♦				To generate a group of nodes, each of which consists of the same two materials having temperature-varying capacitances. For DIM, C1 and C2 are taken from arrays of T vs. C. For DPM, C1 and C2 are computed from polynomials in T.
BIV	♦				To input a single node where the capacitance is a function of time and temperature. The capacitance is found by interpolation using an array of time and temperature vs. capacitance

6.5 Source Data

Block Format:

```
HEADER SOURCE DATA, smn
       source data cards
```

Reference Forms:

```
Q(n),Qn
```

where n is the actual node number.

Within expressions, a node can refer to its own data (T, Q, C) using “#this” instead of its ID.

Units:

Source . . . . . energy/time

Source Data Input Options:

```
N#,Q

GEN N#,#N,IN,Q

TVS N#,AS,AQ,F                  $ Q vs. time

SIV N#,At,F                     $ Q vs. Temp

TVD N#,AS,AQ,FA,AS,AQ,FA        $ Q1+Q2 vs. time

DTV N#,At,F1,AT,F2              $ Q vs. T & time
DTV N#,SUB,F1,AT,F2
DTV N#,At,F1,SUB,F2

PER N#,AS,AQ,F,D,P
```

where:

N# . . . . . Actual node number (positive integer)
Q. . . . . Value of source (floating point data value, expression, or reference to a user constant)
#N . . . . . Number of nodes (non-zero integer)
IN . . . . . Increment to the node number (non-zero integer)
AT . . . . . Reference to an array of temperature vs. heat rate points (integer count form)
At_n . . . . . References to arrays of mean time vs. heat rate points (integer count form). The first time point must be zero
F_n. . . . . Multiplying factors (floating point data value or expression)
AS . . . . . Reference to a singlet time array
AQ . . . . . reference to a singlet Q array
SUB . . . . . Value used as a substitute for an array reference (floating point data value or expression)
D. . . . . Phase displacement as a decimal fraction of the total period. (Must be a floating point literal with a value between O. and 1.0, or an expression.)
P. . . . . Period-specified in time units (floating point data value or expression). P is usually the last time point in the At array, but not restricted to this value. If P is less than the last array value, any values between P and the last array value are not used. If P is greater than the last array time value, the heating value used for the time period between the last array time value and P is constant and equal to the heating rate input for the last array time value.

Summary of SOURCE DATA Input Options

OPTION (CODE)	DESCRIPTION
3 blanks	To impress a constant heat source on a single node
GEN	To impress the same heat source on several nodes
SIV	To impress a temperature-varying heat source on a node
SIT†	To impress a time-varying heat source on a node
DIT†	To impress the sum of two time-varying heat sources on a node
TVS	To impress a time-varying heat source on a node
TVD	To impress the sum of two time-varying heat sources on a node
DTV	To impress the sum of a time-varying source and a temperature-varying source on a node
CYC†	To impress a cyclic time-dependent source on a node
PER	To impress a cyclic time-dependent source on a node

† The SIT, DIT, and CYC options have been replaced with the TVS, TVD, and PER options respectively. The obsolete options are included only for processing old SINDA files.

6.6 Conductor Data

Block Format:

```
HEADER CONDUCTOR DATA, smn [,fac]
       conductor data cards
```

Conductor Types:

LINEAR . . . . . heat transfer a function of (T_i - T_j)

RADIATION . heat transfer a function of (T_i^4 - T_j^4)

Restrictions:

a. Conductor number may be as large as 8 digits.

References:

```
G(n),Gn, HR(n), HRn
```

where n is an actual conductor number (always positive)

Within expressions, a conductor can refer to its own data (G) using “#this” instead of its ID. It can also refer to the parameters of the first node as “#nodeA” and the second node as “#nodeB.”

Units:

LINEAR . . . . . energy/time-degree

RADIATION . energy/time-degree^4

Conductance Calculations:

Conduction . . . kA/L

Convection . . . hA

Radiation. . . . . σFA

Mass Flow . . . mC_p

where:

k . . . . . conductivity (energy/time-length-degree)
A . . . . . area (length^2)
L . . . . . length
h . . . . . film coefficient (energy/time-length^2-degree)
m . . . . . mass flow rate (mass/time)
C_p . . . . . specific heat (energy/mass-degree)
σ . . . . . Stefan-Boltzmann constant (energy/time-length^2-degree^4)
F . . . . . Hottel "script F" or grey body radiation interchange factor

Conductor Data Input Options:

```
G#,NA,NB,G                                  $ One conductor
G#,NA1,NB1,NA2,NB2,... NAn,NBn,G            $ Multi-connected
GEN G#,#G,IG,NA,INA,NB,INB,G                $ Group of cond.

SIV G#,NA,NB,AP,F                           $ G vs T (interp)*
SPV G#,NA,NB,AC,F                           $ G vs T (poly)*
SIM G#,#G,IG,NA,INA,NB,INB,AP,F             $ GEN and SIV
SPM G#,#G,IG,NA,INA,NB,INB,AC,F             $ GEN and SPV
                                            $ * (Note 1)

SIVM G#,NA,NB,AP,F                          $ G vs T (interp)
SPVM G#,NA,NB,AC,F                          $ G vs T (poly)
SIMM G#,#G,IG,NA,INA,NB,INB,AP,F            $ GEN and SIVM
SPMM G#,#G,IG,NA,INA,NB,INB,AC,F            $ GEN and SPVM
                                            $ above use mean T

SIVA G#,NA,NB,AP,F                          $ G vs T (interp)
SPVA G#,NA,NB,AC,F                          $ G vs T (poly)
SIMA G#,#G,IG,NA,INA,NB,INB,AP,F            $ GEN and SIVA
SPMA G#,#G,IG,NA,INA,NB,INB,AC,F            $ GEN and SPVA
                                            $ above use Node A's T

DIV G#,NA,NB,APA,FA,APB,FB                  $ 2 mat'l (interp)
DIV G#,NA,NB,SUB,FA,APB,FB                  $ 2 mat'l GA=const
DIV G#,NA,NB,APA,FA,SUB,FB                  $ 2 mat'l GB=const

DPV G#,NA,NB,ACA,FA,ACB,FB                  $ 2 mat'l (poly)
DPV G#,NA,NB,SUB,FA,ACB,FB                  $ 2 mat'l GA=const
DPV G#,NA,NB,ACA,FA,SUB,FB                  $ 2 mat'l GB=const
```

```
DIM G#,#G,IG,NA,INA,NB,INB,APA,FA,APB,FB  $ GEN & DIV
DIM G#,#G,IG,NA,INA,NB,INB,SUB,FA,APB,FB
DIM G#,#G,IG,NA,INA,NB,INB,APA,FA,SUB,FB

DPM G#,#G,IG,NA,INA,NB,INB,ACA,FA,ACB,FB  $ GEN & DPV
DPM G#,#G,IG,NA,INA,NB,INV,SUB,FA,ACB,FB
DPM G#,#G,IG,NA,INA,NB,INB,ACA,FA,SUB,FB

BIV G#,NA,NB,AB,F                         $ G vs T & time
PIV G#,NA,NB,AT,FT,AAT,FAT                $ G=f(T & g(time))
PIV G#,NA,NB,SUB,FT,AAT,FAT               $ (Note 4)
PIV G#,NA,NB,AT,FT,SUB,FAT

PIM G#,#G,IG,NA,INA,NB,INB,AT,FT,AAT,FAT     $ GEN & PIV
PIM G#,#G,IG,NA,INA,NB,INB,SUB,FT,AAT,FAT    $ (Note 4)
PIM G#,#G,IG,NA,INA,NB,INB,AT,FT,SUB,FAT

TVS G#,NA,NB,AS,AG,F
PER G#,NA,NB,AS,AG,D,P
```

where:

- G# . . . . . Actual conductor number (non-zero integer) [Note 2]
- NA,NB . . . . . Actual node numbers of the nodes to which the conductor is connected (positive integer—see Note 3)
- G . . . . . Conductor value (floating point data value or expression) [Note 4]
- W,X,Y,Z . . . Values used to compute conductance: G = W*X*Y/Z (floating point data value or expression)
- #G . . . . . Total number of conductors to be generated (non-zero integer)
- IG . . . . . Increment to the conductor number (integer)
- INA,INB . . . Increments to the node number, NA and NB, respectively (integer)
- AP_A . . . . . Reference to arrays of temperature vs. conductance POINTS (integer count form)
- AC_A . . . . . References to arrays of polynomial *coefficients* for G = P(T).
- F_A . . . . . Multiplying factors (floating point data values, or expressions—see Note 1)
- SUB . . . . . Value used as a substitute for an array reference (floating point data value or expression).
- AT . . . . . Reference to doublet arrays of an arbitrary variable (A) versus time (t). A = f (t)
- AAT . . . . . Reference to bivariate arrays of conductance (G) versus a temperature variable (T) and an arbitrary variable (A). G = f(T,A). The temperature used in calculation is the average of NA and NB. [Note 6]

AS . . . . . .Reference to a singlet time array
AG . . . . . .Reference to a singlet G array
FT, FAT . . . . .Multiplication factors (floating point data values or expressions)

Notes:

1. Normally, the average temperature of nodes NA and NB is used to evaluate the conductance. To indicate that only the temperature of node NA is to be used, F must be entered as a floating point data value preceded by a minus sign, or as an expression yielding a negative value. The minus sign is considered only as a flag and is ignored in the arithmetic sense. *If an expression is used whose resulting value is positive (even if prefixed by a negative sign), this suboption is assumed to be inactive and the average temperature will be used.* If an expression containing register names is used, it must be preceded by a negative sign that, when stripped, yields a positive number of the same absolute value. Otherwise, an error will be produced. This feature applies only to the options noted (SIV, SPV, SIM, SPM).

   To avoid this convention, use the "M" or "A" suffix (e.g, SIVM, SPVA) to designate whether a mean ("M") or node A ("A") temperature should be used irregardless of the sign of F.

   The following table summarizes the differences between these conductor options

Type	GEN option	Method	F input	F used	Temp. Used
SIV	SIM	interpolation	+F	F	mean
SPV	SPM	polynomial	+F	F	mean
SIV	SIM	interpolation	-F	|F|	node A
SPV	SPM	polynomial	-F	|F|	node A
SIVM	SIMM	interpolation	any	F	mean
SPVM	SPMM	polynomial	any	F	mean
SIVA	SIMA	interpolation	any	F	node A
SPVA	SPMA	polynomial	any	F	node A

2. Conductor type is specified by the sign of the G# as follows:

   LINEAR . . . . . positive conductor number
   RADIATION . . . . . negative conductor number

3. The negative signs on node numbers which define boundary nodes are considered flags and are ignored in the arithmetic sense. Preceding NA or NB with a minus sign will cause the conductor to be treated as a one-way conductor. The "upstream" node should be identified with the minus sign, and will be unaware of the presence of the "downstream" node, but not vice versa.

4. The conductance G of all radiation conductors in all thermal submodels will be multiplied by the global control constant SIGMA (Section 6.7), whose default is unity (1.0). The Stefan-Boltzmann constant can be supplied as that constant (the built-in values *sbcon* and *sbconsi* being convenient, per Section 6.1), but the user should then assure

both that units for all radiation conductors are consistent with the chosen value of SIGMA, and that the Stefan-Boltzmann constant has not also been supplied twice as part of the G value (perhaps as calculated by another program).

5. In both the PIM and PIV options when:

   SUB,FT,AAT,FAT, is used, the arbitrary variable (A) is evaluated as A = SUB*FT.

   AT,FT,SUB,FAT is used, the arbitrary variable (A) is evaluated by interpolation. A = f(e); the conductance (G) is evaluated as G = A*FT*SUB*FAT.

6. Consistent with the bivariate array input format described in Section 2.11.2, the arbitrary variable (A) corresponds to the Y values and the temperature (T) corresponds to the X values. The units of the arbitrary variable entered into the bivariate array must be equal to the product of the arbitrary variable and the multiplication factor (FT) for the time dependent doublet array.

Summary of CONDUCTOR DATA Input Options

OPTION (CODE)	DESCRIPTION
"3 blanks"	To input single conductors
GEN	To generate several conductors with the same conductance
SIV, SIVM, SIVA SPV, SPVM, SPVA	To input single temperature-varying conductors. SIV uses array interpolation. SPV uses a polynomial function of temperature. The "M" or "A" suffices determine whether a mean temperature will be used, or just the temperature of node A.
SIM, SIMM, SIMA SPM, SPMM, SPMA	Same as SIV and SPV for groups of conductors
DIV DPV	To input single conductors that model two materials with different temperature-varying conductances. DIV uses two arrays of G vs. T, and DPV uses polynomial functions of T.
DIM DPM	Same as DIV and DPV for groups of conductors
BIV	To input a single conductor that is both time- and temperature-varying. The conductance is found by interpolation using a bivariate array.
PIV PIM	To input conductors where the conductance is a function of temperature and a time-independent variable. PIV is for single conductors, PIM is for multiple conductors
TVS	To input a time-varying conductor
PER	To input a cyclic time-varying conductor

6.7 Control Data

Block Format:

```
HEADER CONTROL DATA, GLOBAL
       control data cards
HEADER CONTROL DATA, smn
       control data cards
```

Restrictions:

a. GLOBAL data must be entered before any submodel data.
b. Only control constants may be entered in this block.

Reference Forms:

variable_name or submodel_name.variable_name

Control Data Input Options:

variable_name = value or expression

User-Specified Control Constants Required by Solution Routines

Name	Steady State STDSTL, STEADY	Transient TRANSIENT	FORWRD
TIMEO	0.0	0.0	0.0
TIMEND	0.0	0.0	0.0
NLOOPS	1000	NA	NA
DTIMES	0.0	NA	NA
ABSZRO**	-459.67 or -273.15*	-459.67 or -273.15*	-459.67 or -273.15*
PATMOS**	0.0	0.0	0.0
SIGMA**	1.0	1.0	1.0
ACCELX	0.0	0.0	0.0
ACCELY	0.0	0.0	0.0
ACCELZ	0.0	0.0	0.0
ARLXCA	0.01	0.01	0.01
NLOOPT	NA	100	100
DRLXCA	0.01	0.01	NA
EBALSA	0.01	NA	NA
EBALNA	0.0	NA	NA
ATMPCA	NA	1.0E30	1.0E30
DTMPCA	NA	1.0E30	1.0E30
CSGFAC	NA	1.0	1.0
DTIMEL	NA	0.0	0.0
DTIMEH	NA	1.0E30	1.0E30
DTIMEI	NA	0.0	NA
ITEROT	0	NA	NA
OUTPUT	NA	0.01*TIMEND	0.01*TIMEND
ITHOLD	0	NA	NA
EXTLIM	5.0	5.0	5.0
ITERXT	3	3	3
BACKUP	0	0	0
OPEITR	NA	0	0
NSOLOR	2	2	NA
NVARB1	NA	0	NA
MATMET	12	12	NA
AMGERR	1.0E-10	1.0E-10	NA
METAMG	0	0	NA
SPARSEG	0.001	0.001	NA
FBEBALA	NA	0.1	NA
DTSIZF	NA	0.1	0.1
DTTUBF	NA	0.1	0.1
DTMAXF	NA	1.0E30	1.0E30
DTMINF	NA	0.0	0.0
OUTPTF	NA	0.01*TIMEND	0.01*TIMEND
OPITRF	NA	0	0
RSSIZF	0.5 (STDSTL)	NA	NA
RSTUBF	0.5 (STDSTL)	NA	NA
RSMAXF	1 hour (STDSTL)	NA	NA
RERRF	0.01	NA	NA
REBALF	0.01	NA	NA
ITHLDF	0	NA	NA
ITROTF	0	NA	NA
FRAVER	0.	0.	0.
RMSPLT	0.1 (STDSTL)	0.5	0.5
RMFRAC	0.5 (STDSTL)	0.5	0.5
RMRATE	0.1 sec (STDSTL)	0.1 sec	0.1 sec

where:

NA	does not apply
*	default depends on value of UID
**	not intended to be changed in logic blocks or expressions

6.8 User Data

Block Format:

```
HEADER USER DATA, GLOBAL
       user data cards
HEADER USER DATA, smn
       user data cards
```

Restrictions:

a. GLOBAL data must be entered before any submodel data.
b. Only alphanumeric variables are allowed in the global block.
c. Integer variables must be initialized with integer data.
d. Character data is not allowed.
e. Only one GLOBAL block is allowed.
f. Variables that start with I-N will be integers, all other are real.

With a few exceptions, this option is anachronistic. Use registers instead if possible.

User Data Input Options Example:

```
HEADER USER DATA, GLOBAL
       ITEST  = 1
       AB     = 2.0
HEADER USER DATA, SMN
       ID#    = DV
       ID#,DV
       GEN ID#,#K,IK,DV
       GEN ID#,#K,IK,DVI,IDV
```

where:

`ID#` . . . . . Integer identification number
`#K` . . . . . the number of constants to be generated
`IK` . . . . . ID# increment
`DV` . . . . . data value
`SMN` . . . . . a submodel name

Reference Forms:

```
       Kn or K(n)     $ for integer reference
       XKn or XK(n)   $ for real reference
```

6.9 Array Data

Block Format:

```
HEADER CARRAY DATA, smn
        carry data cards
HEADER ARRAY DATA, smn
        array data cards
```

Restrictions:

a. Character data can only be entered in CARRAY DATA.
b. Hollerith data is not allowed.
c. Entries in CARRAY DATA are limited to 1000 characters.

Reference Forms:

```
        NAn or NA(n)   $ for integer reference
        AN  or A(n)    $ for real reference
        UCAn or UCA(n) $ for character reference
```

Array Data Input Options:

```
HEADER CARRAY DATA,smn
        array-number = a string
HEADER ARRAY DATA,smn
        array-number = dv1,dv2,dv3,dv4
```

6.10 Solver Data

Block Format:

```
HEADER SOLVER DATA
       control data cards
```

Reference Forms:

variable_name*

Solver Data Input Options:

variable_name = value

Name	Definition and Description	Default
	Input under SOLVER DATA and/or Updated in Logic	
NLOOPO	Maximum iterations (calls to PROCEDURE)	100
NLOOPR	Maximum iterations (calls to RELPROCEDURE)	100
OBJECT	Value of the design or optimization objective	NONE; real
GOAL	Desired value for OBJECT (fixed, -1.0E30 to minimize, 1.0E30 to maximize)	-1.0E30
METHO	Solver method to use	2
RERRO	Maximum *relative* error in OBJECT between design solutions to be converged	0.01 of current
AERRO	Maximum *absolute* error in OBJECT to be converged	0.0001 of initial
RERRR	Maximum *relative* error in responses between samples to be converged (SAMPLE only)	0.001 of current
AERRR	Maximum *absolute* error in in responses between samples to be converged (SAMPLE only)	0.00001
RDERO	Minimum *relative* size of perturbations to design variables for derivatives	0.02 of current
ADERO	Minimum *absolute* size of perturbations to design variables for derivatives	0.0001 of initial
MDERO	Method for estimating derivatives: 0=forward, 1=central (Solver only)	0 (forward)
RDERR	Minimum *relative* size of perturbations to random variables for derivatives (RELEST only)	0.02 of current
ADERR	Minimum *absolute* size of perturbations to random variables for derivatives (RELEST only)	0.01
MDERR	(Reserved for future use.)	N/A
RCHGO	Maximum *relative* size of changes to design variables during search phase	0.9 of current
ACHGO	Maximum *absolute* size of changes to design variables during search phase	0.1 of initial
RCACTO	Relative measure of proximity of an optimization constraint for it to be considered active	0.1 of current
RCERRO	Relative error tolerated in violation of an optimization constraint in a final solution	0.003 of current
RCVIO	Relative error tolerated in violation of a formula-based design variable limit	0.2 of current
PUSHO	Ambitiousness: less than 1.0 is cautious, greater than is aggressive (Solver only)	1.0

* GOAL, LOOPCO, LOOPCR, DELOBJ, NSTATO are for output/instpection only and should not be changed in logic. NWRKRO and NWRKIO are not available for references in logic or expressions.

Name	Definition and Description	Type
	Input under SOLVER DATA and/or Updated in Logic (Cont'd)	
NERVUS	Tolerance of problems: 0 (stop at first sign of trouble), 1 (tolerate converged, stable but unbalanced thermal), 2 (tolerate also nonconvergence in TRANSIENT), 3 (tolerate also nonconvergence in STDSTL and STEADY).	0 (abort at first sign of trouble)
NEWPRO	If NEWPRO=1, call PROCEDURE before SOLOUTPUT CALLS	0
NCONVO	Number of final search directions to check before assuming convergence (Solver only)	2
NWRKRO	Real workspace array size for NLP program (Solver)	calculated
NWRKIO	Integer workspace array size for NLP program (Solver)	calculated
NSEED	Integer seed for random number generation (SAMPLE, DSAMPLE only)	calculated
	Provided by Solver for Use in Logic	
LOOPCO	Current iteration count (calls to PROCEDURE)	integer
LOOPCR	Current iteration count (calls to RELPROCEDURE)	integer
DELOBJ	Last change in best value of OBJECT (positive or negative)	real
NSTATO	Status of Solver: -n: performing perturbation of the n^{th} design variable 0: finished or terminating +n: performing the n^{th} one dimensional search	integer
OVEREL	Overall reliability (SAMPLE, DSAMPLE only)	real

6.11 Design Data

Block Format:

```
HEADER DESIGN DATA
        definition of one design variable
        definition of another design variable
```

Restrictions:

a. Only one design variable may be defined per line
b. Design variables must be declared as real registers
c. At no time may the upper limit be less than the lower limit
d. Formula-based upper or lower limits are treated as constraints

Reference Forms:

register_name*

Design Data Input Options:

```
HEADER DESIGN DATA
        [ lower_limit <=] reg_name [<= upper_limit]
```

* Should not be changed in logic while the Solver is active.

6.12 Constraint Data

Block Format:

```
HEADER CONSTRAINT DATA
       definition of one constraint variable or expression
       definition of another constraint variable
```

Restrictions:

a. Names of optimization constraint variables must be unique.
b. Only one optimization constraint variable, expression, or GEN may be defined per line
c. At no time may the upper limit be less than the lower limit
d. Constraint variables are all real variables

Reference Forms:

variable_name*

Constraint Data Input Options:

```
HEADER CONSTRAINT DATA
       [ lower_limit <=] cst_name [<= upper_limit]
       [ lower_limit <=] expression [<= upper_limit]
       GEN basis,init,num,incr,lower_limit[,upper_limit]
```

* Constraint variables may not be referenced within expressions

6.13 Random Data

Block Format:

```
HEADER DESIGN DATA
        definition of one random variable
        definition of another random variable
```

Restrictions:

a. Only one random variable may be defined per line
b. Random variables must be declared as real registers
c. For UNIFORM, upper limit cannot be below lower limit
d. For NORMAL, sd or cv must be nonnegative
e. For ARRAY, probability values must be nonnegative, cumulative distribution function must be positive

Reference Forms:

register_name*

Design Data Input Options:

```
HEADER RANDOM DATA
        reg_name, UNIFORM, lower_limit, upper_limit
        reg_name, NORMAL {,mean=R}[, sd=R or cv=R]
        reg_name, ARRAY, smn.An
```

* Should not be changed in logic while the Reliability Engineering routines are active.

6.14 Reliability Constraint Data

Block Format:

```
HEADER RELCONSTRAINT DATA
        definition of one relconstraint variable or expression
        definition of another relconstraint variable
```

Restrictions:

a. Names of reliability constraint variables must be unique.
b. Only one reliability constraint variable or expression per line
c. Reliability constraint variables are all real variables

Reference Forms:

variable_name*

Reliability Constraint Data Input Options:

```
HEADER RELCONSTRAINT DATA
        [ lower_limit <=] rcst_name [<= upper_limit]
        [ lower_limit <=] expression [<= upper_limit]
```

Unlike optimization constraints, upper and lower limits are both optional for reliability constraints.

* Reliability constraint variables may not be referenced within expressions

6.15 Flow Data

Block Format:

```
HEADER FLOW DATA,smn[,FID=I][,FIDx=I][,NWORK=I]
       flow data cards
```

FLOW DATA Restrictions:

a. Lump and path numbers should be 7 digits or less.
b. Tanks and junctions must be connected to at least one but no more than 1,000 other lumps (excluding DUP factors).
c. No model can be built entirely of junctions.
d. No loop of diabatic junctions can exist.
e. Net flow through junctions must be nonzero if the QDOT is nonzero.
f. Model must contain at least one plenum, vapor-filled or compliant tank.
g. Flow Rate cannot be overconstrained along any line or through any junction.
h. PL, TL, VOL, FR, TLEN, DH have no defaults until set in a DEF subblock.
i. Up to 26 constituents per fluid submodel.
j. Constituent letter designators must be unique within a fluid submodel.
k. If more than one constituent is defined, only one may be a library fluid, a 6000 series two-phase fluid, or a 7000 series fluid. The rest must be 6000 NEVERLIQ (real gas), 8000 (perfect gas) or 9000 series (nonvolatile liquid) fluids.

6.15.1 LU (Lump) Subblocks

Lump Types:

Tank . . . . . Finite volume (resists mass and energy changes)
Junction. . . . . Zero volume (state can change instantaneously)
Plenum . . . . . Infinite volume (constant state)

Within expressions, lumps may refer to their own parameters (TL, CZ, etc.) as “#this” rather than using their ID. For twinned tanks, “#twin” can be used to refer to the secondary twin.

Lump Subblock Formats:

```
LU TANK,id[,VOL=R][,TL=R][,XL=R][,PL=R][,PL!=R]
        [,VDOT=R][,QL=R][,COMP=R][,XFx=R][,XGx=R]
        [,CX=R][,CY=R][,CZ=R]
        [,ASL=R][,GLx=R][,FRDx=R][,ZRx=R]
        [,ZJ=R][,PH=R][,FRHx=R]
        [,ULI=R][,UVI=R][,HENx=R][,IDGC=I]
        [,LTWIN=I][,GTx=R][,AST=R][,UVT=R][,ULT=R]
        [,VDRP=R][,VBUB=R][,IDFACE=I][,IDPATH=I][,IDFTIE=I]


LU JUNC,id[,TL=R][,XL=R][,PL=R][,PL!=R]
        [,QL=R][,XFx=R][,XGx=R]
        [,CX=R][,CY=R][,CZ=R][,VOL=R]
        [,ASL=R][,GLx=R][,FRDx=R][,ZRx=R]
        [,ZJ=R][,PH=R][,FRHx=R]
        [,ULI=R][,UVI=R][,HENx=R][,IDGC=I]

LU PLEN,id[,TL=R][,XL=R][,PL=R][,PL!=R][,XFx=R][,XGx=R]
        [,CX=R][,CY=R][,CZ=R]

LU DEF[,VOL=R][,TL=R][,TLADD=R][,TLFACT=R]
        [,XL=R][,PL=R][,PL!=R][,PLADD=R][,PLFACT=R]
        [,VDOT=R][,QL=R][,COMP=R][,XFx=R][,XGx=R]
        [,CX=R][,CY=R][,CZ=R]
        [,ASL=R][,GLx=R][,FRDx=R][,ZRx=R]
        [,ZJ=R][,PH=R][,FRHx=R]
        [,ULI=R][,UVI=R][,HENx=R][,IDGC=I]
        [,GTx=R][,AST=R][,UVT=R][,ULT=R][,VDRP=R][,VBUB=R]
```

6.15.2 PA (Path) Subblocks

Path Types:

Tube . . . . . .Finite inertia (resists flow rate changes)
Connector. . . . .Zero inertia (flow rate can change instantaneously), subclassified by device specification.

Within expressions, paths may refer to their own data (FR, GK, etc.) using "#this" instead of their IDs. The complete list of such operators is as follows for paths:

Operator	Meaning
#this	the identifier of the current path
#up	the identifier of the *defined* upstream lump (does not change with flow reversal)
#down	the identifier of the *defined* downstream lump (does not change with flow reversal)
#twin	the identifier of the twinned path (if applicable)

Path Subblock Formats:

```
PA TUBE,id,l1,l2[,FR=R][,TWIN=I][,STAT=C][,MCH=I],HCH=R]
        [,TLEN=R][,DH=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R]
        [,FK=R][,WRF=R][,IPDC=I][,UPF=R][,FTIE=AXIAL]
        [,HC=R][,FC=R][,FPOW=R][,AC=R][,AM=R][,FG=R][,FD=R]
        [,ASPU=R][,ASPD=R][,GUx=R][,GDx=R]
        [,IUAS=I][,IDAS=I][,FUAS=R][FDAS=R]
        [,UVPU=R][,ULPU=R][,UVPD=R][,ULPD=R]
        [,DUPI=R][,DUPJ=R][,DUP=R]
        [,DEFF=R][,CURV=R][,FC=R][,FCLM=R][,FCTM=R]
        [,ROTR=R][,RADI=R][,RADJ=R][,VAI=R][,VAJ=R]
        [,VXI=R][,VXJ=R][,RVR=R][,EFFP=R]
        [,TCF=R][,TORQ=R][,INTORQ=C]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=R]

PA CONN,id,l1,l2[,FR=R][,MCH=I],HCH=R][,TWIN=I][,STAT=C]
        [,GK=R][,HK=R][,EI=R][,EJ=R][,DK=R]
        [,DUPI=R][,DUPJ=R][,DUP=R]
        [,ROTR=R][,RADI=R][,RADJ=R][,VAI=R][,VAJ=R]
        [,VXI=R][,VXJ=R][,RVR=R][,EFFP=R]
        [,TCF=R][,TORQ=R][,INTORQ=C]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=R]
        [,DEV=C,...,...]
```

```
PA DEF  [,FR=R][,FRFACT=R][,STAT=C][,MCH=I],HCH=R]
        [,TLEN=R][,DH=R][,DEFF=R][,AF=R][,AFTH=R]
        [,WRF=R][,IPDC=I][,UPF=R][,FTIE=AXIAL]
        [,HC=R][,FC=R][,FPOW=R][,AC=R][,AM=R][,FG=R][,FD=R]
        [,GK=R][,HK=R][,EI=R][,EJ=R][,DK=R]
        [,ASPU=R][,ASPD=R][,GUx=R][,GDx=R]
        [,IUAS=I][,IDAS=I][,FUAS=R][FDAS=R]
        [,UVPU=R][,ULPU=R][,UVPD=R][,ULPD=R]
        [,DEFF=R][,CURV=R][,FC=R][,FCLM=R][,FCTM=R]
        [,ROTR=R][,RADI=R][,RADJ=R][,VAI=R][,VAJ=R]
        [,VXI=R][,VXJ=R][,RVR=R][,EFFP=R]
        [,TCF=R][,TORQ=R][,INTORQ=C]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=R]
```

Device Formats within CONN Subblocks:

```
... DEV=STUBE[,TLEN=R][,DH=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R]
              [,FK=R][,WRF=R][,IPDC=I][,UPF=R][,FTIE=AXIAL]
              [,HC=R][,FC=R][,FPOW=R][,AC=R][,FG=R][,FD=R]
              [,ASPU=R][,ASPD=R][,GUx=R][,GDx=R]
              [,IUAS=I][,IDAS=I][,FUAS=R][FDAS=R]
              [,UVPU=R][,ULPU=R][,UVPD=R][,ULPD=R]
              [,DEFF=R][,CURV=R][,FC=R][,FCLM=R][,FCTM=R]

... DEV=REDUCER[,MODEC=I]
              [,DHI=R or AFI=R][,DHJ=R or AFJ=R]
              [,RAD_R=R][,RAD_A=R][,TLEN=R][,WRF=R]
              [,AFTH=R][,FKI=R][,FKJ=R][,HC=R][,FF=C]

... DEV=EXPANDER[,MODEC=I]
              [,DHI=R or AFI=R][,DHJ=R or AFJ=R]
              [,RAD_R=R][,RAD_A=R][,TLEN=R][,WRF=R]
              [,AFTH=R][,FKI=R][,FKJ=R][,HC=R][,FF=C]

... DEV=CAPIL,RC=R,CFC=R[,XVH=R][,XVL=R]

... DEV=LOSS,FK=R[,AF=R][,AFTH=R][,TPF=R]

... DEV=LOSS2,FK=R[,FKB=R][,AF=R][,AFTH=R][,TPF=R]

... DEV=CHKVLV,FK=R[,AF=R][,AFTH=R][,TPF=R]

... DEV=CTLVLV,FK=R[,AF=R][,AFTH=R][,TPF=R]
```

```
... DEV=UPRVLV,PSET=R[,AF=R][,AFTH=R][,TPF=R]
              [,FKH=R][,FKL=R][,FK=R][,RLAG=R]

... DEV=DPRVLV,PSET=R[,AF=R][,AFTH=R][,TPF=R]
              [,FKH=R][,FKL=R][,FK=R][,RLAG=R]

... DEV=ORIFICE, AORI=R[,AF=R][,AFTH=R]
              [CDIS=R][,FK=R][,ELLD=R][,MODO=I] AORI=R
              [,MODA=I] [,PSET=R]
              [,AORI_L=R] [,AORI_H=R]
              [,AORI_C=R] [,AORI_T=R]

... DEV=MFRSET[,SMFR=R]

... DEV=VFRSET,SVFR=R

... DEV=PUMP,HG=A [,GU=I][,HU=I]
      [,COEFS=C][,GCF=R][,HCF=R][,ISTP=C]
      [,HG=A][,SPD=R][,RSPD=R][,GPMP=R][,DGDH=R]
      [,SPEEDS=A,HLIST=A,GLIST=A][,ELIST=A]
      [,EFFP=R or AEFFP=A]
      [,VCOR=R][,HVF=R][,GVF=R][,EFFVF=R]
      [,TCF=R][,SSCF=R][,RNPSH=R][,RNSS=R]
      [,CCE=R or ACCE=A][,CCH=R or ACCH=A]
      [,DEGVF=A][,AF=R][,AFTH=R][,AFI=R,AFJ=R][,UPF=R]

... DEV=TABULAR,  [,AF=R][,AFTH=R][,AFI=R,AFJ=R]
      [,HG=I] or [,GLIST=I,HLIST=I]
              or [,HGTABLE=I] or [,GHTABLE=I]
      [,HU=I][,GU=I] [,ENLT=R][,OFAC=R][,OMULT=R] [,SYM=C]
      [,ISTP=C][,MONOH=C][,UPF=R][,NAF=I][,GCF=R]
      [,MREF=I][,TREF=R][,PREF=R][,GREF=R]
      [,TLIST=I][,QLIST=I]
              or [,THTABLE=I][,QHTABLE=I]
              or [,TGTABLE=I][,QGTABLE=I]
```

```
... DEV=TURBINE [,GU=I][,HU=I]
      [,GCF=R][,ISTP=C][,NAF=I]
      [,MREF=I][,TREF=R][,PREF=R][,GREF=R]
      [,GH=A][,SPD=R][,NSPD=I][,DTURB=R]
      [,GTURB=R][,DGDH=R]
      [,SPEEDS=A,GLIST=A,HLIST=A or HLIST1=A][,NSPD=I]
      [,EFFP=R or ELIST=A or ELISTR=A or PLIST=A or AEFFP=A]
            [,EFFM=R][,EFFA=R]
      [,TCF=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R][,UPF=R]

... DEV=COMPRESS [,GU=I][,HU=I]
      [,GCF=R][,ISTP=C][,NAF=I]
      [,MREF=I][,TREF=R][,PREF=R][,GREF=R]
      [,HG=A][,SPD=R][,NSPD=I]
      [,GCOMP=R][,DGDH=R]
      [,SPEEDS=A,GLIST=A,HLIST=A][,NSPD=I]
            [,EFFP=R or ELIST=A or PLIST=A or AEFFP=A]
            [,EFFM=R][,EFFA=R]
      [,GSURGE=A or HSURGE=A][,SSCF=R]
      [,TCF=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R][,UPF=R]

... DEV=COMPPD [,GU=I][,HU=I]
      [,GCF=R][,ISTP=C],SPD=R [,DISP=R]
      [,EFFV=R or AEFFV=A or ELISTV=A,SPEEDS=A,HLIST=A]
      [,EFFVM=R][,EFFVA=R]
            [,EFFP=R or ELIST=A or PLIST=A or AEFFP=A]
            [,EFFM=R][,EFFA=R]
      [,TCF=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R][,UPF=R]
```

GU	Meaning	Cautions
0	default: m^3/s if uid=si, ft^3/hr if uid=eng	
1	m^3/s	
2	m^3/min	
3	m^3/hr	
4	L/s	
5	L/min	
6	L/hr	
7	cm^3/s	
8	cm^3/min	
9	cm^3/hr	
10	mm^3/s	
11	mm^3/min	

GU	Meaning	Cautions
12	mm^3/hr	
13	kg/s	mass flow rate units
14	kg/min	mass flow rate units
15	kg/hr	mass flow rate units
16-100	*RESERVED*	
101	ft^3/s (CFS)	
102	ft^3/min (CFM)	
103	ft^3/hr (CFH)	
104	gal/s (US GPS)	
105	gal/min (US GPM)	
106	gal/hr (US GPH)	
107	in^3/s	
108	in^3/min	
109	in^3/hr	
110	lb_m/s	mass flow rate units
111	lb_m/min	mass flow rate units
112	lb_m/hr	mass flow rate units

HU	Meaning	Cautions
0	default: m if uid=si, ft if uid=eng	
1	m	
2	cm	
3	mm	
4	cmH_2O, 4°C	pressure units
5	mmH_2O, 4°C	pressure units
6	mmHg, 0°C (Torr)	pressure units
7	Pa	pressure units
8	kPa	pressure units
9	Bar	pressure units
10-100	*RESERVED*	
101	ft	
102	in	
103	inH_20 (39.2°F)	pressure units
104	inHg (32°F)	pressure units
105	psi	pressure units
106	psf	pressure units
201	(absolute) pressure ratio	TABULAR, TURBINE, COMPRESS, or COMPPD only
202	Martin Beta Factor	TABULAR only

6.15.3 T (Tie) Subblocks

Tie Types:

HTN . . . . . Links lump to node with all or part of one path representing size and shape for forced convection heat transfer

HTNC . . . . . Links lump to node with all or parts of two paths representing size and shape for forced convection heat transfer (centered lump)

HTU . . . . . Links lump to node with user-specified UA conductance

HTNS . . . . . Links path to node representing average wall temperature over that segment, path represents size and shape for forced convection heat transfer; heat applied to downstream lump

HTUS . . . . . Links path to node representing average wall temperature over that segment, user-described heat transfer coefficient; heat applied to downstream lump

HTP. . . . . Lunks lump to node to represent quasi-stagnant convective heat transfer, including pool boiling

Note: Any named path can also represent the primary twin of a pair of twinned paths, in which case the involvement of the secondary path is implied. Secondary twins, however, cannot be named directly.

Within expressions, ties may refer to their own data (UA, QTIE, etc.) using “#this” instead of their IDs. The complete list of such operators is as follows for ties:

Operator	Meaning
#this	the identifier of the current tie
#node	the identifier of the attached node[a]
#lump	the identifier of the attached lump (not valid for HTUS, HTNS)
#path	the identifier of the path (not value for HTU)
#path2	the identifier of the second path (only valid for HTNC)
#twin	the identifier of the twinned tie (if applicable)

a. The submodel prefix *cannot* be used in combination with this operator.

Tie Subblock Formats:

```
T HTN,tid,lid,nid,pid[,f][,DUPN=R][,DUPL=R][,TTWIN=I]
        [UAM=R][,MODW=I][,UVEL=R][,IPUV=I][,DTEF=R][MODR=I]
        [XNUL=R][,XNLM=R][,XNLA=R][,XNTM=R][,XNTA=R]
        [CDB=R][,UER=R][,UEC=R][,UEH=R][,UEV=R][,UED=R]
        [UEK=R][,UECP=R][,UEP=R][,UET=R][,RLAM=R][,RTURB=R]
        [CHFF=R][,HFFL=R][,XNB=R]
```

```
T HTNC,tid,lid,nid,pid,f1,p2id[,f2][,TTWIN=I]
        [DUPN=R][,DUPL=R][,TTWIN=I][,UAM=R][,MODW=I]
        [UVEL=R][,IPUV=I][,DTEF=R][,MODR=I]
        [XNUL=R][,XNLM=R][,XNLA=R][,XNTM=R][,XNTA=R]
        [CDB=R][,UER=R][,UEC=R][,UEH=R][,UEV=R][,UED=R]
        [UEK=R][,UECP=R][,UEP=R][,UET=R][,RLAM=R][,RTURB=R]
        [CHFF=R][,HFFL=R][,XNB=R]

T HTU,tid,lid,nid,[UA=R][,DUPN=R][,DUPL=R]
        [UAM=R][,AHT=R][,UB=R][,UEDT=R]
        [MODW=I][,UVEL=R][,IPUV=I][,DTEF=R]
        [TTWIN=I][,ITAC=I][,DEPTH=R][,XOL=R][,XOH=R]

T HTNS,tid,pid,nid,[,f][,DUPN=R][,DUPL=R][,STAY][,TTWIN=I]
        [UAM=R][MODW=I][,UVEL=R][,IPUV=I][,DTEF=R][,MODR=I]
        [XNUL=R][,XNLM=R][,XNLA=R][,XNTM=R][,XNTA=R]
        [CDB=R][,UER=R][,UEC=R][,UEH=R][,UEV=R][,UED=R]
        [UEK=R][,UECP=R][,UEP=R][,UET=R][,RLAM=R][,RTURB=R]
        [CHFF=R][,HFFL=R][,XNB=R]

T HTUS,tid,pid,nid,[UA=R][,DUPN=R][,DUPL=R][,STAY][,UAM=R]
        [MODW=I][,UVEL=R][,IPUV=I][,DTEF=R]
        [AHT=R][,UB=R][,UEDT=R]
        [TTWIN=I][,ITAC=I][,DEPTH=R][,XOL=R][,XOH=R]

T HTP,tid,lid,nid,[,iport=I[,fport=R]],AHT=R
        [,DUPN=R][,DUPL=R]
        [,TTWIN=I][,ITAC=I][,XOL=R][,XOH=R]
        [,UAM=R][,MODW=I][,UVEL=R][,IPUV=I][,DTEF=R]
        [,MODR=I][,XNUL=R][,XNLM=R][,XNLA=R][,DCH=R]
        [,CSF=R][,SPR=R][,CHFF=R][,HFFL=R][,XNB=R]
        [,THKL=R][,HCON=R][,ACCM=R][,DEPTH=R]
```

6.15.4 FT (Ftie) Subblocks

Ftie Types:

USER. . . . . Links two lumps together with user-specified GF conductance

AXIAL. . . . . Links two lumps together, using a tube or STUBE (usually between the two lumps) to define the GF internal calculation based on axial conduction

CONSTQ . . . . . Extracts heat from one lump and injects it into another lump at the rate QF

Within expressions, fties may refer to their own data (GF, QF, etc.) using “#this” instead of their IDs. The complete list of such operators is as follows for fties:

Operator	Meaning
#this	the identifier of the current ftie
#lumpC	the identifier of the first lump
#lumpD	the identifier of the second lump
#path	the identifier of the path (for AXIAL fties only)
#twin	the identifier of the twinned ftie (if applicable)

Ftie Subblock Formats:

```
FT USER,id,lc,ld, GF=R, [,DUPC=R][,DUPD=R][,FTWIN=I]

FT AXIAL,id,lc,ld, IDTUBE=I [,DUPC=R][,DUPD=R][,FTWIN=I]

FT CONSTQ,id,lc,ld, QF=R [,DUPC=R][,DUPD=R]
```

6.15.5 IF (iface) Subblocks

Interface Types:

FLAT. . . . . Maintains equal pressures (neglecting inertial effects if EMA>0)
OFFSET . . . . . Maintains pressure difference at a constant value
SPRING . . . . . Follows a force vs. length (pressure vs. volume) relationship
WICK . . . . . Simulates a curved meniscus and perhaps a portion of porous material
SPHERE . . . . . Maintains pressure difference assuming a spherical vapor bubble
NULL . . . . . Direct access to VA, VB, PA, PB, and FPV: for advanced users only.

Within expressions, ifaces may refer to their own data (VHI, FPV, etc.) using “#this” instead of their IDs. The complete list of such operators is as follows for ifaces:

Operator	Meaning
#this	the identifier of the current iface
#tankA	the identifier of tank A (the reference tank)
#tankB	the identifier of tank B

Interface Subblock Formats:

```
IF FLAT,ifid,laid,lbid [,DUPA=R][,DUPB=R][,EMA=R]
       [,VLO=R][,VHI=R]

IF OFFSET,ifid,laid,lbid [,DUPA=R][,DUPB=R][,EMA=R]
       [,VLO=R][,VHI=R][,DPAB=R]
```

```
IF SPRING,ifid,laid,lbid [,DUPA=R][,DUPB=R][,EMA=R]
        [,VLO=R][,VHI=R][,DPDV=R][,VZRO=R]

IF WICK,ifid,laid,lbid [,DUPA=R][,DUPB=R][,EMA=R]
        [,VLO=R][,VHI=R][,XIL=R][,XIV=R]
        [,RCAP=R][,RBP=R][,VZRO=R][,VSUB=R]
        [,ICAP=I][,VAPOR=C]

IF SPHERE,ifid,laid,lbid [,DUPA=R][,DUPB=R][,EMA=R]
        [,VLO=R][,VHI=R][,XIL=R][,XIV=R]

IF NULL,ifid,laid,lbid [,DUPA=R][,DUPB=R]
        [,VA=R][,VB=R][,PA=R][,PB=R][,FPV=R]
```

6.15.6 M (Macro) Subblocks

Macro Types:

GENLU . . . . . Lump generator
GENPA . . . . . Path generator
GENT. . . . . Tie generator
GENFT . . . . . Ftie generator
GENIF . . . . . Iface generator
CAPPMP . . . . . Capillary evaporator-pump model
LINE . . . . . Generates a duct model with no ties
HX . . . . . Generates a duct model with one row of ties

Lump, Path, Ftie, and Iface Generator Subblock Formats:

```
M GENLU,mid,type,ilid[,N=I][,LUINC=I][,LTWIN=I]
        [,TLINC=R][,PLINC=R][,XLINC=R][,VOLINC=R]
        [,QDINC=R][,VDINC=R][,CXINC=R][,CYINC=R][,CZINC=R]
        [,LTINC=I][,IFINC=I][,IDPINC=I][,IDFTNC=I]
        further/initial lump specifications ...

M GENPA,mid,type,ipid,il1id,il2id[,N=I][,PAINC=I]
        [,L1INC=I][,L2INC=I][,FRINC=R]
        further/initial path specifications ...

M GENFT,mid,type,iftid,il1id,il2id[,N=I][,FTINC=I]
        [,L1INC=I][,L2INC=I][,TUBINC=I]
        further/initial ftie specifications ...
```

Generic Tie Generator Subblock Format:

```
M GENT,mid,type,itid,ilpid,inid {,fs}{,ipid,f1,ip2id,f2}
        [,N=I][,LUINC=I][,PAINC=I][,PA2INC=I][,NINC=I]
        [,TIINC=I][,TTINC=I]
        further/initial tie specifications ...
```

Type-Specific Tie Generator Subblock Formats:

```
M GENT,mid,HTU,itid,ilid,inid, UA = R
        [,N=I][,LUINC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HTUS,itid,ipid,inid, UA = R
        [,N=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R][,STAY]

M GENT,mid,HTN,itid,ilid,inid,ipid [,f1]
        [,N=I][,LUINC=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HTNS,itid,ipid,inid [,fs]
        [,N=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R][,STAY]

M GENT,mid,HTNC,itid,ilid,inid,ipid,f1,ip2id [,f2]
        [,N=I][,LUINC=I][,PAINC=I][,PA2INC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HTP,itid,ilid,inid, AFT = R
        [,N=I][,LUINC=I][,NINC=I]
        [,TIINC=I][,TTINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HXC,itid,ilid,inid,ipid [,f1]
        [,N=I][,LUINC=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,DUPN=R][,DUPL=R]
```

Component Model Subblock Formats:

```
M CAPPMP,mid,opt,ucid,jid,dcid,ulid,dlid,
        {,tid,tsmn.nid},RC=R,CFC=R[,XVH=R][,XVL=R]
        [,TL=R][,PL=R][,XL=R][,QL=R][,FR=R]
        [,DUPI=R][,DUPJ=R][,DUP=R][,DUPN=R][,DUPL=R]
        [,VAPOR=I]{,UA=R}[,AF=R][,AFTH=R][,MCH=I],HCH=R]
```

Duct Generator Subblock Formats:

```
M LINE,mid,opt,ilid,ipid,l1id{,l2id}[,NSEG=I][,STAT=C]
        [,TLENT=R][,DHS=R][,AFS=R][,AFI=R,AFJ=R]
        [,LU=C][,PA=C] [,PL(!)=R][,TL=R][,XL=R][,QL=R]
        [,LUINC=I][,TLINC=R][,PLINC=R][,XLINC=R][,QDINC=R]
        [,PAINC=I][,TWIN=I]
        [,CX=R][,CXINC=R][,CY=R][,CYINC=R][,CZ=R][,CZINC=R]
        [,DUPI=R][,DUPJ=R][,DUP=R][,LTWIN=I]
        [,LTINC=I][,IFINC=I][,IDPINC=I][,IDFTNC=I]
        further lump and path tie specifications ...

M HX,mid,opt,ilid,ipid,itid,inid,l1id{,l2id}[,NSEG=I]
        [,NINC=I][,STAT=C][,TLENT=R][,DHS=R][,AFS=R]
        [,AFI=R,AFJ=R]
        [,LU=C][,PA=C]
        [,LUINC=I][,TL=R][,PL(!)=R][,XL=R][,QL=R]
        [,TLINC=R][,PLINC=R][,XLINC=R][,QDINC=R][,PAINC=I]
        [,CX=R][,CXINC=R][,CY=R][,CYINC=R][,CZ=R][,CZINC=R]
        [,TWIN=I][,DUPI=R][,DUPJ=R][,DUP=R][,DUPN=R]
        [,DUPL=R][,AFRACT=R][,TIINC=I][,STAY][,LTWIN=I]
        [,LTINC=I][,IFINC=I][,IDPINC=I][,IDFTNC=I][,TTINC=I]
        further lump, path, and tie descriptions ...
```

6.16 FPROP Data

Header Subblock Format:

```
HEADER FPROP DATA, Nnnn [,uid [,abszro] ]
       [,keyword=value][,keyword=value]
```

Array Subblock Format:

AT, X, t_1,v_1, t_2,v_2 ... ,t_i,v_i, ... t_n,v_n

FPROP DATA Restrictions:

a. Fluid ID (Nnnn) must be from 6000 to 9999.
b. Local uid must be input in order to input local abszro.
c. All reference-style single-phase inputs use same TREF. For 7000 series fluids, all two-phase vapor properties use the same TREFG, all two-phase liquid properties use the same TREFL.
d. Array-style inputs take priority over reference-style.
e. Temperatures in array data must increase monotonically.
f. All input properties must be in consistent units and positive.

6.16.1 Perfect Gas (8000 Series)

Keywords in Header Subblock:

```
[RGAS=R or MOLW=R][,DIFV=R][,TREF=R]
[,TMIN=R][,TMAX=R]
[,PMIN=R][,PMAX=R]
{,K=R}  [,KTC=R]
{,CP=R} [,CPTC=R]
{,V=R}  [,VTC=R]
[,TCRIT=R] [,PCRIT=R]
[,HFORM=R] [,HSTP=R]
[,WSRK=R] [,VNB=R] [,VSTAR=R]
```

Optional Array Subblocks:

```
{AT,K, ...}
{AT,CP, ...}
{AT,V, ...}
{AT,PSAT,t1,p1}
```

6.16.2 Incompressible Liquid (9000 Series)

Keywords in Header Subblock:

```
        [,TREF=R][,MOLW=R]
        [,TMIN=R][,TMAX=R]
        {,K=R}  [,KTC=R]
        {,CP=R} [,CPTC=R]
        {,V=R}  [,VTC=R]
        {,D=R}  [,DTC=R]
        [,ST=R] [,STTC=R]
        [,COMP=R][,COMPTC=R]
        [,TCRIT=R] [,PCRIT=R]
        [,HFORM=R] [,HSTP=R]
        [,WSRK=R] [,PHI=R]
```

Optional Array Subblocks:

```
{AT,K, ...}
{AT,CP, ...}
{AT,V, ...}
{AT,D, ...}
{AT,ST, ...}
{AT,COMP, ...}
```

6.16.3 Simplified Two-Phase (7000 Series)

Keywords in Header Subblock:

```
HEADER FPROP DATA, 7nnn, [,uid [,abszro] ]
        RGAS=R, [,MOLW=R] [,DIFV=R] TCRIT=R, PCRIT=R
        [,TREFG=R] [,TREFL=R]
        {,KG=R} [,KGTC=R]{,KL=R} [,KLTC=R]
        {,VG=R} [,VGTC=R]{,VL=R} [,VLTC=R]
        {,CPG=R} [,CPGTC=R]{,DL=R} [,DLTC=R]
        {,ST=R} [,STTC=R]
        [,AVDW=R] [,BVDW=R]
        [,TMIN=R] [,TGMAX=R] [,PGMAX=R]
        [,HFORM=R] [,HSTP=R]
        [,WTRUE=R] [,WSRK=R] [,PHI=R]
```

Required Array Subblock:

```
AT,DOME,t1,p1,hfg1,t2,p2,hfg2
```

Optional Array Subblocks:

```
{AT,KG, ...}
{AT,KL, ...}
{AT,VG, ...}
{AT,VL, ...}
{AT,CPG, ...}
{AT,DL, ...}
{AT,ST, ...}
```

6.16.4 Advanced Two-Phase (6000 Series)

Keywords in Header Subblock:

```
HEADER FPROP DATA, 6nnn, [uid [,abszro] ]
       [,NEVERLIQ] [,COMPLIQ]
       PCRIT=R [,TCRIT=R] [,MOLW=R] [,DIFV=R]
       [,TMIN=R] [,TGMAX=R] [,PGMAX=R]
       [,HFORM=R] [,HSTP=R]
       [,WTRUE=R] [,WSRK=R] [,PHI=R]
```

Reserved Names in Functional Subblocks:

T. . . . . .Temperature, local units as selected on the HEADER card
P. . . . . .Pressure, *absolute* local units
H . . . . . .Enthalpy, local units
S. . . . . .Entropy, local units
D . . . . . .Density, local units
V . . . . . .Specific volume, local units
CP . . . . . .Specific heat, local units
U . . . . . .Internal energy, local units
HFG. . . . . .Heat of vaporization, local units
X . . . . . .Quality (vapor mass fraction)
A . . . . . .Void fraction (vapor volume fraction)
VISC . . . . . .Viscosity, local units
COND . . . . . .Conductivity, local units
SOS . . . . . .Speed of sound, local units
METH . . . . . .Two-phase speed of sound method: 1 or 2; *integer*
ST. . . . . .Surface tension, local units
DPDT. . . . . .Derivative of sat. pressure w.r.t. temperature: (dP/dT)sat
DPARG, SPARG, and INTARG are also reserved

Optional First Functional Subblock:

```
VINIT
```

Required Functional Subblocks (always):

```
VSV
VH
VVISCV
VCONDV
```

Required Functional Subblocks (unless the NEVERLIQ option is used):

```
VPS
VDPDTT
VDL
VVISCF
VCONDF
VST
```

Required Functional Subblocks (if the COMPLIQ option is used):

```
VDLC
VHLIQ
```

Optional Functional Subblocks, Additional Properties:

VS
VSOS
VTAV1
VQUALS
VDLC
VMOLWPT

Optional Subblocks, Additional Properties (if the COMPLIQ option is used):

VSLIQC

Optional Functional Subblocks, More Appropriate Methods:

VTAV2
VTS
VDPDT
VCPV
VCPF
VHLIQ
VTLIQ
VULIQ
VQUALH
VHFG
VALPHA
VSOSF
VSOSFV

Optional Subblocks, More Appropriate Methods (if COMPLIQ is used):

VULIQC

6.17 MIXTURE Data

Header Subblock Format:

```
HEADER MIXTURE DATA, BIN, Nnnn, Mmmm [,uid [,abszro] ]
       [,TREF=R] [,PREF=R] [,ZJ=R] [,UNITS=I]
       [,HL=R] [,HTC=R]
       [,OC=R] [,OTC=R] [,OPC=R]
       [,RL]
```

MIXTURE DATA Restrictions:

a. Most FPROP DATA rules apply.
b. One fluid (solute) must be an 8000 series, the other (solvent) can be a 6000, 7000, 9000, or library fluid. More than one solute or solvent may exist in any fluid submodel.
d. UNITS (of Henry's constant):

1 = atm/(mole-solute/mole-solvent)
2 = cc/gm (cc at standard temperature, pressure)
3 = cc/gram-mole (cc at standard temperature, pressure)
4 = mass fraction
5 = mole fraction

Optional Array subblocks

A bivariate array can used to input the either the values of Henry's constant or the Ostwald coefficient:

The format of optional bivariate arrays is:

$$
\begin{aligned}
&\texttt{AT, type, n, } p_1, p_2, \ldots, p_n, \\
&\quad t_1, x_{11}, x_{12}, \ldots, x_{1n}, \\
&\quad t_2, x_{21}, x_{22}, \ldots, x_{2n}, \\
&\quad \vdots \quad\quad \vdots \quad\quad \vdots \\
&\quad t_m, x_{m1}, x_{m2}, \ldots, x_{mn}
\end{aligned}
$$

where:

type . . . . . HL for Henry's constants, or OC for Ostwald coefficients
n . . . . . the number of pressure points that will be input
p_n . . . . . pressure input
t_n . . . . . temperature input
x_{mn} . . . . . solubility data (i.e., Henry's constant or the Ostwald coefficient) at (t_n, p_n)

6.18 Logic Blocks

6.18.1 Block Formats

```
HEADER OPERATIONS
        operations statements
HEADER PROCEDURE
        solver procedure statements
HEADER RELPROCEDURE
        reliability engineering procedure statements
HEADER VARIABLES 0, smn
        variables 0 statements
HEADER VARIABLES 1, smn
        variables 1 statements
HEADER VARIABLES 2, smn
        variables 2 statements
HEADER FLOGIC 0, smn
        flow logic 0 statements
HEADER FLOGIC 1, smn
        flow logic 1 statements
HEADER FLOGIC 2, smn
        flow logic 2 statements
HEADER OUTPUT CALLS, smn
        output operations
HEADER SOLOGIC 0
        solver logic 0 statements
HEADER SOLOGIC 1
        solver logic 1 statements
HEADER SOLOGIC 2
        solver logic 2 statements
HEADER SOLOUTPUT CALLS
        solver output operations
HEADER RELOUTPUT CALLS
        reliability engineering output operations
HEADER SUBROUTINES
        user subroutines
```

6.18.2 F-Type FORTRAN Statements

Permissible Arguments:

a. any acceptable FORTRAN statement.
b. T(m), Q(m), C(m), K(m), G(m), HR(m), A(m), XK(m), NA(m), UCA(m) and FLUINT names, where m = any valid FORTRAN subscript, i.e., a relative (internal) number.

Record Format:

```
1       7
Fsn     statement
```

where `sn` is a statement number.

If a statement must be continued on a second card, this card must have any character, except a blank or a zero, in column 6. Lines cannot be longer than 132 characters.

6.18.3 M-Type FORTRAN Statements

Permissible Arguments:

a. Tn, T(n), Qn, Q(n), Cn, C(n), Gn, G(n), Kn, K(n), An, A(n), HRn, HR(n), XKn, XK(n), NAn, NA(n), UCAn, UCA(n) and FLUINT names where n = actual (user) number
b. any control constant name
   any of the above arguments can (sometimes must) be preceded by a submodel name: smn.T(100) or smn.ARLXCA etc.
c. any data value
d. any register name or named user constant
e. any argument permissible in FORTRAN
f. GLOBAL.reserved(m)
g. smn.reserved(n)
h. GLOBAL.A(m+e) or GLOBAL.NA(m+e)
i. smn.A(n+e) or smn.NA(n+e)

where:

reserved . . . . . T, Q, C, K, G, HR, A, XK, NA, UCA, FLUINT variable, or control constant
e . . . . . any M-type expression
m . . . . . a relative (internal) index
n . . . . . an actual (user ID) number

Record Format:

```
1          7
sn         statement
```

where `sn` is a statement number.

6.18.4 FLUINT Variables

Lumps:

All . . . . . PL, TL, HL (output only), DL (output only), XL, CX, CY, CZ, AL (output only), DF (output only), DG (output only), XGx (output only), XFx (output only), PPGx (output only), MFx (output only), where “x” is the letter identifier of the constituent

Junc/tank . . . . . QDOT (output only), QL, QTM (output only), VOL, ASL, CASL (output only), GLx, CGLx, FRDx, ZRx, ZJ, PH, FRHx, ULI ,UVI, CULI (output only), CUVI (output only), HENx, or IDGC, where “x” is the letter identifier of the constituent

Tank . . . . . VDOT, COMP, AST, CAST (output only), ULT, UVT, CULT (output only), CUVT (output only), GTx, CGTx (output only) where “x” is the letter identifier of the constituent

Paths:

All . . . . . FR, DUPI, DUPJ, GK, HK, EI, EJ, DK, AF, AFTH, MCH (integer), HCH, FRC, PLTH (output only), TLTH (output only), XLTH (output only), RADI, RADJ, VAI, VAJ, VXI, VXJ, ROTR, MREG (integer), EFFP, QTMK, TORQ, CXI, CYI, CZI, CXJ, CYJ, CZJ

Tube . . . . . AM

Tube/STUBE . HC, FC, FPOW, AC, IPDC, UPF, TLEN, DH, AFI, AFJ, FK, WRF, FG, FD, DEFF, CURV, FCLM, FCTM, REY (output only), REW (output only), REX (output only), RVR, MREG (output only), XMA (output only), ASPU, ASPD, GUx, GDx, CASPU, CASPD, CGUx, CGDx, IUAS, IDAS, FUAS, FDAS, UVPU, ULPU, UVPD, ULPD, CUVPU, CULPU, CUVPD, and CULPD where “x” is the letter identifier of the constituent

Paths (connector devices):

MFRSET . . . . . SMFR
VFRSET . . . . . SVFR
Loss/valve . . . . FK, TPF
LOSS2 . . . . . FK, TPF, FKB
(U/D)PRVLV . . FK, FKH, FKL, PSET, RLAG, TPF
ORIFICE . . . . . FK, AORI, CDIS, ELLD, MODO (integer), MODA (integer), AORI_C, AORI_T, AORI_L, AORI_H
TABULAR. . . . OFAC, AFI, AFJ, UPF
CAPIL . . . . . RC, CFC, XVH, XVL (same for 1st CAPPMP connector)
PUMP . . . . . SPD, RSPD, HPMP, GPMP, DGDH, HCF, GCF, TCF, ANPSH, RNPSH, ANSS, RNSS, SSCF, CCH, CCE, VCOR, HVF, GVF, EFFVF, LIMP (integer), AFI, AFJ, UPF
TURBINE . . . . SPD, HTURB, GTURB, DGDH, GCF, TCF, DTURB, EFFM, EFFA, LIMP (integer), AFI, AFJ, UPF, UCR
COMPRESS . . SPD, HCOMP, GCOMP, DGDH, GCF, TCF, EFFM, EFFA, LIMP (integer), AFI, AFJ, UPF, CNS, SSCF
COMPPD. . . . . SPD, HTURB, GTURB, DGDH, GCF, TCF, EFFM, EFFA, EFFV, EFFVA, EFFVM, DISP, LIMP (integer), AFI, AFJ, UPF

Ties:

All . . . . . UA, QTIE, DUPL, DUPN, XOL, XOH, FQ, UAM, AHT, UB, UEDT, TEF, DTEF, MODW (integer), UVEL, IPUV (integer), ITAC (integer)
HTN* . . . . . CBD, UER, UEH, UEC, XNB, XNUL, XNLM, XNLA, XNTM, XNTA, UEV, UED, UEP, UEK, UECP, UET, CHFF, HFFL,RLAM,RTURB, MODR (integer), IHTC (integer, reserved), ICHF (integer)
HTP . . . . . DCH, DEPTH, ACCM, SPR, CSF, XNB, CHFF, HFFL, MODR (integer), XNUL, XNLM, XNLA, HCON, THKL

Fties:

All . . . . . GF (meaningless for CONSTQ), QF, DUPC, DUPD

Interfaces (ifaces):

All . . . . . VA, VB, PA, PB, FPV, DUPA, DUPB
All but NULL . VHI, VLO, EMA
OFFSET. . . . . DPAB
SPRING. . . . . DPDV, VZRO
WICK. . . . . XIL, XIH, RCAP, VZRO, VSUB, RBP
SPHERE . . . . . XIL, XIH

FLUINT Utility Routines:

CHGLMP . . . . Change lump PL, TL, XL, XG, or XF; see also CHALLP, CHGLMP_MIX
CHGVOL . . . . Change tank VOL
CHGLSTAT . . Change LSTAT for a lump
CHGSUC . . . . Change path suction status
CHGFLD . . . . Change working fluid
HLDLMP . . . . Make a tank or junction act like a plenum
HTRLMP . . . . Holds the enthalpy of a junction or STEADY tank
RELLMP . . . . Reverse action of HLDLMP and HTRLMP
NOSLIP . . . . . Force one or all twinned paths to stay homogeneous
GOSLIP . . . . . Reverse action of NOSLIP: enable slip flow in twins
NOTWIN . . . . Force one or all twinned tanks to stay homogeneous
GOTWIN . . . . Reverse action of NOTWIN: enables nonequilibrium
SPLIT_TWIN. Doesn't just enable nonequibrium, it immediately splits the tanks
INTLMP . . . . . Fetch internal lump sequence number
INTTIE . . . . . Fetch internal tie sequence number
INTPAT. . . . . Fetch internal path sequence number
INTFLD . . . . . Fetch internal fluid pointer
INTSPE. . . . . Fetch internal species pointer
MFLTRN . . . . Return fluid submodel sequence number
PUTTIE . . . . . Move a tie to a new node and/or lump
SPRIME . . . . . Keep a capillary device primed
XTRACT . . . . Calculate effective quality extracted by a path; see also XTRACC
XTRACC . . . . Return effective path species suction fraction
SUMFLO . . . . Summed or average properties in a fluid submodel
SUMDFLO. . . Same as SUMFLO but takes into account path duplication factors
REPATH . . . . . Returns the estimated Reynolds number for a path
CHKFLASH. . Verify that no lump, throat state, or pump will flash or cavitate
TOTATTP . . . . Calculates total (stagnation) temperature and pressure

6.18.5 Reserved Words

This subsection summarizes the reserved word list. See also Section 4.1.3.9. Reserved words are SINDA/FLUINT variables and arrays that are contained in data modules and a few named common blocks. These common blocks and USE statements are inserted automatically into the routines resulting from logic blocks such as OPERATIONS, PROCEDURE OUTPUT CALLS, VARIABLES 0/1/2, FLOGIC 0/1/2, SOLOGIC 0/1/2 and SOLOUTPUT CALLS. They are only inserted into routines in SUBROUTINES if the CALL COMMON command is used. *It is the user's responsibility to avoid collisions with the reserved words.* In this context, a "collision" is the inadvertent use of a reserved word as a user variable or array name.

The reserved word list consists of all translatable network parameters and control constants, *but not all reserved words are translatable.* Table 6-1 (continued in Table 6-2) lists all reserved words in alphabetical order. In runs with no fluid submodels, the reserved word list is reduced. However, it is considered good practice to avoid FLUINT reserved names in case fluid submodels are added later.

Finally, note that most Fortran words such as "CONTINUE" or "OPEN" cannot be used.

Table 6-1 Reserved Word List

(A-Z)TEST	BFJ	CZJ	DTIMUF	FI	IF
(A-Z)TEST_DP	C	DABS	DTMAXF	FI(A-Z)	IFIX
A	CABS	DACFR	DTMINF	FK	IFLOAT
ABS	CASE	DACOS	DTMPCA	FKI	IHTC
ABSZRO	CASL	DASIN	DTMPCC	FKJ	IMPLICIT
AC	CASPD	DATA	DTSIZF	FLOOR	INDEX
ACCELX	CASPU	DATAN	DTTUBF	FORALL	INQUIRE
ACCELY	CAST	DATAN2	DTURB	FORMAT	INT
ACCELZ	CCE	DATE	DUPA	FPOW	INTEGER
ACCM	CCH	DBLE	DUPB	FPV	INTENT
ACHGO	CCOS	DCH	DUPC	FQ	INTERFACE
ACOS	CDB	DCOS	DUPD	FR	INTRINSIC
ADERO	CDIS	DCOSH	DUPI	FRAVER	IPDC
ADERR	CEXP	DEALLOCATE	DUPJ	FRC	IPUV
AERRO	CGD	DEFAULT	DUPL	FRD	IREG
AERRR	CGD(A-Z)	DEFF	DUPN	FRD(A-Z)	ISIGN
AF	CGL	DELOBJ	EBALNA	FRH	ITEROT
AFI	CGL(A-Z)	DELTAT	EBALNC	FRH(A-Z)	ITERXT
AFJ	CGT	DEPTH	EBALSA	FUAS	ITHLDF
AFTH	CGT(A-Z)	DEXP	EBALSC	FUNCTION	ITHOLD
AHT	CGU	DF	EFFA	G	ITROTF
AIMAG	CGU(A-Z)	DFLOAT	EFFM	GCF	JNINT
AINT	CHAR	DG	EFFP	GCOMP	JREG
AK	CHARACTER	DGDH	EFFV	GD	K
AL	CHFF	DH	EFFVA	GD(A-Z)	LASTC
ALLOCATABLE	CLOG	DHI	EFFVF	GF	LEN
ALLOCATE	CLOSE	DHJ	EFFVM	GFR	LGE
ALLSAME	CMPLX	DIM	EI	GK	LGT
ALOG	CNS	DIMENSION	EJ	GL	LIMP
AM	COMMON	DINT	ELEMENTAL	GL(A-Z)	LINECT
AMAX0	COMP	DISP	ELLD	GO	LLE
AMAX1	CONJG	DK	ELSE	GOAL	LLT
AMGERR	CONTAINS	DL	ELSEIF	GOTO	LOG
AMGERRF	CONTINUE	DLOG	ELSEWHERE	GPMP	LOG10
AMIN0	COS	DLOG10	EMA	GT	LOGICAL
AMIN1	COSH	DMAX1	END	GT(A-Z)	LOOPCO
AMOD	CSF	DMIN	ENDCASE	GTURB	LOOPCR
ANINT	CSGFAC	DMIN1	ENDDO	GU	LOOPCT
ANPSH	CSGMAX	DMOD	ENDFILE	GU(A-Z)	MATEVR
ANSS	CSGMIN	DNINT	ENDFORALL	GVF	MATMET
AORI	CSIN	DO	ENDIF	HC	MATMETF
AORI_C	CSQRT	DOH	ENDWHERE	HCAC	MAX
AORI_H	CULDP	DOUBLE	ENDWHILE	HCF	MAX0
AORI_L	CULI	DPAB	ENLT	HCH	MAX1
AORI_T	CULPU	DPDV	ENTRY	HCOMP	MCH
ARLXCA	CULT	DPROD	EQUIVALENCE	HCON	MDERO
ARLXCC	CURV	DRLXCA	ESUMIS	HEN	MDERR
ASIN	CUVI	DRLXCC	ESUMOS	HFFL	METAMG
ASL	CUVPD	DSIGN	EXP	HK	METAMGF
ASPD	CUVPU	DSIN	EXTERNAL	HL	METHO
ASPU	CUVT	DSINH	EXTLIM	HPMP	MF
ASSIGN	CX	DSQRT	FBEBALA	HR	MF(A-Z)
AST	CXI	DTAN	FBEBALA	HTURB	MIN
ATAN	CXJ	DTANH	FBEBALC	HVF	MIN0
ATAN2	CY	DTEF	FC	IABS	MIN1
ATMPCA	CYCLE	DTIMEH	FCLM	ICHAR	MLINE
ATMPCC	CYI	DTIMEI	FCTM	IDGC	MMODS
BACKSPACE	CYJ	DTIMEL	FD	IDIM	MOD
BACKUP	CZ	DTIMES	FDAS	IDINT	MODA
BFI	CZI	DTIMEU	FG	IDNINT	MODEC

Table 6-2 Reserved Word List, Cont'd

MODO	NLOOPR	OPEITR	RCHGO	T	UVI
MODR	NLOOPS	OPEN	RCVIO	TAN	UVPD
MODULE	NLOOPT	OPITRF	RDERO	TANH	UVPU
MODW	NLTOT	OPTIONAL	RDERR	TARGET	UVT
MREG	NMACT	OUTPTF	READ	TCF	VA
MXCNTR	NMARI	OUTPUT	REAL	TEF	VAI
MXCOND	NMBD	OVEREL	REBALF	THEN	VAJ
MXFILE	NMDIF	PA	RECURSIVE	THKL	VB
MXFLID	NMHT	PAGECT	RERRF	TIMDY	VBUB
MXFRES	NMOD	PARAMETER	RERRO	TIMEM	VCOR
MXMODF	NNOD	PATMOS	RERRR	TIMEN	VDOT
MXMODS	NOUT	PB	RESULT	TIMEND	VDRP
MXNCON	NRAD	PG	RETURN	TIMEO	VHI
MXNODE	NRTOT	PH	REW	TL	VLO
MXTARY	NSEED	PL	REWIND	TLEN	VOL
MXTOKN	NSOL	PLTH	REX	TLTH	VSUB
MXUARY	NSOLOR	POINTER	REY	TMXA	VXI
MXUCON	NSTATO	PPG	RLAM	TMXB	VXJ
NA	NSTRT	PPG(A-Z)	RMFRAC	TMXC	VZRO
NAMELIST	NTC	PRECISION	RMRATE	TMXD	WHERE
NARLXC	NTJ	PRINT	RMSPLT	TO	WHILE
NARLXN	NTK	PRIVATE	RNPSH	TOLD	WRF
NATMPC	NTL	PSET	RNSS	TORQ	WRITE
NATMPN	NTOTK	PT	ROTR	TYPE	XF
NCGMAC	NTOTL	PUBLIC	RSMAXF	UA	XF(A-Z)
NCGMAN	NTPL	PURE	RSSIZF	UAM	XG
NCONVO	NTT	PUSHO	RSTUBF	UB	XG(A-Z)
NCSGMC	NTTB	Q	RTURB	UCA	XIH
NCSGMN	NTWOP	QCHEM	RVR	UCR	XIL
NCTOT	NULLIFY	QDOT	SAVE	UEC	XK
NDINT	NUMCNT	QF	SELECT	UECP	XL
NDNAM	NUMJNC	QHTRK	SELECTCASE	UED	XLTH
NDRLXC	NUMPLE	QHTRL	SEQUENCE	UEDT	XMA
NDRLXN	NUMTNK	QL	SIGMA	UEH	XMDOT
NDTMPC	NUMTUB	QTIE	SIGN	UEK	XMDOT(A-Z)
NDTMPN	NUSER1	QTM	SIN	UEP	XNB
NEBALC	NUSER2	QTMK	SINH	UER	XNLA
NEBALN	NVARBL1	RADI	SNGL	UET	XNLM
NERVUS	NWWWW1	RADJ	SPACE	UEV	XNTA
NEWPRO	NWWWW2	RAD_A	SPARSEG	UID	XNTM
NFLO	NZZZZ1	RAD_L	SPR	ULDP	XNUL
NFM	NZZZZ2	RBP	SQRT	ULI	XOL
NGSTRT	NZZZZ3	RCACTO	SSCF	ULPU	ZJ
NGTOTAL	OBJECT	RCAP	STOP	UPF	ZR
NLIN	OFAC	RCERRO	SUBROUTINE	UVEL	ZR(A-Z)
NLOOPO	ONLY				

6.19 Model Size Limits

SINDA/FLUINT is dimensioned to handle a *total* of 1,000,000 nodes and 25,000 lumps. The default size limits are listed below in Table 6-3.

Table 6-3 Model Size Limits

THERMAL:	Submodels	999
	Nodes	1,000,000
	Conductors	unlimited
FLUID:	Submodels	999
	Lumps	25,000
	Paths	50,000
	Paths per nonplenum lump	1,000
	Ties	500,000
	Fties	25,000
	Interfaces	25,000
	Macros	10,000
	Constituents per submodel	26
	User fluids	1,000
	Total length of FPROP arrays	100,000
BOTH:	User arrays	unlimited
	Values per array	unlimited
	CARRAYs	unlimited
	Named constants	unlimited
	Numbered constants	unlimited
	Registers	15,000
	Design Variables	1,000
	Optimization Constraints	1,000
	Random Variables	1,000
	Reliability Constraints	1,000

Please contact CRTech (see page xlvii of the preface for contact information) if any of these limits are exceeded.

7 SUBROUTINE LIBRARY

This section describes the subroutines that may be called from within the logic blocks. Subroutines are available to perform network solutions (described earlier in Section 4.2), network mapping and output, restart and parametric operations, component simulations, mathematical operations, etc. Because the operations performed by the program are determined by which routines are called and in which order, it is well worth the reader's time to be familiar with all of the support routines that are available.

7.1 Introduction

Introductory remarks and general usage recommendations are included in this section with the intent of giving the novice user some insight into the scope of the available routines, as well as identifying those routines and associated techniques which enjoy frequent usage in real engineering applications. Consider interpolation subroutine D1D1DA, whose calling sequence is as follows:

```
CALL D1D1DA (X,AX(IC),AY(IC),Y)
```

where:

X . . . . . Value of independent variable
AX. . . . . Singlet array reference of the form smn.An or An
AY. . . . . Singlet array reference to array of Y-values corresponding to each X-value in AX (form smn.An or An)
Y . . . . . Dependent variable result

It will be noted, first, that there are no record column designations. The word "CALL" begins to the right of column 6, in keeping with FORTRAN convention. Next, it will be seen that the descriptions of the arguments are rather terse and make no explicit mention of arithmetic type or permissible reference forms. Arithmetic type is always implied by the first letter of each formal dummy argument, in accordance with implicit Fortran conventions. Letters I through N, inclusive, imply integer arguments; all other letters imply floating point arguments. If an argument must be a character string, this fact will be stated explicitly. For example, D1D1DA requires two simple arguments, X and Y, which must be floating point.

7.1.1 Basic Conventions

The following paragraphs describe the conventions that will be used in this section.

Data Types—Unless otherwise specified, formal arguments whose first letter is I, J, K, L, M, or N represent actual arguments which must be *integers*. Formal arguments beginning with any other letter represent actual arguments which must be *floating point* or *real* values, unless otherwise specified. When an actual argument must be a *character* value, this will be identified.

The Fortran compiler will check arguments in subroutines and functions against the expected number and type (see Section 4.1.4.2). Some routines provide optional arguments, but otherwise failure to meet the expected data type will result in a compiler error.

For example, this means NA and K cannot be substituted for A and XK if real argument is expected, even though the arrays are equivalenced (point to the same memory). It also means “0” and “0.0” cannot be used arbitrarily as the argument in a routine: the right one (integer or real) must be used according to what the routine expects.

Reference Forms—Arguments which serve as input data to a subroutine may be supplied as literal data values or as variables. In the former case, the value must be of the type (i.e., integer, floating point, etc.) shown in the calling sequence. In the latter case, the variable name must refer to a memory location which contains a value of the proper type. Arguments which will receive the results (i.e., output) of a subroutine must be supplied as Fortran variables.

Formal arguments that are suffixed with (IC) (e.g., smn.AX(IC), smn.A(IC), Y(IC), etc., smn refers to a submodel name) indicate that the actual argument supplied by the user *must be an array reference of the integer count form*. Integer count form is the convention for storage of array data in SINDA/FLUINT, where the first cell contains the size of the array stored as an integer. Formal arguments that are suffixed with (DV) (e.g., smn.G(DV), Q(DV), smn.T(DV), etc.) indicate that the actual argument supplied by the user must refer to the location of the first data value (i.e., the starting location) in an array. Hence, the user may supply an array reference of the *data value* (A(n+i) or smn.A(n+i)) form, or in general, any reference which refers to the starting location of the desired sequence of data values.

Actual To Relative Conversions—Some subroutine arguments rely upon the preprocessor’s ability to convert from actual constant, array, node, conductor, lump, tie, and path numbers to relative numbers. To use this capability the user may supply an actual constant number, array number, node number, conductor number, lump number, tie number, or path number by preceding the actual number with K, A, T, G, or FLUINT variable respectively. This causes the preprocessor to replace the entry with the relative number. Consider for example the argument list shown below:

```
(2, A14, POD2.T5, G7)
```

In this example, following the preprocessor phase, A14 would be replaced by the pointer to the first cell in the A array, corresponding to the start of array number 14 (the user should remember that this first cell is equivalenced to the integer size of the array), T5 would be replaced by the relative node number for actual node number 5, and G7 would be replaced by the relative conductor number for actual conductor number 7.

Recall that all subroutine calls with arguments that require conversion must be made using translatable statements since F-type statements by-pass the conversion process. Translatable statements begin with a blank, an M or an integer number in Column 1. F-type statements must begin with an F, or be bracketed by FSTART/FSTOP instructions. The majority of the library subroutines may be called with a translatable statement, whether or not conversions must be performed. In the few instances where an F is mandatory, it will be included in the calling sequence. Otherwise, the examples are assumed to be translatable statements unless otherwise indicated.

Arguments which represent arrays are indicated by suffixing the formal argument with a parenthesized “IC” or “DV.” IC indicates that the argument supplied by the user must be an array reference of the integer-count form. This terminology results from the fact the first cell of all SINDA/FLUINT arrays is equivalenced to an integer containing the array size, IC, followed by IC data values. For example, D1D1DA requires two arrays, AX and AY, which must contain floating point numbers and which must be supplied as references of the form: smn.An, where n is the array number and smn is the associated submodel name. DV indicates that the argument supplied by the user may be an array reference of the data-value form or any type of reference which is associated with a single data value or the starting location of a sequence of data values. A floating point argument is expected—the use of the above form (smn.An) would cause an error even if array contained floating point values because the first cell is equivalenced to an integer value, and the subroutine is expecting the first cell to be a floating point value.

When several similar subroutines are described, the formal description of similar arguments will be deleted. For example, consider subroutine D11DAI. This routine performs single variable linear interpolation on an array of Xs to produce an array of Ys. The number of input Xs must be supplied separately as the argument N and must agree with the number of output Ys. The calling sequence is as follows:

```
CALL D11DAI (N,X(DV),AX(IC),AY(IC),Y(DV))
```

It should be clear, from the basic action of the routine and the description of the arguments for D1D1DA, that the required arguments for D11DAI are as follows:

N . . . . . Number of values in X and Y arrays (integer)
X . . . . . Array of input values of the independent variable
AX. . . . . Singlet array of Xs in ascending order (integer count form)
AY. . . . . Singlet array of Ys corresponding to the Xs in AX (IC form)
Y . . . . . Array of output values of the dependent variable

D11DAI is equivalent to N calls to D1D1DA. If these conventions are not obvious, the user should not become alarmed. Rather, he or she should recognize that ease and confidence in understanding and using the more complex routines in the library come only after experience is gained in using the simpler ones.

7.1.2 Interpolation Subroutines

Section 7.3 describes a large variety of interpolation routines. Analytically minded users might, at first, shun the concept of approximating smooth curves with a series of straight lines. However, since SINDA is built around discretized variables, interpolation is the method of choice for evaluating time or temperature dependent properties and single or multi-variable functions. Though parabolic interpolation is also available, basic linear interpolation serves the great majority of user’s needs:

$$Y = Y_1 + \frac{(Y_2 - Y_1)}{(X_2 - X_1)} (X - X_1)$$

Linear interpolation is available in two elementary forms: subroutine D1DEG1, which requires one doublet array of (Xi,Yi) ordered pairs, and subroutine D1D1DA, which requires two matching singlet arrays, one containing values of Xi and the other containing corresponding values of Yi. If the user must approximate several dependent variables over roughly the same domain of a single independent variable, then the use of subroutine D1D1DA, along with its associated array structure, will be preferred.

7.1.3 Input and Output Subroutines

Section 7.4 describes the routines that users might use to construct their OUTPUT CALLS blocks. This set of routines provide printed output as well as output to disk files. The disk file outputs are for use in parametric runs, restart runs and for external programs such as EZXY®, Sinaps®, and C&R Thermal Desktop®, which provide graphical visualization of time dependent data. Section 4.7 describes the routines used for reading data from disk files for use by the program. The restart and parametric output routines each have a counterpart input routine to retrieve the data. Other input routines are available for reading in time dependent boundary temperatures and heat source data that are written to disk files by external programs. Fluid submodel output routines are described in Section 7.11.8.

An overview of output options is available in Section 4.6.

7.1.4 Application and Utility Subroutines

Section 7.6 and Section 7.7 describe the utility subroutines and application subroutines. Utility subroutines allow the user to conveniently locate such things as submodel names and other model attributes in the form of Fortran-addressable memory locations. Application subroutines are devoted to specific applications such as phase change and heat exchanger modeling.

FLUINT utility and application routines are described in Section 7.11.1.

7.2 Subroutine Name Index

Name	Paragraph	Page	Description
FLOMAP	7.11.8	7-382	Prints Fluid Submodel Map
FORCER	7.12.1	7-476	Forces Complete Propagation of Registers to Model
FORWRD	4.2.1	4-51	Transient network solution
FRICT	7.11.5	7-360	Darcy Friction Factor
FRIEDL	7.11.5	7-360	Two Phase Friction Gradient
FRPRNT	7.11.8	7-382	Prints Flow Rates For Paths
FTTAB	7.11.8	7-382	Tabulates Basic Ftie Parameters
FUSION	7.7.7	7-119	Phase change (PCM) simulation
FWDBCK	4.2.2	4-55	Transient network solution (old name for TRANSIENT)
GADDi	7.7.6	7-117	Calculate Effective Network Conductance
GASTAB	7.11.8	7-382	Tabulates lump mass transfer coefficients
GASTB2	7.11.8	7-382	Tabulates lump homogeneous nucleation data
GASATB	7.11.8	7-382	Tabulates path interface area data
GASGTB	7.11.8	7-382	Tabulates path mass transfer coefficients
GENOUT	7.4.9	7-45	Prints Out Arrays
GOHOMO	7.11.1.3	7-235	See NOSLIP
GOSLIP	7.11.1.3	7-235	Undoes Actions of NOSLIP; Enables Slip Flow
GOTWIN	7.11.1.3	7-235	Undoes Actions of NOTWIN; Enables Nonequilibrium
GPRINT	7.4.1	7-33	Prints Out Conductances
GVTEMP	7.3.1	7-16	Update Temp.-Dependent Conductances
GVTIME	7.3.2	7-16	Update Time-Dependent Conductances
HEATER	7.7.3	7-106	Thermostatic Heater Switching
HEATPIPE	7.7.9	7-130	Variable and Constant Conductance Heat Pipes
HEATPIPE2	7.7.9	7-130	Variable and Constant Conductance 2D Heat Pipes
HENRY	7.11.2.3	7-277	Henry’s constant at a given T,P
HLDCST	7.13.2	7-500	Fixes a Constraint Variable to be an Equality Constraint
HLDDES	7.13.1	7-497	Fixes a Design Variable as a Temporary Constant
HLDLMP	7.11.1.4	7-241	Makes Tank or Junction Act like Plenum
HLDTANKS	7.11.1.4	7-241	Holds all Tank Thermodynamic States
HNQCAL	7.7.1	7-103	Heater Node Energy Requirements
HNQPNT	7.4.13	7-49	Prints Qs For Heater Nodes
HPGLOC	7.7.9	7-130	Heat pipe utility: returns gas front location
HPUNITS	7.7.9	7-130	Heat pipe utility: set/change units
HRPRINT	7.4.1	7-33	Conductor heat rate printing routine
HTCDIF	7.11.3	7-287	Corrects condensation to account for diffusion
HTCENH	7.11.4.7	7-325	Heat transfer enhancement by fins, ribs, protrusions
HTCENTR	7.11.4.6	7-322	Heat transfer enhancement for entrances
HTCMIX	7.11.3	7-287	Convective Heat Transfer Coef. Multiple-constituent
HTCRSIO	7.11.4.9	7-354	Convection between rotating and stationary disk w/ flow
HTCRSFF	7.11.4.9	7-354	Convection between rotating and stationary disk, no flow
HTCURVE	7.11.4.5	7-320	Heat transfer enhancement for curved tubes or coils
HTCWRF	7.11.4.4	7-318	Heat transfer enhancement for rough walls
HTFDIF	7.11.3	7-287	Solves for liq./vap. interface temps and flows
HTLTAB	7.11.8	7-382	Tabulates lump internal heat transfer data
HTPTAB	7.11.8	7-382	Tabulates path internal heat transfer data
HTRLMP	7.11.1.4	7-241	Hold Enthalpy of Lump
HTRMOD	7.6.20	7-90	Hold Temperature of Thermal Submodel
HTRNOD	7.6.19	7-89	Hold Temperature of Node
HTUDIF	7.11.3	7-287	Corrects condensation for diffusion (HTU ties)
HUMD2X	7.11.2.4	7-278	Convert Dew Point to Mass Fraction
HUMR2X	7.11.2.4	7-278	Convert Relative Humidity to Mass Fraction
HUMX2R	7.11.2.4	7-278	Convert Mass Fraction to Relative Humidity

Name	Paragraph	Page	Description
HXCNT	7.7.5	7-112	Counter Flow Heat Exch. Effectiveness
HXCOND	7.7.5	7-112	Condensing Heat Exchanger
HXCROS	7.7.5	7-112	Cross Flow Heat Exchanger Effectiveness
HXEFF	7.7.5	7-112	Heat Exchanger Simulations
HXMASTER	7.11.10.3	7-461	System-level simulation of heat exchangers
HXPAR	7.7.5	7-112	Parallel Flow Heat Exch. Effectiveness
IFCTAB	7.11.8	7-382	Tabulates iface parameters
INTCON	7.6.2	7-74	Internal Pointer For a Given Conductor
INTFLD	7.11.1.6	7-248	Internal Pointer For a Given Working Fluid
INTFTI	7.11.1.6	7-248	Internal Pointer For a Given Ftie
INTIFC	7.11.1.6	7-248	Internal Pointer For a Given Iface
INTLMP	7.11.1.6	7-248	Internal Pointer For a Given Lump
INTNOD	7.6.5	7-76	Internal Pointer For a Given Node
INTPAT	7.11.1.6	7-248	Internal Pointer For a Given Path
INTPTS	7.11.1.6	7-248	Internal Pointer For a Given Subpath
INTSPE	7.11.1.6	7-248	Internal Pointer For a Given Fluid Species
INTTIE	7.11.1.6	7-248	Internal Pointer For a Given Tie
ISREG1	7.12.4	7-488	Check if Given Value is a Parameter
JETARN	7.11.4.3	7-312	Heat transfer for arrays of nozzles (jets)
JETASN	7.11.4.3	7-312	Heat transfer for rows of slots (jets)
JETSRN	7.11.4.3	7-312	Heat transfer for a single round nozzle (jet)
JETSSN	7.11.4.3	7-312	Heat transfer for a single slot (jet)
JF_OSF	7.11.11.4	7-472	Colburn J factor for Offset Strip Fin heat exchangers
LAGRAN	7.3.6	7-29	Lagrangian Int. Using Doublet Arrays
LAYINIT	7.7.12	7-170	Layer initialization utility for ACCRETE, RECESS
LGRNDA	7.3.6	7-29	Lagrangian Int. Using Singlet Arrays
LOCMAR	7.11.5	7-360	Two Phase Friction Gradient
LOSSTAB	7.11.8	7-382	Output for K-factor devices
LMPMAP	7.11.8	7-382	FLOMAP for One Lump
LMPRNT	7.11.8	7-382	Prints Lump Thermodynamic State Variables
LMCTAB	7.11.8	7-382	Tabulates Lump Parameters, First 10 Constituents
LMPTAB	7.11.8	7-382	Tabulates Basic Lump Parameters
LMXTAB	7.11.8	7-382	Tabulates Extra Lump Parameters
LSTSQU	7.6.9	7-78	Least Square Curve Fit - Polynomial
LUMPRATES	7.11.1.10	7-262	Returns lump dM/dt and dU/dt
L2TAB	7.11.8	7-382	Tabulates Additional Twinned Tank Parameters
MACONE	7.11.1.9	7-260	Reinitializes a LINE or HX macro
MACINTAB	7.11.8	7-382	Duct macro input echo
MACROTAB	7.11.8	7-382	Duct macro output tabulation
MAPHEN	7.11.8	7-382	Tabulates Henry's constant
MCADAM	7.11.5	7-360	Two Phase Friction Factor Gradient
MDLTRN	7.6.4	7-75	Relative Location of Thermal Submodel
MFLTRN	7.11.1.6	7-248	Relative Location of Fluid Submodel
MIN_FIND	7.10.1	7-209	Co-solving minimum finder
MODTRN	7.6.4	7-75	Relative Location of Thermal Submodel
MPYARY	7.8.5	7-187	Multiplies Two Arrays

Name	Paragraph	Page	Description
NCCONCYL	7.11.4.8	7-328	Nat'l convection: Between concentric cylinders
NCCONSPH	7.11.4.8	7-328	Nat'l convection: Between concentric spheres
NCCYLIN	7.11.4.9	7-354	Mixed convection filling/emptying cylinders
NCFINS	7.11.4.8	7-328	Nat'l convection: Rectangular fins, isothermal (air)
NCGBUB	7.11.9.2	7-409	Simulates Stationary Noncondensible Gas Bubble
NCHCT	7.11.4.8	7-328	Nat'l convection: Horizontal cylinder, isothermal
NCHSD	7.11.4.8	7-328	Nat'l conv.: Horiz. surf., hot downside or cool upside
NCHSU	7.11.4.8	7-328	Nat'l conv.: Horiz. surf., hot topside or cool downside
NCPPT	7.11.4.8	7-328	Nat'l convection: Parallel plates, isothermal (heated)
NCPPF	7.11.4.8	7-328	Nat'l convection: Parallel plates, isoflux (heated)
NCPROP	7.11.4.8	7-328	Nat'l convection: Rayleigh, Prandtl number, etc.
NCRENC	7.11.4.8	7-328	Nat'l convection: Thin enclosure (panel cavity)
NCSET	7.11.4.8	7-328	Control constant reset for Nat'l convection
NCSPHERE	7.11.4.8	7-328	Nat'l convection: Gas inside a spherical vessel
NCVCF	7.11.4.8	7-328	Nat'l convection: Vertical cylinder, isoflux
NCVCT	7.11.4.8	7-328	Nat'l convection: Vertical cylinder, isothermal
NCVFPF	7.11.4.8	7-328	Nat'l convection: Near-vertical) flat plates, isoflux
NCVFPT	7.11.4.8	7-328	Nat'l convection: Near-vertical flat plates, isothermal
NODBAL	7.6.13	7-80	Net nodal heat input
NODMAP	7.4.2	7-34	QMAP for One Node
NODTAB	7.4.1	7-33	Node data tabulation
NODTRN	7.6.5	7-76	Relative Location of T, C, and Q For Node
NORMPROB	7.14.3	7-515	Normal (Gaussian) probability given value
NORMVAL	7.14.3	7-515	Normal (Gaussian) value given probability
NOSLIP	7.11.1.3	7-235	Forces Twinned Paths to Stay Homogeneous
NOTWIN	7.11.1.3	7-235	Forces Twinned Tanks to Stay Homogeneous
NTU2EFF	7.11.10.2	7-457	Calculates effectiveness given NTU/UA_{tot}
NUMTRN	7.6.6	7-76	Relative Location on UC of a Constant
OFFCST	7.13.2	7-500	Temporarily disables an optimization constraint
ORIFTAB	7.11.8	7-382	Output tabulation for controlled ORIFICE devices
PAGHED	7.4.16	7-51	Change page set-up (titles, line counts)
PCOOLCON	7.7.4	7-109	Cooler (refrigerator) controller, proportional
PHEATER	7.7.3	7-106	Proportional Heater Simulation
PID	7.7.8	7-121	PID controller utility
PIDGET	7.7.8	7-121	PID controller performance metrics
PIDINIT	7.7.8	7-121	PID controller initialization
PIDS	7.7.8	7-121	PID controller for steady states
PIDSET	7.7.8	7-121	PID controller (re)initialization
PIDSETW	7.7.8	7-121	PID controller: set limits on control variable, wind-up
PIDTAB	7.7.8	7-121	PID controller output tabulation
PLAWICK	7.11.9.4	7-416	Planar Wet Wick Conductance USER Ftie Utility
POOLBOIL	7.11.4.1	7-301	Pool boiling heat transfer correlation
PORTTAB	7.11.8	7-382	Tabulates path port end locations and depth
PR8000	7.11.2.5	7-282	Generate simplified FPROP blocks (perfect gas)
PR9000	7.11.2.5	7-282	Generate simplified FPROP blocks (simple liquid)
PREPDAT1	7.13.3	7-504	Prepares Test Data for COMPARE
PREPDAT2	7.13.3	7-504	Prepares Test Data for COMPARE
PREPLIST	7.13.3	7-504	Prepares Predictions for COMPARE
PREPMC	7.11.2.2	7-275	Set Concentrations for multiple constituent calculation
PRINT	7.4.14	7-50	Controls Printout Routine
PRINTA	7.4.9	7-45	Prints An Array Of Values
PRINTTL	7.6.23	7-101	Prints a list of the reasons for the smallest time steps
PRPMAP	7.11.8	7-382	Maps Working Fluid Properties
PSWEEP	7.12.7	7-494	Parametric sweep of a single register

Name	Paragraph	Page	Description
PTHTAB	7.11.8	7-382	Tabulates Path Parameters
PTCTAB	7.11.8	7-382	Tabulates Path Species Suction Action
PUMPMAP	7.11.8	7-382	PUMP device output
PUTPAT	7.11.1.8	7-257	Moves a Path to a New Lump
PUTTIE	7.11.1.8	7-257	Moves a Tie to a New Lump and/or Node
QDPRNT	7.11.8	7-382	Prints Values of QDOT For Lumps
QFLOW	7.6.17	7-85	Finding heat flows between nodes or groups of nodes
QFLOWSET	7.6.17	7-85	Utility for QFLOW
QFORCE	7.8.13	7-191	Multiplies A(1) by A(2)-A(3) Forms A(4)
QMAP	7.4.2	7-34	Prints Thermal Network Map
QMETER	7.8.14	7-191	Multiplies C1 by C2-C3 Forms C4
QMTRI	7.8.15	7-192	Multiplies A(1) by A(2)-A(3) Forms A(4)
QPRINT	7.4.1	7-33	Prints Nodal Heat Sources (Q)
QVTEMP	7.3.1	7-16	Update Temp.-Dependent Sources
QVTIME	7.3.2	7-16	Update Time-Dependent Sources
RANTAB	7.14.1	7-510	Tabulates Random Variables and Rel. Engr. Status
RCGET	7.14.3	7-515	Fetches RCSTTAB data for named Rel. Constraints
RCGETNO	7.14.3	7-515	Fetches RCSTTAB data for unnamed Rel. Constraints
RCSTTAB	7.14.1	7-510	Tabulates Rel. Constraints and Rel. Engr. Status
RECESS	7.7.11.2	7-160	Surface recession simulation (heat input)
RECESS_RATE	7.7.11.2	7-160	Surface recession simulation (rate of recession input)
RECESS_SET	7.7.11.2	7-160	Set surface recession simulation controls
REC_QRATE	7.7.12	7-170	Heat rate utility for RECESS (same as ACCQRATE)
RELCST	7.13.2	7-500	Releases Action of HLDCST
RELDES	7.13.1	7-497	Releases Action of HLDDES
RELEST	5.21	5-95	Gradient-based Reliability Estimation
RELLMP	7.11.1.4	7-241	Releases HLDLMP and HTRLMP Actions
RELMOD	7.6.20	7-90	Releases HTRMOD Actions
RELNOD	7.6.19	7-89	Releases HTRNOD Actions
RELTANKS	7.11.1.4	7-241	Releases all Tank Thermodynamic States
REGTAB	7.4.15	7-50	Tabulates registers and expressions
REPATH	7.11.1.10	7-262	Returns path Reynolds number
REREG	7.12.3	7-486	Reconnects Register Changes to Model
REREG1	7.12.3	7-486	Reconnects Changes to One Parameter
REREGC	7.12.3	7-486	Custom Reconnect of Register Changes
RESAMP	7.14.3	7-515	Resets sampling from being cumulative
RESAVE	7.4.3	7-36	Stores Data For Restart
RESETM	7.14.3	7-515	Resets random variables to mean values
RESPAR	7.5.3	7-70	Re-Initializes For Parametric Cases
RESTAR	7.5.4	7-71	Reads In Parameters From Restart File
RESTDB	7.14.2	7-512	Restores from REDB file
RESTDP	7.6.15	7-82	Restarts Suspended Double Precision Options
RESTNC	7.5.4	7-71	Same as RESTAR without Collision Checks
RDTNQS	7.8.16	7-192	$[(T1)^4 - (T2)^4] * Constant$
ROHSEN	7.11.3	7-287	Condensation Heat Transfer Coefficient
ROOT_FIND	7.10.2	7-215	Co-solving root finder
ROTTAB	7.11.8	7-382	Tabulation of path rotation options
RVSWEEP	5.26	5-109	Parametric sweep of a single random variable

Name	Paragraph	Page	Description
SAMPLE	5.19	5-88	Monte Carlo Reliability Estimation
SAVE	7.4.5	7-40	Writes Variables To SAVE folder
SAVEDB	7.14.2	7-512	Writes Variables to REDB folder
SAVPAR	7.4.6	7-41	Saves Data For Parametric Cases
SETTL	7.6.23	7-101	Sets the time step list size for later PRINTTL call
SETTMOD	7.5.1	7-68	Set temperatures at the thermal submodel level
SHAH	7.11.3	7-287	Shah's Condensation Correlation
SINDA_CLOCK	7.6.22	7-100	Returns wall clock time since start of run
SINDA_CPU	7.6.22	7-100	Returns CPU time since start of run
SIZETAB	7.4.18	7-54	Model size information (no. of nodes etc.)
SOLCHECK	7.13.4	7-509	Solver diagnostic utility
SOLVER	5.2	5-10	Optimization, Goal-seeking, Data Correlation
SORTPR	7.4.11	7-47	Prints Sorted Nodal Temperatures
SPLIT_TWIN	7.11.1.3	7-235	Causes twinned tanks to immediately split
SPRIME	7.11.1.7	7-255	Forces a Capillary Device to Prime
STDATMOS	7.7.13	7-183	Standard Atmospheric Temperature and Pressure
STDHTC	7.11.3	7-287	Convective Heat Transfer Coefficients
STDSTL	4.2.3	4-59	Steady state network solution
STEADY	4.2.4	4-64	Steady state network solution
STNDRD	7.4.10	7-46	Prints Line of Data - Time, CSGMIN, etc.
STOPDP	7.6.15	7-82	Temporarily Suspends Double Precision Option
STP1AM	7.3.7	7-30	Step Interpolation On Doublet Array
STP1AS	7.3.7	7-30	Step Interpolation On Doublet Array
STP2AM	7.3.7	7-30	Step Interpolation On Singlet Arrays
STP2AS	7.3.7	7-30	Step Interpolation On Singlet Arrays
STPRNT	7.11.8	7-382	Prints STUBE Parameters
SUBARY	7.8.7	7-188	Subtracts Two Arrays
SUBMAP	7.4.21	7-65	Submodel-level output
SUMARY	7.8.9	7-189	Sums Elements of an Array
SUMDFLO	7.11.1.10	7-262	Same as SUMFLO, Uses Path Duplication Factors
SUMFLO	7.11.1.10	7-262	Sums or Averages Values in a Fluid Submodel
SVPART	7.4.6	7-41	Saves Partial Data for Parametric Cases
TankCalc60	7.11.9.10	7-449	Import and modeling utility for tank modeling
TANKTRHO	7.11.1.1	7-221	Change tank temperature at constant density/mass
TANK_LINKER	7.11.9.3	7-411	IFACE-like utility for linking tanks between submodels
TBPRNT	7.11.8	7-382	Prints Tube Parameters
TB2TAB	7.11.8	7-382	Tabulates Tube and STUBE Flow Areas and AC Factors
TDUMP	7.4.1	7-33	Prints Nodal Temperatures
TECINFO	7.7.10	7-146	TEC/Peltier device sizing utility
TECUNITS	7.7.10	7-146	Units specification for TEC/Peltier utilities
TEC1	7.7.10	7-146	TEC/Peltier device simulation, single stage, 1D
TEC2	7.7.10	7-146	TEC/Peltier device simulation, single stage, 2D
THRMST	7.7.2	7-104	Thermostat Switch
TIEHTPTAB	7.11.8	7-382	Output tabulation for HTP ties
TIETAB	7.11.8	7-382	Tabulates Tie Parameters
TIE2P	7.11.8	7-382	Returns the "2P" column data from TIETAB in logic
TI2TAB	7.11.8	7-382	Tabulates Additional Tie Parameters
TKPRNT	7.11.8	7-382	Prints Tank Parameters
TMNMX	7.4.8	7-44	Prints Max./Min. Nodal Temperatures

Name	Paragraph	Page	Description
TORCYL1	7.11.7.2	7-376	Torque for 1 path, cylindrical (axial) gap
TORCYLN	7.11.7.2	7-376	Torque for N paths, cylindrical (axial) gap
TORCYLM	7.11.7.2	7-376	Torque for duct macro path, cylindrical gap
TORCYLU	7.11.7.2	7-376	Torque in cylindrical gap, user dimensions and speed
TORDISK1	7.11.7.1	7-367	Torque for 1 path, radial gap
TORDISKN	7.11.7.1	7-367	Torque for N paths, radial gap
TORDISKM	7.11.7.1	7-367	Torque for duct macro path, radial gap
TORDISKU	7.11.7.1	7-367	Torque in radial gap, user dimensions and speed
TOTALTP	7.11.1.10	7-262	Returns Total (stagnation) Temperature, Pressure
STATICTPA	7.11.1.10	7-262	Returns Static absolute ref. frame
TPRINT	7.4.1	7-33	Prints Nodal Temperatures
TRANSIENT	4.2.2	4-55	Transient network solution
TRPZDA	7.6.7	7-77	Integrates By Trapezoidal Rule
TSAVE	7.4.7	7-43	Saves Temperatures For Post Processing
TSINK	7.4.20	7-57	Calculates and outputs multiple sink temperatures
TSINK1	7.4.20	7-57	Calculates a single sink temperature
TTTAB	7.11.8	7-382	Tabulates Twinned Tank Parameters
TUBTAB	7.11.8	7-382	Tabulates Tube and STUBE Parameters
TURBMAP	7.11.8	7-382	TURBINE device output
TWNTAB	7.11.8	7-382	Tabulates Twinned Tube and STUBE Parameters
UDFERR	7.11.8	7-382	Error Routine for 6NNN Fluids
UPREG	7.12.1	7-476	Propagates Register Changes to Model
UPREG1	7.12.1	7-476	Propagates Changes to One Parameter
UPREGC	7.12.1	7-476	Custom Propagation of Register Changes
UPREGF	7.12.1	7-476	Propagates Register Changes to Fluid Submodels
UPREGT	7.12.1	7-476	Propagates Register Changes to Thermal Submodels
USECSR	7.4.3	7-36	Use CSR folders instead of traditional SAVE Files
USESAVEFILE	7.4.3	7-36	Use traditional SAVE files instead of CSR folders
USRFIL	7.4.17	7-52	Opens an Extra User File
USRFIL2	7.4.17	7-52	Like USRFIL, but doesn't fail if file not found
VALPHA through VVISCV	7.11.2	7-272	Fluid Property Routines
VELPATH	7.11.1.10	7-262	Returns path flow velocity and rotation components
WAVLIM	7.11.9.6	7-423	Maximum Time Step to Resolve Pressure Waves
WCHOKE	7.11.9.5	7-420	Monitors one or more paths for choking limit
WETWICK	7.11.9.4	7-416	Wet Wick Conductivity Estimation
XTRACC	7.11.1.10	7-262	Effective Species Suction Fraction
XTRACT	7.11.1.10	7-262	Effective Path Suction Quality
YSMPWK	7.11.8	7-382	YSMP Workspace Utilization Reporting (FLUINT)
YSMPWS	7.6.18	7-88	YSMP Workspace Utilization Reporting (SINDA)
YTHUSH	7.11.9.7	7-428	Wye-Tee error and abort suppressing
YTKALG	7.11.9.7	7-428	Wye-Tee (diverge/merge) K-Factor Corr. (Gardel)
YTKALI	7.11.9.7	7-428	Wye-Tee (diverge/merge) K-Factor Corr. (Idelchik)
YTKONV	7.11.9.7	7-428	Wye-Tee K-factor Conversion Utility
YTWARN	7.11.9.7	7-428	Wye-Tee abort suppressing

7.3 Interpolation - Extrapolation Subroutines

Temperature-Dependent Variable Interpolation (7.3.1):

CVTEMP . . . . . Linear interpolation of capacitance vs. temperature according to NODE DATA block inputs.
GVTEMP. . . . . Linear interpolation of conductor values vs. temperature according to CONDUCTOR DATA block inputs.
QVTEMP. . . . . Linear interpolation of Q-sources vs. temperature according to SOURCE DATA block inputs.

Time-Dependent Variable Interpolation (7.3.2):

GVTIME . . . . . Linear interpolation of conductor values vs. time according to CONDUCTOR DATA block inputs.
QVTIME . . . . . Linear interpolation of Q-sources vs. time according to SOURCE DATA block inputs.

Generalized Linear Interpolation (7.3.3):

D1DEG1 . . . . . Linear interpolation using doublet array.
D1D1DA . . . . . Linear interpolation using two singlet arrays.
D1D1DAH. . . . Linear interpolation with hysteresis
D11MDA . . . . . Same as D1D1DA except allows single value multiplier.
D3DEG1 . . . . . Linear interpolation using trivariate arrays.
D3D1WM . . . . Trivariate linear interpolation with single value.
D1D1WM . . . . Uses D1DEG1 and multiplies the interpolation by the Z value.
D2D1WM . . . . Bivariate liner interpolation with single value multiplier.
D1MDG1. . . . . Same as D1DEG1 except arithmetic mean of two input values used as independent variable.
D1M1DA. . . . . Same as D1MDG1 except uses two singlet arrays.
D1M1WM . . . . Same as D1MDG1 except a single value multiplier is allowed.
D1M1MD . . . . Same as D1M1DA except a single value multiplier is allowed.
D1DG1I . . . . . Perform interpolation on an array of Xs to obtain an array of Ys.
D1D1IM. . . . . Same as D1DG1I with a single value multiplier.
D1D1MI. . . . . Same as D1D1IM except an array of multipliers is allowed.
D11DAI . . . . . Same as D1DG1I except uses two singlet arrays.
D11DIM. . . . . Same as D1D1IM except uses two singlet arrays.
D11MDI. . . . . Same as D1D1MI except uses two singlet arrays.
D2DEG1 . . . . . Linear interpolation using bivariate arrays.
D2MXD1. . . . . Same as D2DEG1 except uses arithmetic mean of two values as independent variable.
D2MX1M . . . . Same as D2MXD1 except a single value multiplier is allowed.
D1IMD1. . . . . Double array linear interpolation using arithmetic mean of two values as independent variable.

D1IMWM . . . . Same as D1IMD1 with a single value multiplier.
D1IMIM . . . . . Same as D1IMD1 except allows an array of multipliers.

Cyclical Interpolation (7.3.4):

DA11MC . . . . Same as DA11CY except allows single value multiplier.
CSIFLX . . . . . Same as DA11MC except there are two dependent variables for each independent variable
D11MCY . . . . Same as D11CYL except allows single value multiplier.
D11CYL . . . . . Cyclical linear interpolation using a single cycle doublet array.
DA11CY. . . . . Same as D11CYL except two singlet arrays are used.
DA11DP . . . . . Same as D11CY except integer array, integer result is used.
D12CYL . . . . . Same as D11CYL except a parabolic interpolation is performed.
DA12CY. . . . . Same as DA11CY except a parabolic interpolation is performed.
D12MCY . . . . Same as D11MCY except parabolic interpolation is performed.
DA12MC . . . . Same as DA11MC except parabolic interpolation is performed.

Generalized Parabolic Interpolation (7.3.5):

D1DEG2. . . . . Parabolic interpolation using a doublet array.
D1D2DA. . . . . Parabolic interpolation using two singlet arrays.
D1D2WM. . . . Same as D2DEG2 except a single value multiplier is allowed.
D12MDA . . . . Same as D1D2WM except uses two singlet arrays.
D1MDG2 . . . . Same as D1DEG2 except the arithmetic average of two values is used as independent variable.
D1M2DA . . . . Same as D1D2DA except the arithmetic average of two values is used as independent variable.
D1M2WM . . . Same as D1MDG2 except a single value multiplier is used.
D1M2MD. . . . Same as D1M2WM except uses two singlet arrays.
D2DEG2. . . . . Parabolic interpolation using a bivariate array.
D2D2WM. . . . Same as D2DEG2 except a single value multiplier is used.
D2MX2M. . . . Same as D2DEG2 except the arithmetic mean of two input values is used as independent variable.
D2MXD2 . . . . Same as D2D2WM except the arithmetic mean of two input values is used as independent variable.

Generalized Lagrangian Interpolation (7.3.6):

LAGRAN . . . . Performs Lagrangian interpolation using a doublet array.
LGRNDA . . . . Performs Lagrangian interpolation using two singlet arrays.

Step Interpolation (7.3.7):

STP1AS . . . . . Performs step interpolation using doublet arrays.
STP2AS . . . . . Performs step interpolation using two singlet arrays.
STP1AM. . . . . Performs step interpolation on doublet arrays using a singlet array if Xs.
STP2AM. . . . . Same as STP1AM except uses singlet X and singlet Y arrays.

7.3.1 Temperature-Dependent Variable Interpolation

Subroutine Names: CVTEMP, GVTEMP, QVTEMP

Description: These subroutines perform the capacitance, conductance and Q-source vs. temperature interpolations required to support the temperature-dependent options allowed in the NODE, CONDUCTOR and SOURCE DATA blocks. They also account for FAC records and user constant multipliers used in the temperature-dependent CAPACITANCE, SOURCE and CONDUCTORS DATA inputs. They all interpolate on arrays input in the ARRAY DATA blocks. Linear interpolation with Z value multiplication on the result is performed. In multi-submodel problems a subroutine call is required to update each submodel's parameters.

Restrictions and Guidance: Calls to these routines at the end of each VARIABLES 1 block are generated by the preprocessor. The user may insert VARIABLES 1 statements after the calls by inserting his own calls to CVTEMP, GVTEMP and/or QVTEMP. User calls to these routines may be appropriate in OPERATIONS for initialization. Calls from VARIABLES 0, VARIABLES 2 and OUTPUT CALLS are inappropriate. User calls to these routines *must not* have an "F" in column 1.

Calling Sequences:

```
M     CALL CVTEMP ('SMN')
M     CALL GVTEMP ('SMN')
M     CALL QVTEMP ('SMN')
```

where SMN is a thermal submodel name.

7.3.2 Time-Dependent Variable Interpolation

Subroutine Names: GVTIME, QVTIME

Description: These subroutines perform the conductor and Q-source interpolations required to support the time-dependent options allowed in the CONDUCTOR and SOURCE DATA blocks. They also account for FAC records and user constant multipliers that are used in the time-dependent SOURCE and CONDUCTOR DATA inputs. They interpolate on dual singlet arrays input in the ARRAY DATA block. Linear interpolation with Z-value multiplication on the result is performed. In multi-submodel problems, a subroutine call is required for updating each submodel's parameters.

Restrictions and Guidance: Calls to these routines at the end of the VARIABLES 0 blocks are generated by the preprocessor. This means that the CONDUCTOR DATA and SOURCE DATA blocks override any user-input VARIABLES 0 code as a default condition. To override this, the user may insert a call QVTIME (SMN) into VARIABLES 0, and follow this with any logic required to override source data. The same procedure also works for GVTIME.

User calls to these routines may be appropriate in OPERATIONS for initialization. Calls from VARIABLES 1, VARIABLES 2 and OUTPUT CALLS are inappropriate. User calls to these routines *must not* have an “F” in column 1.

Calling Sequences:

```
M     CALL GVTIME ('SMN')
M     CALL QVTIME ('SMN')
```

where SMN is a submodel name.

7.3.3 Generalized Linear Interpolation

Subroutine Names: D1DEG1, D1D1DA, D11MDA

Description: Subroutines D1DEG1 and D1D1DA perform a linear interpolation using a doublet array and two singlet arrays respectively. Subroutine D11MDA uses D1D1DA and multiplies the result by a Z-value.

Restrictions and Guidance: These subroutines may be called from any logic block. All independent variable array values must be entered in monotonically increasing order. If the independent variable value supplied is outside the array’s range, the appropriate end-value will be used. Z-value multipliers and X-values may be literal floating point numbers, user constants, program variables or elements from a user’s array.

Calling Sequences:

```
M       CALL D1DEG1 (X,A(IC),Y)
M       CALL D1D1DA (X,AX(IC),AY(IC),Y)
M       CALL D11MDA (X,AX(IC),AY(IC),Z,Y)
```

where:

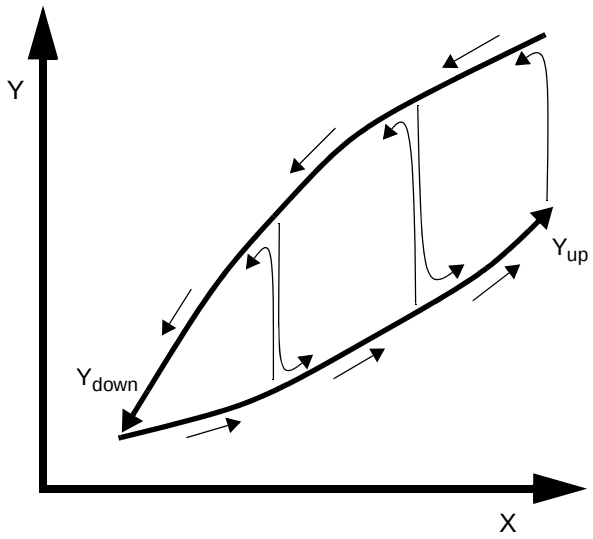
X . . . . . Value of independent variable (floating point)
A . . . . . Doublet array of X-Y pairs (floating point)
AX. . . . . Singlet array reference of the form smn.An or An (floating point values)
AY. . . . . Singlet array reference to array of Y values corresponding to each X value in AX (form smn.An or An, floating point values)
Z . . . . . Value result is multiplied by before being returned as Y (floating point)
Y . . . . . Dependent variable result (floating point)

Examples:

```
CALL D1DEG1(3.0, A101, YTEST)
CALL D1D1DA(TIMEN,A1,A2,XK3)
```

Subroutine Name: D1D1DAH

Description: Subroutine D1D1DAH performs a linear interpolation using singlet arrays, as does D1D1DA (see above). Unlike D1D1DA, however, D1D1DAH employs hysteresis. D1D1DAH interpolates one of two dependent variable (Y) arrays, with the choice of arrays made depending on whether or not the independent variable X is increasing or decreasing, as depicted in the figure on the right.



If X is increasing (since the last call for this variable), then the curve Y_{up} is employed. If instead X is decreasing, Y_{down} is used. If there is no change in X, the prior Y curve is used such that redundant calls are tolerated.

As with D1D1DA, the end points of the current Y curve are used if X exceeds the range of the supplied X array; the routine interpolates but never extrapolates. However, there is no requirement that the Y curves must have the same values at these end points. (In the figure above at the right, the Y values are the same at the minimum value of X but differ at the maximum value of X.)

For the routine to know whether or not X is increasing or staying the same (and if so it must remember which Y curve was last used), it must keep an internal memory of past calls. To enable this tracking of trends, the user must identify each D1D1DAH call with a unique integer ID assigned to each resulting Y variable.*

When the first call to D1D1DAH is made is made for any particular ID, no trend history is yet available. In this case, the code assumes that X will increase in the future, and it will the Y_{up} array.† An option exists to erase the memory (signaled by a negative sign on the ID) and return an average

* The same X variable can be used for multiple D1D1DAH calls that employ the same or different X and Y curves, but the user must then assign a unique ID for each such set on the basis of unique usage of the resulting Y value. For example, if multiple valves share the same set of performance curves, each valve should be assigned a separate ID for its D1D1DAH call.

† To initialize instead with Y_{down} values, a second call with a slightly lower value of X can be made, perhaps in OPERATIONS.

of the Y_{up} and Y_{down} values, which might be a useful choice for some steady-state analyses, and will be necessary in order to reset the history between parametric transient analyses (including those appearing in PROCEDURE or RELPROCEDURE).

Restrictions and Guidance: This subroutines may be called from any logic block. The values in the independent variable (X) array must be entered in monotonically increasing order. If the independent variable (X) value supplied is outside the array's range, the appropriate end-values of Y will be returned: no extrapolations are made. Trends in X are measured compared to the last call with the same ID (made from any location). Zero change in X (perhaps from duplicate or redundant calls) does not affect the internally stored trend.

Calling Sequence:

```
CALL D1D1DAH (ID, X, AX(IC), AYU(IC), AYD(IC), Y)
```

where:

- `ID` ..... Unique integer ID for each end application of the dependent variable (Y). A negative sign causes the history of this ID to be erased, and returns the average of the two dependent variable curves (AYU and AYD) as Y.
- `X` ..... Value of independent variable (floating point).
- `AX` ..... Singlet array reference of the form smn.An or An. Values in this array must increase monotonically.
- `AYU` ..... Singlet array reference to array of Y_{up} real values corresponding to each X value in AX (form smn.An or An), to be used while X is increasing. If there is no history available for this ID, X is assumed to be increasing.
- `AYD` ..... Singlet array reference to array of Y_{down} real values corresponding to each X value in AX (form smn.An or An), to be used while X is decreasing. AYD may be the same argument as AYU.
- `Y` ..... Dependent variable result (floating point)

Examples:

```
      CALL D1D1DAH( 1023, Xtest, A10, A201, A202, Ytest)
c
      if(nsol .le. 1)then    $ use average and/or clear memory in STEADY
            CALL D1D1DAH( -991, Pressure, A991, A9910, A9911, CondIns)
      else
            CALL D1D1DAH(  991, Pressure, A991, A9910, A9911, CondIns)
      endif
```

Subroutine Names: D3DEG1, D3D1WM

Description: These subroutines perform trivariate linear interpolation. The interpolation array must be constructed as shown for trivariate array format. (See Section 2.11.2.) Subroutine D2DEG1 is called on which calls on D1DEG1. Hence, the linear extrapolation feature of these routines applies. Subroutine D3D1WM multiplies the interpolated answer by F prior to returning it as T.

Restrictions: See trivariate array format. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D3DEG1(X,Y,Z,TA(IC),T)
M     CALL D3D1WM(X,Y,Z,TA(IC),F,T)
```

Example:

```
      CALL D3DEG1(3.0,2.,1.0, A11, FRED)
```

Subroutine Names: D1D1WM

Description: This subroutine performs single variable linear interpolation by calling on D1-DEG1. However, the interpolated answer is multiplied by the values addressed as Z prior to being returned as Y.

Restrictions: Same as D1DEG1. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D1D1WM(X,A(IC),Z,Y)
```

Subroutine Name: D2D1WM

Description: This subroutine performs bivariate linear interpolation by calling on D2DEG1. The interpolated answer is multiplied by the W value prior to being returned as Z.

Restrictions: Same as D2DEG1. All variables and array values must be real (floating point).

Calling Sequence:

```
M      CALL D2D1WM (X, Y, BA(IC), W, Z)
```

Example:

```
CALL D2D1WM(TIMEN, T101, A33, WTEST, ZTEST)
```

Subroutine Names: D1MDG1, D1M1DA

Description: These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. They require a doublet or two singlet arrays respectively.

Guidance: See D1DEG1 or D1D1DA. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D1MDG1 (X1,X2,A(IC),Y)
M     CALL D1M1DA (X1,X2,AX(IC),AY(IC),Y)
```

Subroutine Names: D1M1WM, D1M1MD

Description: These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

Restrictions: Same as D1MDG1 or D1M1DA. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D1M1WM(X1,X2,A(IC),Z,Y)
M     CALL D1M1MD(X1,X2,AX(IC),AY(IC),Z,Y)
```

Subroutine Names: D1DG1I, D1D1IM, D1D1MI

Description: These subroutines perform single variable linear interpolation on an array of Xs to obtain an array of Ys. D1D1IM multiplies all interpolated values by a constant Z value while D1D1MI allows a unique Z value for each X value. They all call on D1DEG1.

Restrictions: The number of input Xs must be supplied as the integer N and agree with the number of Y and Z locations where applicable. All variables except N, and all array values must be real (floating point).

Calling Sequences:

```
M    CALL  D1DG1I (N,X(DV),A(IC),Y(DV))
M    CALL  D1D1IM (N,X(DV),A(IC),Z,Y(DV))
M    CALL  D1D1MI (N,X(DV),A(IC),Z(DV),Y(DV))
```

Examples:

```
CALL D1DG1I(NTEST, A(101+1), A(102), A(103+1))
CALL D1D1IM(33, FRED(1), A123, ZTEST, WILMA(1))
```

Subroutine Names: D11DAI, D11DIM, D11MDI

Description: These subroutines are virtually identical to D1DG1I, D1D1IM and D1D1MI respectively. The difference is that they require singlet arrays for interpolation and call on D1D1DA.

Guidance: Same as D1DG1I, D1D1IM and D1D1MI. All variables (except N) and all array values must be real (floating point).

Calling Sequence:

```
M    CALL D11DAI (N,X(DV),AX(IC),AY(IC),Y(DV))
M    CALL D11DIM (N,X(DV),AX(IC),AY(IC),Z,Y(DV))
M    CALL D11MDI (N,X(DV),AX(IC),AY(IC),Z(DV),Y(DV))
```

Subroutine Name: D2DEG1

Description: This subroutine performs two linear interpolations using a bivariate array.

Restrictions and Guidance: The bivariate array must be entered in an ARRAY DATA block using the bivariate array format, (Section 2.11.2). If an independent variable value outside the dependent variable range in the array is supplied, the appropriate end-point value is returned. Independent-variable values supplied may be literal floating point numbers, program variables, user constants or elements in user arrays.

All variables and array values must be real (floating point).

Calling Sequence:

```
M    CALL D2DEG1 (X,Y,BA(IC),Z)
```

where:

X . . . . . Independent variable value
Y . . . . . Independent variable value
BA(IC) . . . . Bivariate array reference of the form smn.An or An
Z . . . . . Dependent variable value returned

Examples:

```
CALL D2DEG1( TIMEN, MOD.T11, OMOD.A123, RESULT)
CALL D2DEG1( 10.0, 20.0, A231, ZTEST)
```

Subroutine Name: D2MXD1

Description: This subroutine is identical to D2DEG1 except that it uses the arithmetic mean of two X values as the independent variable.

Restrictions: Same as D2DEG1. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D2MXD1(X1,X2,Y,BA(IC),Z)
```

Subroutine Name: D2MX1M

Description: This subroutine is identical to D2MXD1 except that the Z value is multiplied by W before being returned.

Restrictions: Same as D2MXD1. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D2MX1M(X1,X2,Y,BA(IC),W,Z)
```

Subroutine Name: D1IMD1, D1IMWM, D1IMIM

Description: These are indexed subroutines which use the arithmetic mean of two input values as the independent variable for linear interpolation. The array of answers (Y) produced are left as is (D1IMD1), are all multiplied by a single factor (D1IMWM), or each answer is multiplied by a separate factor.

Restrictions: The interpolation array addressed must have an even number of input values and the independent variables must be in ascending order. These routines call up D1D1WM. N is the number of times the operation is to be performed. All variables (except N) and all array values must be real (floating point).

Calling Sequences:

```
M     CALL D1IMD1(N,X1(DV),X2(DV),A(IC),Y(DV))
M     CALL D1IMWM(N,X1(DV),X2(DV),A(IC),Z,Y(DV))
M     CALL D1MIM(N,X1(DV),X2(DV),A(IC),Z(DV),Y(DV))
```

Example:

```
      CALL D1MIM(NTEST, A(101+1), A(102+1), A(103), A(104+1), A(105+1))
```

7.3.4 Interpolation Using Cyclical Arrays

Subroutine Name: DA11MC, CSIFLX

Description: These subroutines perform linear interpolations using cyclical arrays and multiply the result by a Z-value. CSIFLX is like DA11MC except that for each independent variable there are two dependent variables. For both routines. scale factors can be input for each of the dependent variables.

Restrictions and Guidance: Both routines may be called from any logic block. DA11MC requires two singlet arrays in the ARRAY DATA block. CSIFLX requires a singlet array for the independent variable and a doublet array for the dependent variable. The independent variable array range must be equal to or greater than the value of the cyclic period supplied, and values must be input in ascending order. Any X-values in the array that are greater than the period will be ignored. Independent variable and Z-values supplied may be literal floating point numbers, program variables, user constants, or elements of user arrays.

All variables and array values must be real (floating point).

Restriction: The starting value of the independent variable *must* be zero. This means that if time is used as the independent variable, the initial time must be zero and the final time must be greater than or equal to the period.

Calling Sequence:

```
M     CALL DA11MC (PER,X,AX(IC),AY(IC),Z,Y)
```

where:

PER ..... period of cyclic function
X ..... Independent variable value supplied
AX(IC) .... Independent variable array reference of the form smn.An or An. The first value must be zero, and the last value must be greater than or equal to PER.
AY(IC) .... Dependent variable array reference of the form smn.An or An
Z ..... Value that result is multiplied by before being returned as Y
Y ..... Dependent variable result

Calling Sequence:

```
M     CALL CSIFLX(PER,X,AX(IC),AY(IC,IB),Z,YS,YI,Y)
```

where:

PER ..... period of cyclic function
X ..... Independent variable value supplied
AX(IC) .... Independent variable array reference of the form smn.An or An. The first value must be zero, and the last value must be greater than or equal to PER.
AY(IC,IB) Dependent variable array reference of the form smn.An or An structured as follows: AY(1,1),AY(1,2),AY(2,1),AY(2,2) ...
Z ..... Scaling factor for the sum of the scaled dependent variables
YS ..... Scale factor for the first dependent variable AY(IC,1)
YI ..... Scale actor for the second dependent variable AY(IC,2)
Y ..... Dependent variable result

Subroutine Name: D11MCY

Description: This subroutine is virtually identical to D11CYL except that the interpolation is multiplied by the floating point Z value prior to being returned as Y.

Restrictions: Calls subroutine D1DEG1. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D11MCY(PR,X,A(IC),Z,Y)
```

Subroutine Names: D11CYL, DA11CY

Description: These subroutines reduce memory requirements for cyclical interpolation arrays. The arrays need cover one period only, and the period (PR) must be specified as the first argument. Linear interpolation is performed, and the independent variables must be in ascending order.

Restrictions: All values must be floating point, except for NAY and NY, which must be integers (NAY is an array containing integer values). Subroutines INTRFC is called on by both D11CYL and DA11CY, then D1DEG1 or D1D1DA respectively.

All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D11CYL(PR,X,A(IC),Y)
M     CALL DA11CY(PR,X,AX(IC),AY(IC),Y)
M     CALL DA11SP(PR,X,AX(IC),NAY(IC),NY)
```

Subroutine Names: D12CYL, DA12CY

Description: These subroutines are virtually identical to D11CYL and DA11CY except that parabolic interpolation is performed.

Restriction: Discontinuities are not tolerated in the dependent variables. This means the value at the starting point must equal the value at PR.All variables and array values must be real (floating point).

Guidance: See D11CYL and DA11CY. Subroutines LAGRAN and LGRNDA respectively are called.

Calling Sequences:

```
M     CALL D12CYL(PR,X,A(IC),Y)
M     CALL DA12CY(PR,X,AX(IC),AY(IC),Y)
```

Subroutine Names: D12MCY, DA12MC

Description: These subroutines are virtually identical to D11MCY and DA11MC except that parabolic interpolation is performed.

Restriction: Discontinuities are not tolerated in the dependent variables. This means the value at the starting point must equal the value at PR. All variables and array values must be real (floating point).

Guidance: Calls on subroutines LAGRAN and LGRNDA respectively.

Calling Sequences:

```
M     CALL D12MCY(PR,X,A(IC),Z,Y)
M     CALL DA12MC(PR,X,AX(IC),AY(IC),Z,Y)
```

7.3.5 Generalized Parabolic Interpolation

Subroutine Names: D1DEG2, D1D2DA

Description: These subroutines perform single variable parabolic interpolation. The first requires a double array of X,Y pairs while the second requires singlet arrays of X and Y values. They call on subroutines LAGRAN and LGRNDA respectively.

Restrictions: See LAGRAN or LGRNDA respectively. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D1DEG2(X,A(IC),Y)
M     CALL D1D2DA(X,AX(IC),AY(IC),Y)
```

Examples:

```
      CALL D1DEG2(TIMEN, A101, YTEST)
      CALL D1D2DA(T100, A102, FRED.A102, XK3)
```

Subroutine Names: D1D2WM, D12MDA

Description: These subroutines perform single variable parabolic interpolation by calling on LAGRAN or LGRNDA respectively. However, the interpolated answer is multiplied by the valued addressed as Z prior to being returned as Y.

Restrictions: Same as LAGRAN or LGRNDA and Z must be a floating point number.

Calling Sequences:

```
M     CALL D1D2WM(X,A(IC),Z,Y)
M     CALL D12MDA(X,AX(IC),AY(IC),Z,Y)
```

Subroutine Names: D1MDG2, D1M2DA

Description: These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. They require a double or two singlet arrays respectively.

Restrictions: See LAGRAN or LGRNDA as they are called on respectively. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D1MDG2(X1,X2,A(IC),Y)
M     CALL D1M2DA(X1,X2,AX(IC),AY(IC),Y)
```

Subroutine Names: D1M2WM, D1M2MD

Description: These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

Restrictions: Same as D1MDG2 or D1M2DA. All variables and array values must be real (floating point).

Calling Sequences:

```
M     CALL D1M2WM(X1,X2,A(IC),Z,Y)
M     CALL D1M2MD(X1,X2,AX(IC),AY(IC),Z,Y)
```

Subroutine Name: D2DEG2

Description: This subroutine performs parabolic interpolation on bivariate arrays. The arrays must be formatted as shown in Section 2.11.2, Bivariate array format.

Restrictions: N.GT.3, M.GT.3. See Bivariate array format.

Calling Sequence:

```
M     CALL D2DEG2(X,Y,BA(IC),Z)
```

Subroutine Name: D2D2WM

Description: This subroutine performs parabolic interpolation on a bivariate array by calling on D2DEG2. The result of the interpolation is multiplied by the W value prior to being returned as Z.

Restrictions: Same as D2DEG2. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D2D2WM(X,Y,BA(IC),W,Z)
```

Subroutine Name: D2MX2M

Description: This subroutine is virtually identical to D2D2WM except it uses the arithmetic mean of two X values as the independent variable for interpolation.

Restrictions: Same as D2D2WM. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D2MX2M(X1,X2,Y,BA(IC),W,Z)
```

Subroutine Name: D2MXD2

Description: This subroutine is virtually identical to D2DEG2 except that the arithmetic mean of two X values is used as the independent variable for interpolation.

Restrictions: Same as D2DEG2. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL D2MXD2(X1,X2,Y,BA(IC),Z)
```

7.3.6 Generalized Lagrangian Interpolation

Subroutine Names: LAGRAN, LGRNDA

Description: These subroutines perform interpolation of up to order 50. The first requires one doublet array of X, Y pairs while the second requires two singlet arrays, one of Xs and the other of Ys. They contain an extrapolation feature such that if the X value falls outside the range of the independent variable the nearest dependent Y variable value is returned and no error is noted.

Restrictions: All values must be floating point except N which is the order of interpolation plus one and must be an integer. The independent variable values must be in ascending order.

Calling Sequences:

```
M     CALL LAGRAN(X,Y,A(IC),N)
M     CALL LGRNDA(X,Y,AX(IC),AY(IC),N)
```

Note: A doublet array is formed as follows:

```
X1,Y2,X2,Y2,X3,Y3.....,XN,YN
```

and singlet arrays are formed as follows:

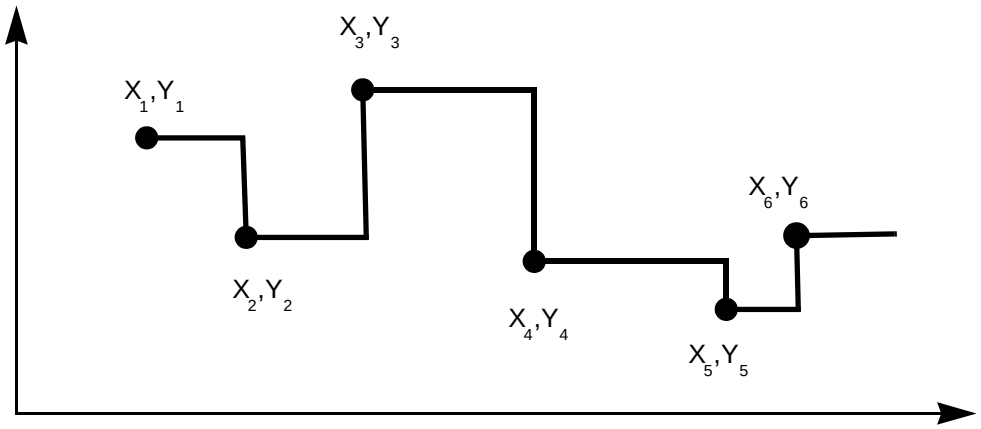
```
X1,X2,X3,....,XN
Y1,Y2,Y3,....,YN
```

7.3.7 Step Interpolation

Subroutine Name: STP1AS

Description: This subroutine performs step interpolation on a doublet array of X,Y pairs. Step interpolation minimizes the input data and interpolation calculation time.

Consider the following example:



Only the points shown in the above example need be input. For independent variable values less than X_1, Y_1 is returned as the answer. For independent variable values greater than X_6, Y_6 is returned as the answer. Otherwise for X_i .LE. X .LT. X_{i+1} the value Y_i is returned as the answer.

Restrictions: The X independent variable values in the doublet array A must be in ascending order. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL STP1AS (X,A(IC),Y)
```

where:

X . . . . . Input value of independent variable
A . . . . . Doublet array of X,Y pairs
Y . . . . . Output value of dependent variables

Example:

```
CALL STP1AS(TIMEN, A301, ZTEST)
```

Subroutine Name: STP2AS

Description: This subroutine performs step interpolation on a pair of singlet arrays containing corresponding values of X and Y.

Restrictions: The X independent variable values in the AX array must be in ascending order. The number of values in the AX and AY arrays must be the same. All variables and array values must be real (floating point).

Calling Sequence:

```
M     CALL STP2AS(X,AX(IC),AY(IC),Y)
```

where:

X . . . . . Input value of the independent variable
AX. . . . . Singlet array of X values
AY. . . . . Singlet array of Y values corresponding to the X values in AX
Y . . . . . Output value of the dependent variable

Subroutine Name: STP1AM

Description: This subroutine performs step interpolation on a doublet array A of X,Y pairs using a singlet array of Xs (independent variable input values) to form a singlet array of Ys. The X independent variable values in the double array A must be in ascending order.

Restrictions: The number of input Xs must be supplied as the integer N and agree with the number of Y locations. All variables (except N) and all array values must be real (floating point).

Calling Sequence:

```
M     CALL STP1AM(N,X(DV),A(IC),Y(DV))
```

Subroutine Name: STP2AM

Description: This subroutine is virtually identical to STP1AM except the interpolation is performed on a pair of singlet arrays of AX and AY.

Restrictions: Same as STP1AM. All variables (except N) and all array values must be real (floating point).

Calling Sequence:

```
M     CALL STP2AM(N,X(DV),AX(IC),AY(IC),Y(DV)
```

7.4 Output Subroutines

7.4.1 Printout of Thermal Network Attributes

Subroutine Names: CPRINT, QPRINT, TPRINT, NODTAB, GPRINT, HRPRINT, TDUMP

Description: These subroutines write values of thermal capacitance, conductance, heat rates, and temperatures to the system print file (the plain text or ASCII output file named in OPTIONS DATA). All temperatures and non-zero conductance, capacitance and heat rates values are printed for a specified submodel. TDUMP is just like TPRINT except that it does not print out the balance criteria header or page between submodels. Instead, it just prints the submodel name and the time for each header.

NODTAB provides extensive data on each node, including its energy imbalance (dE/dt).

Restrictions and Guidance: All routines are callable from OUTPUT CALLS, but not restricted to this. A submodel name must be specified and should agree with the submodel name on the preceding HEADER OUTPUT CALLS record.

Guidance: Q (heat rate) values might be adjusted from their SOURCE DATA values, perhaps including the actions of FLUINT ties, which are essentially treated as time-varying sources from a thermal submodel perspective.

Calling Sequences:

```
CALL CPRINT ('SMN')
CALL QPRINT ('SMN')
CALL TPRINT ('SMN')
CALL NODTAB ('SMN')
CALL GPRINT ('SMN')
CALL HRPRINT ('SMN')
CALL TDUMP  ('SMN')
```

where SMN is either a submodel name or 'ALL' to print all submodels (which is the default if the calling argument is absent*). Character delineators (') are required. Note that, unlike the absolute value in HR, conductor heat rates in HRPRINT are positive from input node A to input node B. This result may be of opposite sign to that returned by CONDAT.

7.4.2 Printout of Network Connectivity/Heat Flow Map

Subroutine Name: QMAP

Description: This subroutine computes and writes out a *thermal* network connectivity and heat flow map based upon current nodal temperatures and conductance values. If called prior to a solution routine (e.g., STEADY) a node-conductor connectivity map will be output that gives the values of all conductors connected to each node expressed both as an absolute conductance value and as a percentage of the total connected to the node. If called after a solution routine, the heat flow through each conductor will be output also. QMAP may be used to map a single submodel or all submodels. If the QMAP file name is set in OPTIONS DATA, the map will be written to this file. Output can optionally be routed to the OUTPUT file, although this copy can be suppressed if a QMAP file is named.

Restrictions and Guidance: Because of the volume of output, QMAP calls should all appear in OPERATIONS or PROCEDURE and should never appear in a VARIABLES or FLOGIC block. If called prior to a solution routine, a QMAP call should be preceded by calls to subroutines GVTEMP and GVTIME in order to initialize the conductor values.

* For example, "CALL NODTAB" is equivalent to "CALL NODTAB('ALL')"

Guidance: The ERN temperature reported by QMAP and NODMAP is the temperature the node would be if it were an effective radiation node: if it had no sources, were arithmetic, and if its linear connections are ignored. In other words: $T_{ERN} = [\Sigma_i(G_{ri}T_i^4) / \Sigma_i G_{ri}]^{1/4}$. This may or may not be useful as the sink temperature for a reduced model. It is more a measure of the radiative environment of the node, except that excludes source terms (Q) which might also contain such influences.

For a more concise reporting on the status of nodes and their energy imbalances, consider instead the NODTAB routine documented above.

Calling Sequence:

```
CALL QMAP ('SMN', 'ARGS', NPRNT)
```

where:

SMN	a thermal submodel name in single quotes, or 'ALL' for all submodels
ARGS	character string containing commands (see below)
NPRNT.....	integer OUTPUT print flag: 0 = print to OUTPUT file (the default if this last argument is omitted), 1 = suppress printing to OUTPUT file (but still write to QMAP file if named)

where 'ARGS' can be one of the following values:

'QDAHB'	Heat flow map plus conductance map for all node types
'ALL'	same as above
'Q'.....	Heat flow map only - all node types
'D'.....	Heat flow and conductance map - diffusion nodes only
'A'.....	Heat flow and conductance map - arithmetic nodes only
'B'.....	Heat flow and conductance map - boundary nodes only
'H'.....	Heat flow and conductance map - heater nodes only
'XXXX'	XXXX = some combination of Q,D,A,H and/or B

Note: character arguments must be enclosed by character delineators(').

If both SMN and ARGS are absent, then all data in all thermal submodels will be printed. In other words, "CALL QMAP" is equivalent to "CALL QMAP('ALL','ALL',0)." If either SMN or ARGS is missing, they *both* must be missing.

Subroutine Name: NODMAP

Description: This subroutine is the same as QMAP, but only produces a mapping of one node. This routine can be used in place of QMAP to reduce the amount of outputs. Both heat flow and conductance maps are printed.

Calling Sequence:

```
CALL NODMAP ('SMN', NODE, NPRNT)
```

where:

SMN.....a thermal submodel name in single quotes
NODE.....ID of node to be mapped
NPRNT .....integer OUTPUT print flag: 0 = print to OUTPUT file (default if this argument is absent), 1 = suppress printing to OUTPUT file (but still write to QMAP file if named)

7.4.3 Binary Formats for Files and Folders

NOTE: Binary input and output are undergoing a transition from an older and soon-to-be-obsolete single "SAVE file" format to a newer format: a set of files stored within a compressed subdirectory, or "CSR folder." The term **file** *in phrases such as "SAVE file" or "RSO file" or "RSI file" can refer to a* **CSR folder** *too. In other words, several different CSR folders may be named in OPTIONS for the purposes of output (SAVE, RESAVE, SAVEDB, CRASH) or restarting (RSO, RSI). RESTAR can auto-determine whether its data source is a file or folder. We apologize for any confusion during this transition.*

Routines that work with binary data (SAVE, RESAVE, RESTAR, SAVEDB, etc.) use a single file by default: the traditional "SAVE file." However, this is the default format only to enable backward compatibility with existing text files, which are version independent (unlike a Sinaps *.smdl file or a Thermal Desktop *.dwg file). Sinaps and Thermal Desktop use a newer format, the CSR folder, for newly created models. Those GUIs allow existing models to use the old format, but conversion to the newer format is urged. To "convert" a SINDA/FLUINT text file to the newer format, use the USECSR routine documented below. A call to this routine affects all subsequent binary output streams.

The traditional single file (SAVE file) will become obsolete, and is already frozen: it will not be expanded to include new parameters that will be added in future versions.

Subroutine Name: USECSR

Calling Sequence:

```
CALL USECSR
```

Invoking this routine will change the operation of SAVE, RESAVE, SAVEDB, CRASH, etc. to use the newer CSR folder binary format instead of the traditional single "SAVE file" format.

Use of this routine is strongly recommended, but is unnecessary when using a GUI (e.g., Sinaps or Thermal Desktop) since its invocation is controlled through those programs.

Calling USECSR does not affect binary input such as the RESTAR routine accessing the RSI input; RESTAR auto-determines whether a file or folder has been supplied as a data source.

Restriction: Call only with OPERATIONS before any solution or output routine has been invoked.

Restriction: Do not change formats within a run: do not call USESAVEFILE after calling USECSR if any data has been generated by the run.

Subroutine Name: USESAVEFILE

Calling Sequence:

```
CALL USESAVEFILE
```

Invoking this routine will change the operation of SAVE, RESAVE, SAVEDB, CRASH, etc. to use the traditional single “SAVE file” format instead of the newer CSR folder format.

The only purpose for this routine is to override a call to USECSR that is being inserted by a GUI (e.g., Sinaps or Thermal Desktop).

Restriction: Call only with OPERATIONS before any solution or output routine has been invoked.

Restriction: Do not change formats within a run: do not call USECSR after calling USESAV-EFILE if any data has been generated by the run.

7.4.4 Restart Data Save Subroutine

Subroutine Name: RESAVE

Description: (See Section 4.7) This subroutine writes all program variables (temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays, registers, and flow data) to the binary restart (RSO) file or folder. RESAVE requires that a user’s file name (or CSR folder name) be equated to the name RSO in the OPTIONS DATA. Each time RESAVE is called, a sequential restart record number and current problem time will be printed for future reference. At the user’s option, parts of the program variables (e.g., capacitances, Q-sources) may be left out of the output process.

Refer to Section 7.4.3 for the format options for RESAVE: whether a traditional single “SAVE file” is used or the newer CSR folder.

Restrictions and Guidance: RESAVE should never be called from a VARIABLES or FLOGIC block. Since its output is not restricted to a single submodel, calls to RESAVE should appear in only one OUTPUT CALLS block in a multi-submodel problem. In this case, the user controls the frequency of output to RSO with the control constants OUTPUT and/or ITEROT for the submodel associated with the OUTPUT CALLS block that contains the call to RESAVE. The "ALL" option should be used unless the RSO data set threatens to grow to an unmanageable size. To simplify program control, *named (global) user constants are never saved.*

Caution: To avoid discrepancies between restored data and the register expressions (formulas) used to define any such data, registers should not normally be excluded from the output set.*

Calling Sequence:

```
CALL RESAVE ('ARGS')
```

where `'ARGS'` is a combination of characters that define the types of data to be written to the restart file or folder. It works as follows:

'ALL' . . . . . All parameters are written to RSO (default, if no arguments are provided)
'T' . . . . . Write Temperatures to RSO
'C' . . . . . Write Capacitance values to RSO
'Q' . . . . . Write Q-source values to RSO
'G' . . . . . Write Conductor values to RSO
'N' . . . . . Write control constants to RSO
'U' . . . . . Write *numbered* user constants to RSO
'A' . . . . . Write user arrays and CARRAYs to RSO
'L' . . . . . Write lump PL, TL, XL, and QDOT to RSO (for plots)
'P' . . . . . Write path FR to RSO (for plots)
'M' . . . . . Save advanced mixture and iface data
'F' . . . . . Save remaining FLUINT data
'R' . . . . . Save register names and expressions
'xxxxxx' . . . . . xxxxxx is any combination of T,C,Q,G,U,N,A,L,P,R,M and/or F
'-xxxxx' . . . . . save all data *except* the combination of 'xxxxx'

Examples:

```
CALL RESAVE('ALL')      $ NORMAL USAGE: SAVE EVERYTHING
CALL RESAVE             $ equivalent to the above
CALL RESAVE('T')        $ JUST SAVE TEMPS (e.g., SMALL PLOT FILES)
```

Note: ARGS must always be enclosed in character delineators ('). In order to restart FLUINT models, ARGS must be 'ALL' or include 'LPFM'.

* In prior versions, the exclusion of registers was recommended in some circumstances that no longer exist.

Subroutine Name: CRASH

A variation of RESAVE is available for creating and updating crash files or folders (i.e., simulation states that assist in the recovery from an unexpected run termination). The routine, CRASH, is equivalent to calling RESAVE with an argument of 'ALL', *except that repeated calls overwrite the previously stored snapshot.* Therefore, CRASH can be called an arbitrary number of times without causing the crash file or folder size to grow. Also, the contents (file buffers) are flushed each call, so that a subsequent failure or abort does not render the last saved state invalid.

The cost of a call to CRASH is not completely insignificant; the user must trade-off how often to call it versus how much ground must be retraced in the event of a crash. Calling it from FLOGIC 0 or VARIABLES 0 will maximize both its cost and its 'freshness,' while calling it from OPERATIONS or PROCEDURE will minimize both. OUTPUT CALLS is probably the best compromise location for most uses. Often, not all OUTPUT CALLS blocks are used for all submodels; using such a otherwise unused block enables the user to customize the frequency of calls to CRASH.

Description: CRASH saves one snapshot of the entire model for later use in restarting. Subsequent calls overwrite the previously saved snapshot, conserving file sizes. CRASH is equivalent to a call to RESAVE with an argument 'ALL', except only the most recent snapshot is saved if repeated calls are made. CRASH may be called from any logic block.

The crash file or folder is closed after each call, and is reopened with STATUS='UNKNOWN.' The first call to CRASH will generate an output file message containing the record number: "CRASH FILE RECORD NUMBER NNN." This record number should be used in subsequent calls to RESTAR, with the crash file name input as the RSI file in OPTIONS DATA.

Refer to Section 7.4.3 for the format options for CRASH: whether a traditional single "SAVE file" is used or the newer CSR folder.

Calling Sequences:

```
CALL CRASH(cfname)
```

where:

`cfname` .... crash file name (including path if desired) in single quotes, or CARRAY reference.

On case-sensitive UNIX machines, use the CARRAY option to preserve lower case letters in file names. Otherwise, the file name will be capitalized.

Examples:

```
CALL CRASH('OHNO')      $ LOCAL FOLDER NAME
CALL CRASH(UCA10)       $ CARRAY #10 CONTAINS FOLDER NAME
```

7.4.5 Variable Save Routine

Subroutine Name: SAVE

Description: (See Section 4.7) This subroutine writes all program variables (registers, temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays, and flow data) to the SAVE file or SAVE folder. SAVE requires that a user's file name (or CSF folder name) be equated to the name SAVE in the OPTIONS DATA. At the user's option, parts of the program variables (e.g., capacitances, Q-sources) may be left out of the output process. At the user's option, the control constants and energy balance criteria will be printed when the subroutine is called.

Refer to Section 7.4.3 for the format options for SAVE: whether a traditional single "SAVE file" is used or the newer CSR folder.

Restrictions and Guidance: SAVE should never be called from a VARIABLES or FLOGIC block. Since its output is not restricted to a single submodel, calls to SAVE should appear in only one OUTPUT CALLS block in a multi-submodel problem. In this case, the user controls the frequency of output to SAVE with the control constants OUTPUT and/or ITEROT for the submodel associated with the OUTPUT CALLS block that contains the call to SAVE. The 'ALL' option should be used unless the SAVE file or folder threatens to grow to an unmanageable size. To simplify program control, *named (global) user constants are never saved.*

Caution: To avoid discrepancies between restored data and the register expressions (formulas) used to define any such data, registers should not normally be excluded from the save set.*

Calling Sequence:

```
CALL SAVE ('ARGS',NFLAG)
```

where `'ARGS'` is a combination of characters that define the types of data to be written to the SAVE file or folder. It works as follows:

'ALL' . . . . . All parameters are written to SAVE (default, if no arguments are provided)
'T' . . . . . Write Temperatures to SAVE
'C' . . . . . Write Capacitance values to SAVE
'Q' . . . . . Write Q-source values to SAVE
'G' . . . . . Write Conductor values to SAVE
'N' . . . . . Write control constants SAVE
'U' . . . . . Write *numbered* user constants to SAVE
'A' . . . . . Write user arrays and CARRAYs to SAVE

* In prior versions, the exclusion of registers was recommended in some circumstances that no longer exist.

'L' . . . . . Write lump PL, TL, XL, and QDOT to SAVE (for plots)
'P' . . . . . Write path FR to SAVE (for plots)
'M' . . . . . Save advanced mixture and iface data
'F' . . . . . Save remaining FLUINT data
'R' . . . . . Save register names and expressions
'xxxxxx' . . . . . xxxxxx is any combination of T,C,Q,G,U,N,A,L,P,R,M and/or F
'-xxxxx' . . . . . save all data *except* the combination of 'xxxxx'

Examples:

```
CALL SAVE('ALL',0)      $ NORMAL USAGE: SAVE EVERYTHING
CALL SAVE('ALL')        $ equivalent to the above
CALL SAVE               $ equivalent to the above
CALL SAVE('T')          $ JUST SAVE TEMPS (e.g., SMALL PLOT FILES)
CALL SAVE('-G')         $ SAVE ALL BUT CONDUCTORS
```

Note: ARGS must always be enclosed in character delineators ('). In order to restart FLUINT models, ARGS must be 'ALL' or include 'LPFM'.

The optional NFLAG argument controls whether or not the current values of the control constants are printed when SAVE is called. If NFLAG = 1, the control constants and model energy balance data will be printed. If NFLAG = O (the default if the argument is missing), only the message "SAVE called at TIMEN = xxxxx" will appear.

7.4.6 Saving Data for Parametric Cases

Subroutine Name: SAVPAR

Description: (See Section 4.7) This subroutine saves all program variables (registers, temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays, and all flow data) to an internal program (temporary) folder. Each call to SAVPAR (or SVPART) returns a record number. This is in record number where SAVPAR began its write. Each call to SAVPAR returns a different record number.*

Restrictions and Guidance: Calls to SAVPAR are probably most useful in the OPERATIONS or PROCEDURE blocks. Calls from OUTPUT CALLS may also be made. Calls from a VARIABLES or FLOGIC block are not appropriate. If a solution routine (e.g., TRANSIENT) call was preceded and followed by calls to SAVPAR, this would make the "initial parameters" (before TRANSIENT) and the "final parameters" (after TRANSIENT) available for subsequent calls to RESPAR. To simplify program control, *named (global) user constants are never saved.*

* Subroutines SAVPAR and RESPAR replace the more restrictive Initial Parameters and Final Parameters features used on prior versions of SINDA.

Calling Sequence:

```
CALL SAVPAR (NREC)
```

NREC is an integer variable that identifies the data package that is written to the internal folder. NREC will be used by subroutine RESPAR to restore the data. Each call to SAVPAR or SVPART will return a unique argument. This argument should be a user constant (e.g., NTEST) or an array reference.

Subroutine Name: SVPART

Description: (See Section 4.7) Unlike SAVPAR, which saves *all* program variables to an internal program folder, SVPART lets the user select parts of the model to save. Each call to SVPART (or SAVPAR) returns a record number. This is in record number where SVPART began its write. Each call to SVPART returns a different record number.

Caution: To avoid discrepancies between restored data and the register expressions (formulas) used to define any such data, registers should not normally be excluded from the data set.*

Guidance: Calls to SVPART are probably most useful in the OPERATIONS or PROCEDURE blocks. Calls from OUTPUT CALLS may also be made. Calls from a VARIABLES or FLOGIC block are not appropriate. To simplify program control, *named (global) user constants are never saved.*

Calling Sequence:

```
CALL SVPART ('ARGS', NREC)
```

where `'ARGS'` is a combination of characters that define the types of data to be written to the restart file. It works as follows:

'ALL' . . . . . All parameters are saved (same as using SAVPAR)
'T' . . . . . Save temperatures
'C' . . . . . Save capacitance values
'Q' . . . . . Save Q-source values
'G' . . . . . Save Conductor values
'N' . . . . . Save control constants
'U' . . . . . Save *numbered* user constants
'A' . . . . . Save user arrays and CARRAYs

* In prior versions, the exclusion of registers was recommended in some circumstances that no longer exist.

'L' . . . . . Save lump PL, TL, XL, and QDOT
'P' . . . . . Save path FR
'M' . . . . . Save advanced mixture and iface data
'F' . . . . . Save remaining FLUINT data
'R' . . . . . Save register names and expressions
'xxxxxx' . . . . . xxxxxx is any combination of T,C,Q,G,U,N,A,L,P,R,M and/or F
'-xxxxx' . . . . . save all data *except* the combination of 'xxxxx'

Examples:

```
CALL SVPART('ALL',MTEST)$ SAVE EVERYTHING (SAVE AS "SAVPAR(MTEST)")
CALL SVPART('T',NTEST)  $ JUST SAVE TEMPS (e.g., SMALL PLOT FILES)
CALL SVPART('-G',NREC)  $ SAVE ALL BUT CONDUCTORS
```

Note: ARGS must always be enclosed in character delineators ('). In order to restart FLUINT models, ARGS must be 'ALL' or include 'LPFM'.

NREC is an integer variable that identifies the data package that is written to the internal file. NREC will be used by subroutine RESPAR to restore the data. Each call to SVPART or SAVPAR will return a unique argument. This argument should be a user constant (e.g., NTEST) or an array reference.

7.4.7 Temperature Save Subroutine

Subroutine Name: TSAVE

Description: This subroutine saves nodal temperatures and problem time on a file (or CSR folder) for use in post processors such as temperature history plot programs. At the users option, the control constants and energy balance criteria will be printed or not printed when the subroutine is called. TSAVE is a specialized call to the SAVE subroutine. TSAVE requires that a user's file name (or CSR folder name) be equated to the name SAVE in OPTIONS DATA.

Refer to Section 7.4.3 for the format options for TSAVE: whether a traditional single "SAVE file" is used or the newer CSR folder.

Restrictions and Guidance: A call to TSAVE writes the temperatures for the entire model (all thermal submodels) to the SAVE file or SAVE folder. Each set of temperatures is preceded by the number of active nodes in the model and the problem time, TIMEN. Calls to TSAVE are inappropriate in any VARIABLES block. Since all the temperatures are written, TSAVE should be called from only one OUTPUT CALLS block in a multi-submodel problem. The values of control constants OUTPUT and/or ITEROT for this submodel will control the frequency of TSAVE outputs.

Calling Sequence:

```
CALL TSAVE (NFLAG)
```

The optional NFLAG argument controls whether or not the current values of the control constants are printed when TSAVE is called. If NFLAG = 1, the control constants and model energy balance data will be printed. If NFLAG = 0 (default if the argument is absent), only the message “SAVE called at TIMEN = xxxxx” will appear.

7.4.8 Printing Minimum and Maximum Temperatures

Subroutine Name: TMNMX

Description: This subroutine prints the maximum and minimum temperatures of either all the nodes in a submodel or of a specified list of nodes in one or more thermal submodels. The user may also specify one or more time intervals for TMNMX to be in effect. The minimum and maximum temperatures are printed at the end of the time interval(s) specified.

TMNMX should be called from OUTPUT CALLS unless the maximum or minimum values are expected to exist between output intervals, in which case TMNMX might be placed in VARIABLES 2. Only one set of input parameters can be used in any one SINDA/FLUINT run: *multiple calls with different input parameters (e.g., two different lists of nodes) are not allowed.*

Array ATM is an array which contains the floating point times at which the maximums and minimums are to be output. For example, if the argument were “A1” and array 1 contained the following entries: 1 = 1.0, 3.0, 3.5, then three outputs would occur: the first at TIMEN = 1.0 covering the interval TIMEO to 1.0; the second at 3.0 for the interval 1.0 to 3.0; and the final output at 3.5 for the period 3.0 to 3.5. For a printout at every interval, enter zero (0.0) for ATM.

The next three arguments describe the node numbers for which the maximum and minimum temperatures are desired. These node numbers are input via a CARRAY DATA block. The SMN argument specifies which submodel contains the CARRAY DATA block that contains the node numbers. NCA1 is the actual number (user-provided ID) of the first CARRAY to use. NUMCAS is the number of *consecutive* CARRAYs to use if all the instructions do not fit in a single line. The numeric identifiers of the additional CARRAYs are irrelevant, but they must be input in the order in which they are to be processed, as if they were “continuations” of the first starting CARRAY. Use ALL instead of a node number list to check all of the nodes in any one submodel.

CARRAY Format:

```
HEADER CARRAY DATA, SMN
       nca1 = smn1, n1, n2 ..., smn2, n1, n2,
       nca2 = n3, n4, ..., smn3, ALL
```

Calling Sequence:

```
M     CALL TMNMX (ATM(IC),'SMN',NCA1,NUMCAS)
```

where:

ATM ..... an array of output times or 0 for all output times
SMN ..... submodel name containing CARRAY NCA1
NCA1 ..... actual number of first CARRAY to use
NUMCAS .... number of *sequential* CARRAYs to use

Example 1: The following shows the relevant data blocks and their use with TMNMX.

```
HEADER CARRAY DATA, MAIN
      22 = MOD1, 100, 200, 300, MOD2, 100, 200, 300
      25 = 400, 500, MOD3, ALL
C   The word ALL may appear to indicate all nodes in a submodel
      319 = MAIN,ALL
HEADER ARRAY DATA, TIMES
      10 = 0.0, 1.0, 2.0
HEADER OUTPUT CALLS, OTHER
      CALL TMNMX(TIMES.A10,'MAIN',22,2)$ Use lists in #22 and #25
```

Example 2: An alternative usage based on the same model (above).

```
HEADER OUTPUT CALLS, OTHER
      CALL TMNMX(0.0,'MAIN',319,1) $ Use #319 (all nodes in MAIN)
```

7.4.9 Array Data Printout

Subroutine Name: PRINTA

Description: This subroutine allows the user to print out an array of values, five to the line. The integer array length N and the first data value location must be specified. Each value receives an indexed label. The user must supply a six-character alphanumeric word L to be used as a common label and an integer value M to begin the index count.

Restrictions: The array values to be printed must be floating point numbers. If L is supplied as a literal Hollerith data value (instead of a reference to a user constant containing same), it must be entered in character type notation (e.g., 'TEMP').

Calling Sequence:

```
CALL PRINTA (L,A(DV),N,M)
```

If the label L were the word 'TEMP', N were 3 and M were 6, the line of output would look as follows:

```
TEMP   (  6)  value TEMP     (  7)  value TEMP       (  8)  value
```

Subroutine Name: GENOUT

Description: This subroutine prints out whole arrays or parts of arrays, printing up to 10 values per line. The numbers may be real, integer, or both.

Calling Sequence:

```
CALL GENOUT(A(IC),ISTRT,ISTP,'NAME')
```

where:

A. . . . . array containing numbers to be printed (real, integer, or mix)
ISTRT . . . . . location of the first value to be printed (ISTRT .GE. 2).
ISTP. . . . . location of the last value to be printed (ISTP .GT. ISTRT and .LE. the integer count of A, NA(IC), plus one)
'NAME' . . . . . title of up to 22 characters for identification

Guidance: Recall that the first cell in every SINDA array is equivalenced to its integer count. To print an entire array, use ISTRT=2 and ISTP=NA(ic)+1 where ic is the array ID. NAME is printed starting in column 1, such that the first character in NAME may be used for carriage control on an appropriate printer.

7.4.10 Numerical Differencing Characteristics Printout

Subroutine Name: STNDRD

Description: Subroutine STNDRD causes a line of output to be printed giving the present time, the last time step used, the most recent CSGMIN value, the maximum diffusion temperature change calculated over the last time step and the maximum relaxation change calculated over the last iteration. ANN refers to the actual node number that caused the appropriate value. The line of output looks as follows:

```
  *    *    *    *
TIME     DTIMEU     CSGMIN(ANN)     TEMPCC(ANN)     RELXCC(ANN)
```

Restriction: Same as TPRINT.

Calling Sequence:

```
CALL STNDRD ('SMN')
```

where SMN is a submodel Name

7.4.11 Sorted Temperature Print

Subroutine Name: SORTPR

Description: This subroutine prints out nodal temperatures. These temperatures may appear sorted by node number or by temperature.

Restrictions and Guidance: This routine is usually called from OUTPUT CALLS data, but is not restricted to this. A submodel name must be specified. To print out all submodels the word 'ALL' may appear in place of a submodel name. The print flag IFLAG must be 0 or 1. If IFLAG=0 temperatures will be sorted by node number. If IFLAG=1 output will be sorted by temperature.

Calling Sequence:

```
CALL SORTPR ('SMN', IFLAG)
```

where:

`SMN` ..... a thermal submodel name ('ALL' is the default if the argument is omitted)
`IFLAG`..... print flag: 0 = print by node, 1 = print by temperature

7.4.12 Save Temperatures for Boundary Temperature Driving

Subroutine Name: BNDGET

Description: Subroutine BNDGET gets temperatures from one or more SAVE files or SAVE folders and writes them to an ASCII output file. The temperatures to fetch are specified by a list of submodel names and node numbers appearing in CARRAY DATA. The call to BNDGET also specifies the start and end times. In this way data for a specific segment of time may be extracted from the SAVE file or folder. Data from any number of SAVE files or folders may be extracted by using multiple calls to BNDGET. Multiple calls may also be used to extract data for a number of different time segments on the same SAVE set. The data may be transferred to a number of independent output files if a different FILOUT name is used on each call. A number of sets of data may be sent to a single output file if FILOUT is the same for a series of calls, because FILOUT is not rewound prior to the write commands. In this case, it is the users responsibility to be sure that there are no overlaps among the time segments written out.

Restriction: BNDGET should be called in OPERATIONS or PROCEDURE. It should be called after all relevant solution routines have been performed if it is to use a currently active SAVE file or SAVE folder. It can also be used during initialization to prepare a file for BNDDRV (Section 7.5.2) from existing SAVE files or SAVE folders.

The user should be aware that if BNDGET is called more than once in a single run, a new 'FILOUT' name must be used for each call. Otherwise, multiple files will be written to the output file and BNDDRV will fail to work. BNDGET can use any set of CARRAY data. The only restriction is that the arrays to be used must appear sequentially in the names block.

Node numbers are input via a CARRAY DATA block. The SMN argument specifies which submodel contains the CARRAY DATA block that contains the node numbers. NCA1 is the actual number of the first CARRAY to use. NUMCAS is the number of *consecutive* CARRAYs to use if all the instructions do not fit in a single line. The numeric identifiers of the additional CARRAYs are irrelevant, but they must be input in the order in which they are to be processed, as if they were "continuations" of the first starting CARRAY. Use ALL instead of a node number list to check all of the nodes in any one submodel.

CARRAY Format:

```
HEADER CARRAY DATA, SMN
       nca1 = smn1, n1, n2 ..., smn2, n1, n2,
       nca2 = n3, n4, ..., smn3, ALL
```

Calling Sequence:

```
M       CALL BNDGET (NUMCAS, SMN, NCA1, FILIN, FILOUT,
     .      TIME1, TIME2, TIMDEL, FIGNAM, NFLAG)
```

where:

NUMCAS . . . . Number of *sequential* CARRAYs to use (integer)
SMN. . . . . Submodel name where CARRAY data appears (character)
NCA1. . . . . Actual number of 1st CARRAY to use (integer)
FILIN. . . . . A SAVE file or SAVE folder name for input (character)
FILOUT. . . . . name of output file (character)
TIME1 . . . . . Start time of read from FILIN (real)
TIME2 . . . . . End time of read (real)
TIMDEL . . . . . A delta to add to read times (real)
FIGNAM . . . . . configuration name from relevant BUILD record or the word 'ALL' (character)
NFLAG . . . . . Print flag: if not 0 then writes TIME1, TIME2, FILIN, TIMDEL (integer)

Example: The following shows the relevant data blocks and their use with BNDGET.

```
HEADER OPTIONS DATA
         SAVE = NAME.TSV
HEADER CARRAY DATA, MAIN
         1 = TEST
         2 = MOD1, 100, 200, 300, MOD2, 100, 200, 300
         3 = 400, 500, MOD3, ALL
C   The word ALL may appear to indicate all nodes in a submodel
         4 = TEST
HEADER OPERATIONS
BUILD MODELS, MAIN, MOD1, MOD2, MOD3
               .
               .
               .
      CALL BNDGET(2,'MAIN',2,'NAME.TSV','NAME.BND',
     .      0.0,1.0,0.0,'MODELS',0)
```

7.4.13 Print Heater Node Q Values

Subroutine Name: HNQPNT

Description: This subroutine internally calls HNQCAL. It then prints out the Q values calculated for the heater nodes. In this context, “heater nodes” includes any diffusion or arithmetic nodes that have been temporarily held as heater nodes using calls to HTRNOD or HTRMOD.

Restrictions and Guidance: This routine is callable from OUTPUT CALLS but is not restricted to this. A thermal submodel name must be specified and should agree with a submodel name appearing on the a BUILD card.

Calling Sequence:

```
CALL HNQPNT ('SMN')
```

where SMN is a thermal submodel name, or use the word ’ALL’ to print all thermal submodels (the default if no arguments are used). Character delineators (’) are required.

7.4.14 Print Routine Controller

Subroutine Name: PRINT

Description: Given the name of a *thermal* print routine to control, this routine will control the number of times the print routine is called. In addition, if a temperature print routine is being controlled the name of a temperature conversion routine may also be passed in. The passed in routine will perform the required conversions for output.

Restrictions and Guidance: This routine is callable from OUTPUT CALLS but is not restricted to this. A thermal submodel name must be specified and should agree with a submodel name appearing on the BUILD card. The submodel name is the only character argument. All other arguments are either subroutine references or integers.

Calling Sequence:

```
CALL PRINT ('SMN', PFUNC, NSKIPS, CONV)
```

where:

SMN . . . . . a thermal submodel name or the word 'ALL'. Character delineators (') are required.

PFUNC . . . . . The name of the print routine to control. Valid entries are: TPRINT,QPRINT,CPRINT,GPRINT. This argument must *not* be in quotes.

NSKIPS . . . . . Number of times PRINT should be called with SMN before PFUNC is called.

CONV. . . . . The name of a conversion routine. Valid entries are: CTF, CTK, FTC, FTK, FTR, KTC, KTF, and RTF. Enter a 0.0 for no conversion. This argument must *not* be in quotes.

7.4.15 Tabulate Registers and Expressions

Subroutine Name: REGTAB

Description: Tabulates registers, their values, and defining expressions to the output file. REGTAB has no arguments.

Calling Sequence:

```
CALL REGTAB
```

7.4.16 Reset Page Header Information

Subroutine Name: PAGHED

Description: This subroutine gives the user access to the number of lines on a page and the titles on the page. The user may also use this routine to add extra lines to the standard output file NOUT without disrupting the starting point of new pages.

MLINE is the maximum number of output lines per page, and is normally set once in OPTIONS DATA. LINDEL allows the user to add lines to the output file NOUT and to increment the line counter accordingly to avoid disrupting the starting point for new pages. TITLE, the main page title, is similarly normally set once in OPTIONS DATA. STITLE is normally set by solution routines and mapping routines.

Calling Sequence:

```
CALL PAGHED(MLINE, LINDEL, TITLE, STITLE)
```

where:

`MLINE.....` the number of lines per page (an integer). If MLINE is 0 then the value of MLINE will not be changed.

`LINDEL ....` a number to increase the current line counter. If the current line counter plus LINDEL is greater or equal to MLINE then a new page will be started, using current MLINE, TITLE, and STITLE values as supplied by the current call to PAGHED.

`TITLE.....` the main page title (single quotes are required, or TITLE can be a CARRAY entry such as UCA10). If this argument is a space, (' '), the current TITLE will not be changed.

`STITLE ....` the page subtitle (single quotes are required, or STITLE can be a CARRAY entry such as UCA10). The subtitle is set to the name of the current solution routine or mapping routine at the start of those routine, so any calls to PAGHED to customize the subtitle will be overwritten at the start of those routines. If this argument is a space, (' '), the current STITLE will not be changed.

7.4.17 Creating Additional User Files

Subroutine Name: USRFIL

Description: This subroutine opens a named file and returns a valid unit number. The returned unit number can be used in subsequent WRITE or READ statements, and hence should appear in HEADER USER DATA, GLOBAL or as an integer register. This routine is intended to extend the current USER1 and USER2 options.

USRFIL can also be used to return a valid unit number without actually opening a file. For example, if the user wishes to open and use an unformatted (binary) file, the USRFIL routine can be called with a fake file name of 'UNITONLY' or 'unitonly.' It will then return a valid, unused unit number that the user can employ in a subsequent Fortran OPEN statement.*

Restrictions: Do not attempt to open the same file twice. The processor will abort if the file could not be opened. Use USRFIL2 (below) instead to avoid such aborts. Users are cautioned that many operating systems impose a quota on the maximum number of files that may be opened at one time; exceeding this limit will cause the system to abort the processor.

Calling Sequence:

```
CALL USRFIL (NUNIT, FNAME, STATUS)
```

where:

NUNIT ..... the returned unit number (an integer). The variable that will contain the unit number should appear in HEADER USER DATA, GLOBAL or as an integer register.

FNAME ..... the name of the file to be opened (single quotes are required, or FNAME can be a CARRAY entry such as UCA10).
The file name 'UNITONLY' is used as a flag signaling the routine to return a new unused unit number without opening it as a file.

STATUS ..... the status flag to use in the Fortran OPEN statement, in single quotes:
'OLD' if the file exists and it can be overwritten, or
'NEW' if it shouldn't exist yet, producing an abort if it does exist to protect it from being overwritten, or 'UNKNOWN' (default if argument is omitted)
No OPEN statement will be executed if FNAME is 'UNITONLY'

* Do not use an OPEN statement directly without having called USRFIL or USRFIL2 with 'UNITONLY'. SINDA/FLUINT reserves many units for its own use, and it employs temporary working directories that will be deleted (along with any local user-created files) upon program completion. USRFIL and USRFIL2 avoids both of these difficulties.

Examples:

```
CALL USRFIL(ntest,'localFile.txt','unknown')
CALL USRFIL(nuser3,hotmod.UCA22,'old')
CALL USRFIL(myfile,'c:\cr\good\stuff.inc','unknown')
CALL USRFIL(newone,'unitonly','unknown')
CALL USRFIL(newone,'unitonly')   $ equivalent to above
```

Subroutine Name: USRFIL2

Description: This subroutine is the same as USRFIL (above), except that it does not abort if the file could not be opened successfully. Instead, it returns an integer flag denoting success or failure.

Restrictions: Do not attempt to open the same file twice.

Calling Sequence:

```
CALL USRFIL2 (NUNIT, FNAME, STATUS, MESS)
```

where:

NUNIT..... the returned unit number (an integer). The variable that will contain the unit number should appear in HEADER USER DATA, GLOBAL or as an integer register.

FNAME..... the name of the file to be opened (single quotes are required, or FNAME can be a CARRAY entry such as UCA10).
The file name 'UNITONLY' is used as a flag signaling the routine to return a new unused unit number without opening it as a file.

STATUS .... the status flag to use in the Fortran OPEN statement, in single quotes:
'OLD' if the file exists and it can be overwritten, or
'NEW' if it shouldn't exist yet, producing an abort if it does exist to protect it from being overwritten, or 'UNKNOWN' (default if this argument is omitted)
No OPEN statement will be executed if FNAME is 'UNITONLY'

MESS ..... Returned status flag: 0 if successful, 1 if not.

Examples:

```
CALL USRFIL2(ntest,'localFile.txt','unknown',iostat)
CALL USRFIL2(nuser3,hotmod.UCA22,'old',iostat)
CALL USRFIL2(myfile,'c:\cr\good\stuff.inc','unknown',nother)
CALL USRFIL2(newone,'unitonly','unknown',metoo)
```

7.4.18 Model Size Information

Subroutine Name: SIZETAB

Description: This subroutine prints out size information to the output file for the current model: number of nodes, conductors, lumps, paths, etc.

Restrictions: When using the dynamic mode in Thermal Desktop, additional conductors and ties will exist in the model that will not be listed by SIZETAB.

Calling Sequence:

```
CALL SIZETAB
```

7.4.19 Changing OUTPUT and SAVE Destinations Dynamically

This section describes routines that allow files or folders named in OPTIONS DATA to be changed during processor execution. Contents of the previous files or folders are saved.

Also, another related utility, CLRSAVE, can be called to clear the contents of the current file or folder.

Subroutine Name: CHGOUT

Description: This subroutine allows the user to change OUTPUT files, which are originally named in OPTIONS DATA, as many times as needed during a run. This switching is convenient for storing results of parametric runs in separate files, for example.

Caution: The file named in CHGOUT will be filled at the starting position in subsequent output operations: the named file is rewound. This means that calling CHGOUT twice with the same file name (or calling it with the same name as was specified in OPTIONS DATA) will erase prior contents. This is permitted as a secondary usage (namely, the clearing the current contents of the OUTPUT files), but if repeated calls are executed by mistake, a loss of data will result and no warnings will be produced.

Calling Sequence:

```
CALL CHGOUT (FNAME)
```

where:

FNAME..... the name of the new file to be opened (single quotes are required, or FNAME can be a CARRAY entry such as UCA10). This can be a local file name (relative to the input file directory) or an absolute name including path information.

Examples:

```
CALL CHGOUT('newfile.out')
CALL CHGOUT('d:\action\run10\newfile.out')
CALL CHGOUT(hermod.uca301)
```

Subroutine Name: CHGSAVE

Description: This subroutine allows the user to change SAVE files or SAVE folders (CSR folders), which are originally named in OPTIONS DATA, as many times as needed during a run. This switching is convenient for storing results of parametric runs in separate locations, for example.

Caution: The file or folder named in CHGSAVE will be filled at the starting position in subsequent SAVE operations. This means that calling CHGSAVE twice with the same file or folder name (or calling it with the same name as was specified in OPTIONS DATA) will erase prior contents. This is permitted as a secondary usage (namely, the clearing the current contents of the SAVE sets), but if repeated calls are executed by mistake, a loss of data will result and no warnings will be produced.

Calling Sequence:

```
CALL CHGSAVE (FNAME)
```

where:

FNAME..... the name of the new file or CSR folder to be opened (single quotes are required, or FNAME can be a CARRAY entry such as UCA10). This can be a local file or folder name (relative to the input file directory) or an absolute name including path information.

Examples:

```
CALL CHGSAVE('newfile.sav')
CALL CHGSAVE('..\newfile.sav')
CALL CHGSAVE(hismod.uca311)
```

Subroutine Name: CLRSAVE

A call to CLRSAVE (no arguments) at any point in the run clears the current SAVE file or folder such that future calls to SAVE start at the beginning record. CLRSAVE should be used with caution since, once called, the any previously stored data will be lost.

Calling Sequence:

```
CALL CLRSAVE
```

See also CHGSAVE (above) to swap SAVE files or folders during a run.

7.4.20 Sink Temperature Utilities

A *sink temperature* is an approximation method that may be used to reduce the complexity of a model by replacing parts of it with an effective reduced (and invariant) network: a boundary condition for the object of interest.

The sink temperature approximation is normally used (1) to eliminate recalculation of radiative environments in order to speed up parametric sweeps and sensitivity studies, and (2) to enable a designer of a component or subsystem to work independently of a full system model (which is presumed to be independent of small variations in the component model). Often, sink temperatures representing a full system response are provided to a component designer (who often works for a distinct organization lacking access to the full system model) as a means of communicating design requirements. Sink temperature sets must be regenerated periodically throughout the design cycle.

Two utilities are provided for generating sink temperatures and their associated conductances:*

TSINK1 . . . . . Generate one conductance/sink pair for a single node.
TSINK . . . . . Generate a set of conductances and one or more sink temperatures for a group of nodes, output either as arrays or as input files for future runs of a component-level model.

The latter routine is considerably more complex, not only because more data is flowing in and out, but also because the user must pay more attention to which nodes are to be retained and which will be eliminated in a future component-level model, and whether one sink can be used to represent to entire set, or whether each node in the reduced model should have its own sink temperature.

Subroutine Name: TSINK1

Description: TSINK1 calculates the sink temperature (and the conductance to that sink) for a single node. See TSINK for multiple nodes.

TSINK1 should be called in OPERATIONS (normally after a solution routine), or perhaps within VARIABLES 2 or OUTPUT CALLS. It should *not* be called from within VARIABLES 0 or 1, since temperatures have not yet been updated at those points.

When called, TSINK1 uses the previous solution results for the current BUILD configuration.

Within TSINK1, the user has the options of:

1. Including or excluding heat rate terms (nodal Q)

* Traditionally only sink temperatures have been produced, often because the conductance was known. However, to replace a network with an effective invariant subset that matches energy flows requires both a sink and a conductance. In large software generated models (e.g., Thermal Desktop), these conductances are themselves a valuable result. Modification of the conductances to represent changes in emissivity, conductivity, etc. can still be performed using (initially unit) registers as multiplying factors. Or, the user can simply ignore the generated conductance data and return to traditional methods if desired.

2. Excluding nodes (by submodel) from being lumped into the effective sink: those nodes which will be preserved in the reduced model, for example (and are therefore referred to as the “keep list”).

3. Including only radiative terms, or only linear terms, or both. Depending on this selection, the user can treat the resulting sink as either a linear or radiative sink.

For example, if only radiation terms are included (mode=’R’), then the resulting sink is also purely radiative. For node i connecting to NR other nodes via radiative connections, the resulting conductance and sink temperature are:

$$\hat{G}_{s,\,i} = \sum_{j\,=\,1}^{NR} \langle \hat{G}_{ij} \rangle$$

$$T_{s,\,i} = \left[\frac{\sum_{j\,=\,1}^{NR} (\sigma \hat{G}_{ij} T_j^{\,4}) + Q_m Q_i}{\sigma \hat{G}_{s,\,i}} \right]^{\frac{1}{4}}$$

For simplicity, the above formula as well as others in this section assume temperatures in absolute units: K or R.

Q_m is a multiplier allowing the user to include (Q_m=1.0) or exclude (Q_m=0.0) source terms. Source terms should be included, for example, if they represent external (environmental) heating rates that are not to be recalculated in future (reduced model) runs, otherwise they would be accounted for twice. Source terms should therefore be excluded if the future runs in which the sinks are applied also contain environmental heating. Note that the sink temperature routines cannot distinguish between external heating and (say) thermostatic heater power since both are summed into the Q term.

If all nodes to which node i is connected will be collapsed into the effective sink (e.g., keeplist=’none’), then all radiation conductances are considered in the above example and the returned radiative conductance to the sink (G_s, a radiative term with units of area = power/(σ∗degree^4), as indicated by the diacritical mark in the above equation) should be approximately equal to εA, where ε is the surface emissivity and A is the surface area.*

Otherwise, the user may elect to name (by submodel) the nodes that will be *retained in the reduced model* (along with the existing conductances to those nodes). Those nodes are *excluded from the above summation*, so the returned G_s should then be less than εA. Note that if all nodes are excluded or no appropriate linear or radiation connections exist, then the returned result will be zero G_s and (as an error flag) T_s = -999.0.

* This is only strictly true when RADKs to inactive surfaces and to the surface itself are included.

If only linear terms* are included (mode='L'), a linear (conductive) sink results. For node i connecting to NL other nodes via linear connections:

$$G_{s,\,i} = \sum_{j\,=\,1}^{NL} \langle G_{ij} \rangle$$

$$T_{s,\,i} = \left[\frac{\sum_{j\,=\,1}^{NL} (G_{ij} T_j) + Q_m Q_i}{G_{s,\,i}} \right]$$

The lack of diacritical marks for the G_s term denotes that the units for this conductance are power/degree: a linear conductance.

As a final example, consider a linear sink that includes both radiative and linear terms (mode='B'), where there are NL+NR nodes to which node i connects (that are not part of the "keep list"). In this case, *the resulting sink will be linear*:

$$G_{s,\,i} = \sum_{j\,=\,1}^{NR} \langle \sigma \hat{G}_{ij} \rangle (T_i^{\,2} + T_j^{\,2})(T_i + T_j) + \sum_{j\,=\,1}^{NL} \langle G_{ij} \rangle$$

$$T_{s,\,i} = \left[\frac{\sum_{j\,=\,1}^{NR} (\sigma \hat{G}_{ij} T_j^{\,4}) + \sum_{j\,=\,1}^{NL} (G_{ij} T_j) + Q_m Q_i}{G_{s,\,i}} \right]$$

However, *a better approximation would be to produce two sink temperatures*: one combining linear terms into a linear sink, and another combining radiative terms into a radiative sink. This can be accomplished via two calls to TSINK1: one with mode='R' and one with mode='L'.

Calling Sequence:

```
CALL TSINK1(smn, node, keeplist, mode, Qm, Gs, Ts)
```

* FLUINT ties are considered part of the linear connections, and QTIEs are extracted from the nodal Q term.

where:

smn. . . . . .submodel name containing node, single quotes
node . . . . . .ID of node (integer)
keeplist. . . . . .comma-delimited character string (single-quotes) containing submodels that are to be *retained* in the reduced model and are therefore to be *excluded* from the sink calculation:* only connections to submodels *not* in this list will considered as part of the G_s and T_s calculation.
'NONE' is a valid input: all connections will be lumped into the sink.
mode . . . . . .character string directing operating mode:
'R' ... consider only radiation conductors ("RADKs")
'L' ... consider only linear conductors and FLUINT ties
'B' ... consider both radiation and linear, and treat the sink as linear
Qm . . . . . .input heat rate multiplier (real): 0.0 to exclude nodal Qs, 1.0 to include them. Other values are legal as well, perhaps as needed to exclude internal heating but include external heating. Generally, if Qm=1.0 the user should make sure that heat rates are not applied a second time in a future (reduced) model since they have already been accounted for in the sink temperature.
Gs. . . . . .returned conductance (real, user units) between the node and sink temperature, either power/temperature or power/temperature^4 depending on the value of *mode* and the control constant SIGMA. G_s is a radiation conductance if mode='R', otherwise it is a linear conductance.
Ts . . . . . .returned linear or radiative sink temperature (real, user units: R, F, C, or K)

Examples:

```
C "Traditional" radiative sink, with Q included (Qm=1.0) and linear ignored:
      CALL TSINK1('battery', 103, 'none', 'R', 1.0, GTEST, TTEST)

C A linear sink, with Q excluded (Qm=0.0) and other parts of the same
C component not included in the calculation (they will be retained)
      CALL TSINK1('batt1', 103, 'batt1,batt2,batt3', 'L', 0., GTEST, TTEST)
```

Subroutine Name: TSINK

Description: TSINK calculates the sink temperature (and the conductance to that sink) for multiple nodes within a single submodel. See TSINK1 for single nodes.

The product of the TSINK routine may be selected to be either a pair of pre-sized arrays or a pair of files containing data in SINDA/FLUINT input format.

* Caution: In the Thermal Desktop Dynamic Mode, dynamic conductances cannot be excluded from being lumped into the sink. The user should not use the dynamic mode when generating sink information, or should exclude relevant submodels from being dynamically regenerated.

TSINK should be called in OPERATIONS (normally after a solution routine), or perhaps within VARIABLES 2 or OUTPUT CALLS. It should *not* be called from within VARIABLES 0 or 1, since temperatures have not yet been updated at those points. It is normally called once per run; multiple calls will overwrite previous results for array output, or be appended to previous results for file output.

When called, TSINK uses the previous results for the current BUILD configuration.

Within TSINK, the user has the options of:

1. Including or excluding heat rate terms (nodal Q)
2. Excluding nodes (by submodel) from being lumped into the effective sink: those nodes which will be preserved in the reduced model, for example (and are therefore referred to as the “keep list”). All nodes included in the current TSINK call are assumed by default to be included in the keep list and are therefore excluded from being lumped into the sink terms.
3. Including only radiative terms, or only linear terms, or both.* Depending on this selection, the user can treat the resulting sink as either a linear or radiative sink.
4. Storing results in user-provided arrays, or writing input files (perhaps to be used in later runs).

If all connected nodes are excluded or no appropriate connections exist, the returned result will be zero G_s and (as an error flag) T_s = -999.0. *See TSINK1 for example calculations.*

Calling Sequence:

```
CALL TSINK(smn, list, keeplist, mode, Qm, GUA, TUA)
```

where:

smn . . . . . submodel name containing node, in single quotes
list . . . . . this integer argument can be one of the following:
0 ... meaning all diffusion and arithmetic nodes in submodel;
a SINDA integer array reference such as smn.NA1; or
the ID of if a single node (integer)

* Ideally, two calls to TSINK should be made for nodes with both linear and radiative connections: one with mode=’R’ and another with mode=’L’. Refer to the TSINK1 routine for more details.

keeplist. . . . . comma-delimited character string (single-quotes) containing submodels that are to be *retained* in the reduced model and are therefore to be *excluded* from the sink calculation:* only connections to submodels *not* in this list will considered as part of the G_s and T_s calculation. The list of nodes provided above (smn, list) is automatically included in the keep list.
'NONE' is a valid input: all connections will be lumped into the sink, excepting those in the input list above.
'NONE!" may be used to override even this exclusion, such that *no* connections are ignored.

mode . . . . . character string directing operating mode:
'R' ... consider only radiation conductors ("RADKs")
'L' ... consider only linear conductors and FLUINT ties
'B' ... consider both radiation and linear, and treat the sink as linear
If *mode* contains a '1' ... 'R1' or 'B1' for example ... then a single sink temperature is produced: the same sink is to be applied to all nodes listed. (This option is present since this is a traditional approach. However, one sink per node is more generally accurate.) This choice affects the TUA argument: it must be a real variable instead of an array reference or a file unit number. This choice does *not* affect the GUA argument, which always must be either an array reference or a file unit number, and can *never* be a single real variable since multiple conductances are always generated.

Qm . . . . . input heat rate multiplier (real): 0.0 to exclude nodal Qs, 1.0 to include them. Other values are legal as well, perhaps as needed to exclude internal heating but include external heating. Generally, if Qm=1.0 the user should make sure that heat rates are not applied a second time in a future (reduced) model since they have already been accounted for in the sink temperature.

GUA. . . . . either the integer file unit or the real SINDA array reference to contain the returned conductances (real, user units) between the nodes and sink temperatures.
The units of these conductors will either be power/temperature or power/temperature^4 depending on the value of *mode* and the constant SIGMA. The conductances are radiation if mode='R', otherwise they are linear.
If an array is specified, the size (perhaps reserved using the SPACE command in ARRAY DATA) should be N where N is the number of nodes listed in the first two arguments (smn, list).

TUA. . . . . either the integer file unit or the real SINDA array reference to contain the returned linear or radiative sink temperatures (real, user units: R, F, C, or K).
If an array, the size (perhaps reserved using the SPACE command in ARRAY DATA) should be N+1 where N is the number of nodes listed in the first two arguments (smn, list). The final value is the total effective sink temperature, hence the requirement for N+1 cells instead of N.
As long as '1' does not appear in *mode*, then TUA must match GUA: if

* Caution: In the Thermal Desktop Dynamic Mode, dynamic conductances cannot be excluded from being lumped into the sink. The user should not use the dynamic mode when generating sink information, or should exclude relevant submodels from being dynamically regenerated.

GUA refers to a file unit, TUA must also refer to a separate file unit. If GUA refers to a SINDA array, then TUA should also refer to a separate SINDA array. If on the other hand *mode* contains a '1', then TUA must be a single real variable independent of the choice of GUA.

To summarize, TSINK may be used in either an array or file mode for output. *In the array mode*:

1. GUA must be a SINDA array sized to contain the resulting calculations. If there are N nodes listed in the first two arguments, then the receiving array should be sized (perhaps using the SPACE command in ARRAY DATA) to contain N values, where the i^{th} conductance value will correspond to the i^{th} node in the input list.
2. If *mode* contains '1' (a single sink temperature is desired) then TUA must be a real variable which will receive the single effective sink temperature for the set of nodes.
3. Otherwise, if *mode* does *not* contain '1' (and therefore multiple sink temperatures are desired), then TUA must also be a SINDA array sized to contain the resulting calculations. If there are N nodes listed in the first two arguments, then the receiving array should be sized (perhaps using the SPACE command in ARRAY DATA) to contain N+1 values, where the i^{th} temperature value will correspond to the i^{th} node in the input list, and the final value will contain the single effective sink temperature for the set of nodes.
4. If multiple calls are made using the array mode, the data in the arrays will be overwritten. Unique arrays should be used in each call if this is not desirable.

The above election (array mode) is to be contrasted with *the file mode*:

1. GUA must be an integer unit number referring to a file opened previously via a USRFIL call (if not NUSER1 or NUSER2). TSINK will fill this file with a CONDUCTOR DATA block for submodel "TSINK" with a conductor generated for each input node to the sink (whether to unique sinks or a to a single sink temperature numbered 999999).
2. If *mode* contains '1' (a single sink temperature is desired) then TUA must be a real variable which will receive the single effective sink temperature for the set of nodes.
3. Otherwise, if *mode* does *not* contain '1' (and therefore multiple sink temperatures are desired), TUA must also be a *different* integer unit number referring to a file opened previously via a USRFIL call (if not NUSER1 or NUSER2). TSINK will fill this file with a NODE DATA block for submodel "TSINK" with a boundary node* generated for every sink required: one if mode contains a '1' and N+1 sinks otherwise (with the last one being identified as 999999 to contain the single effective sink). This block can be used with the CONDUCTOR DATA block in future runs to replace the system level model (i.e., all but the included nodes).

   A short commented-out CONTROL DATA block is also generated for reference showing the units used.

* Actually, a slightly more general-purpose "heater node" is used instead.

4. If multiple calls are made using the file mode, the new data will be appended at the end of the file. The user may wish to mark distinctive calls by writing a comment to each file between TSINK calls. For example, if NTEST is a unit number for one of the files:

```
      WRITE(NTEST,99) TIMEN
99    FORMAT('C    THE FOLLOWING WAS MADE AT TIME = ',1pG13.5)
```

Guidance: The results of TSINK represent a single "snap shot" at one time point or at a steady state representing an average. In a transient with time-varying heat loads and/or articulating infrared RADKs, a series of sink temperatures would ideally be needed that vary with time. Although planned for the future, the current version of TSINK offers no built-in capability for generating time-varying sink temperatures and their associated (perhaps time-varying) conductors. Work-arounds include using TSINK1 or the array option in TSINK to create and save multiple sets of conductor and sink data (each time point saved in a different array), and then generating TVS conductors plus an input file for the BNDDRV routine for the boundary nodes representing the sinks. It would be more difficult to adapt the file mode of TSINK towards the goal of producing time-varying sinks. In the file mode, for example, each set (time point) could be assigned a different submodel name which is built in series, but this would cause a step function in the reduced model as each boundary condition is swapped.

Examples:

```
C "Traditional" radiative sink, with Q included (Qm=1.0) and linear ignored.
C All nodes in BATTERY are included in the set, and are therefore part
C of the keeplist.
C ARRAY VERSION:
      CALL TSINK('battery', 0, 'none', 'R', 1.0, sinks.A100, sinks.A101)
C Assuming there are 504 diffusion and arithmetic nodes in submodel
C Battery (use SIZETAB if unsure), then the following might have been
C used:
HEADER ARRAY DATA,SINKS
      100   = space,504       $ will hold sink RADKs
      101   = space,505       $ will hold 504 sinks plus single effective


C FILE VERSION: The above, creating files instead:
      CALL TSINK('battery', 0, 'none', 'R', 1.0, NTEST, MTEST)
C where somewhere before (in OPERATIONS usually), the files were opened,
C and NTEST and MTEST were not used elsewhere (an integer register could
C have been assigned instead as a unit number):
      CALL USRFIL(NTEST,'gsink.inc','unknown')
      CALL USRFIL(MTEST,'tsink.inc','unknown')
```

```
C A linear sink, with Q excluded (Qm=0.0) and other parts of the same
C component not included in the calculation (they will be retained).
C A single sink temperature is returned (not normally recommended!)
C otherwise the linear conductances are stored Array-style
      CALL TSINK('batt1', batt1.NA201, 'batt1,batt2,batt3', 'L1', 0.,
     +    sinks.A100, TTEST)
C the list of nodes in array data:
HEADER ARRAY DATA, BATT1
      201 = 1,2,304,305,3,4, 4001, 4002, 4003, 4004
C and the space to hold the conductors
HEADER ARRAY DATA, SINKS
      100 = space,10
```

7.4.21 Submodel-level Map

Subroutine Name: SUBMAP

SUBMAP is an output routine that produces connectivity and heat flow information at the submodel level for all currently active thermal and fluid submodels.

Calling Sequence:

```
CALL SUBMAP
```

The output for SUBMAP includes various terms that may require clarifications. See Table 7-1 for an example of SUBMAP outputs.

Each submodel is mapped, with summary information for that submodel followed by a table of submodels to which it connects. Heat rates ("HEAT RATE TO") are defined as positive *from* the current mapped model *into* the submodel on each line of the table.

"HEAT RATE IMPOSED" includes nodal source terms (Q) for thermal submodels and lump source terms (QL) for fluid submodels (but excluding make-up heat for heater junctions).

"HEAT TO INTL BDYS." for thermal submodels includes all heat flowing into boundary and heater (and held) nodes within that submodel, whether or not the node exists in the current submodel or some other submodel. In general, this number plus the imposed heat rate plus the total heat rate (for all attached submodels) in is approximately zero at steady-state. Similarly, for fluid submodels this "HEAT TO INTL BDYS." term includes all heat energy flowing into plena and held lumps.

"Average temperatures" for a thermal submodel include diffusion and arithmetic nodes but excludes boundary and heater nodes as well as any held (via HTRNOD or HTRMOD) nodes. Similarly, for a fluid submodel this term includes tanks and junctions but not plena nor any held (via HLDLMP) lumps. ("Heater junctions" created using HTRLMP, however, are included in the average.)

"TOTAL CAPACITANCE" is the sum of all C values for nonheld diffusion nodes within the submodel. This value, divided by the sum of all "TOTAL COND." values (provided in the last "TOTALS" line), is a rough indication of the time constant of the submodel as a unit.

"LIN/TIE COND." refers to the sum of linear conductances (G) or tie conductances (UA) from all nodes in the currently mapped ("from") submodel to all nodes or lumps in the listed ("to") submodel. The adjacent heat rate (just to the right) corresponds to the heat transfer through these linear elements. For a fluid submodel, the equivalent term is "TIE COND." and it also represents the total.

"RAD. COND." refers to the sum of radiation conductances (G, in units of power per degree^4) from all nodes in the currently mapped ("from") submodel to all nodes or lumps in the listed ("to") submodel. The adjacent heat rate (just to the right) corresponds to the heat transfer through these nonlinear elements.

"TOTAL COND." refers to the sum of all conductances, including linearized radiation conductances. This number may be used comparatively to determine how strongly interconnected the submodels are. The adjacent heat rate (just to the right) corresponds to the heat transfer through all conductors and ties, and is the sum of the linear and nonlinear heat flows.

"TOTAL TANK/JUNC VOLUMES" is the sum of all VOL terms in the model, including those in junctions (which are otherwise ignored by the solution routines) but excluding volumes of held (HLDLMP) lumps. In many models, this value can be compared against "TOTAL TUBE/STUBE VOLS." which is the sum of the AF*TLEN product for all tubes and STUBE connectors within the fluid submodel as a quick check.

Caution: One-way conductors and path/iface duplication factors may cause distortions. SUBMAP works from the perspective of each currently mapped submodel. While it does include tie duplication factors, it does not account for magnification effects resulting from path or iface duplication factors. Consider SUMDFLO as an alternative for certain submodel-level information such as volume or mass.

Table 7-1 Excerpt Results from a Sample SUBMAP Call

A SUBMAP OF THERMAL SUBMODEL MLI

AVERAGE DIFF/ARIT TEMP.	= 532.864	(DEG)
TOTAL CAPACITANCE	= 58.4280	(ENERGY/DEG)
TOTAL HEAT RATE IMPOSED	= 0.00000	(ENERGY/TIME)
TOTAL HEAT TO INTL BDYS.	= -48.1718	(ENERGY/TIME)

SUBMODEL	TYPE	AVG TEMP.	LIN/TIE COND	HEAT RATE TO	RAD. COND.	HEAT RATE TO	TOTAL COND.	TOTAL RATE
TANK	THERMAL	42.614	1.24000E-02	6.1668	0.0000	0.0000	1.24000E-02	6.1668
VCS	THERMAL	312.712	1.86000E-04	3.92010E-02	0.35186	41.966	0.20110	42.005
	TOTALS		1.25860E-02	6.2060	0.35186	41.966	0.21350	48.172

A SUBMAP OF HYDRO SUBMODEL FLUID

AVERAGE TANK/JUNC TEMP.	= 42.600	(DEG)
TOTAL TANK/JUNC VOLUMES	= 2.812118E-02	(LENGTH CUBED)
TOTAL TUBE/STUBE VOLS.	= 2.988833E-02	(LENGTH CUBED)
TOTAL HEAT RATE IMPOSED	= 0.00000	(ENERGY/TIME)
TOTAL HEAT TO INTL BDYS.	= 26481.6	(ENERGY/TIME)

SUBMODEL	TYPE	AVG TEMP.	TIE COND.	HEAT RATE TO
IHX	THERMAL	42.379	281.66	-3.24494E-02
TANK	THERMAL	42.614	769.69	-14.216
VCS	THERMAL	312.712	74.267	-34.837
	TOTALS		1125.6	-49.085

7.5 Input Subroutines

7.5.1 Resetting Some or all Nodes' Temperature in a Submodel

Subroutine Name: SETTMOD

Description: This routine sets the temperature of some or all nodes within a thermal submodel to a given value. If a subset of the submodel's nodes are chosen to be changed, they are selected by type: all diffusion and/or arithmetic and/or boundary/heater nodes in a submodel.

Guidance: Although the calling arguments for SETTMOD are similar to those of BDYMOD and HTRMOD, note that 'ALL' means 'AD' for BDYMOD/HTRMOD, but it means 'ADB' for SETTMOD. Also note that arithmetic and diffusion nodes held by BDYMOD or HTRMOD are still considered to be their original type in SETTMOD: the inclusion of the letter B in the EXTENT argument will not change their temperature.

Calling Sequence:

```
CALL SETTMOD(MODNAM, EXTENT, TEMP)
```

where

MODNAM. . . . Thermal submodel name, in single quotes
EXTENT . . . . . 'A' to change the temp. of all arithmetic nodes in the submodel MODNAM
'D' to change the temp. of all diffusion nodes in the submodel MODNAM
'B' to change the temp. of all boundary and heater nodes in submodel MODNAM
'DAB', 'ADB', etc. or 'ALL' to change the temp. of all nodes in the submodel MODNAM
TEMP. . . . . Temperature to which the nodes will be set, in user units (R, K, C, or F)

Example:

```
CALL SETTMOD('THISMOD', 'AD', temp_new)
CALL SETTMOD('THATMOD', 'ALL', 20.0)
```

7.5.2 Boundary Node Temperature Driving

Subroutine Name: BNDDRV

Description: Subroutine BNDDRV drives boundary temperatures. This routine allows the use of temperature history file (generally a FILOUT file from subroutine BNDGET, Section 7.4.12) to provide boundary node temperature histories during execution. The history file may be externally generated if it meets the following requirements (written to an 8G25.16E3 format).

```
TIME,(TEMP (I), I=1,N)
```

where:

`TIME` ..... time point
`TEMP (I)`... temperature of I^{th} boundary node at TIME
`N` ..... number of boundary nodes to be driven

This routine requires an array of the actual numbers of the boundary nodes which are to be driven. The input file is searched and linear interpolation is performed to find the correct temperature for each boundary node in the specified input at TIMEM.

Restrictions and Guidance: BNDDRV should be called from VARIABLES 0. Alternately the user may provide CARRAY data for each model and have a call in each VARIABLES 0 block.

Node numbers are input via a CARRAY DATA block. The SMN argument specifies which submodel contains the CARRAY DATA block that contains the node numbers. NCA1 is the actual number of the first CARRAY to use. NUMCAS is the number of *consecutive* CARRAYs to use if all the instructions do not fit in a single line. The numeric identifiers of the additional CARRAYs are irrelevant, but they must be input in the order in which they are to be processed, as if they were "continuations" of the first starting CARRAY. Use ALL instead of a node number list to check all of the nodes in any one submodel.

CARRAY Format:

```
HEADER CARRAY DATA, SMN
     nca1 = smn1, n1, n2 ..., smn2, n1, n2,
     nca2 = n3, n4, ..., smn3, ALL
```

Calling Sequence:

```
CALL BNDDRV (NUMCAS, SMN, NCA1, FILIN)
```

where:

NUMCAS . . . . . number of CARRAYS to use (integer)
SMN . . . . . Submodel name where CARRAY data appears (character)
NCA1. . . . . Actual Number of 1st CARRAY to use (integer)
FILIN . . . . . A file name for input, usually created by BNDGET (character)

Example: The following shows the relevant data blocks and their use with BNDGET:

```
HEADER CARRAY DATA, MAIN
         1 = TEST
         2 = MOD1, 100, 200, 300, MOD2, 100, 200, 300
         3 = 400, 500, MOD3, 100
         4 = TEST
HEADER VARIABLES 0, MAIN
              .
              .
              .
         CALL BNDDRV (2, 'MAIN', 2,'MODELS.BND')
              .
              .
```

7.5.3 Re-initialization for Parametric Cases

Subroutine Name: RESPAR

Description: (See Section 4.7) This subroutine reads in all program variables (temperature, capacitances, conductances, Q-sources, control constants, user constants, user arrays, and all flow data) from an internal program (temporary) folder.

Restrictions and Guidance: Parameters read in are accessed by record number, (Reference subroutines SAVPAR and SVPART, Section 7.4.6). Called generally from OPERATIONS or PROCEDURE prior to a solution routine. With subroutine STDSTL, conditional calls from VARIABLES 0 are appropriate. This requires logic making calls conditional on pseudo-time, TIMEN. Each call to SAVPAR or to SVPART will be associated with a record number to be used with subroutine RESPAR.

Caution (SVPART restorations only): If registers and expressions were saved independently of model data (nodal temperatures, hydraulic diameters, etc.), or if model data were saved and registers were excluded, then the restored state may contain inconsistencies (that are assumed purposeful) between register values and the data to which they would otherwise correspond. Furthermore, subsequent calls to the UPREG family of routines (Section 7.12.1) will have no effect unless the registers *themselves* have changed *after* the call to RESTAR. (See also Section 4.9.6.3 and Section 4.9.7.)

Guidance: Values of registers may be overwritten when calling RESPAR. This routine therefore autodetects when it is called from within solutions such as the Solver, reliability engineering modules, and parametric sweeps. In this case, it returns to the original value of the key registers after the restore operation is complete, and propagates these values before returning.

Calling Sequence:

```
CALL RESPAR (NREC)
```

where NREC is the record number associated with the set of parameters to be read in. NREC is an integer variable name or literal.

7.5.4 Reading In Parameters from the Restart Folder

NOTE: Binary input and output are undergoing a transition from an older and soon-to-be-obsolete single "SAVE file" format to a newer format: a set of files stored within a compressed subdirectory, or "CSR folder." The term file *in phrases such as "SAVE file" or "RSO file" or "RSI file" can refer to a* CSR folder *too. In other words, several different CSR folders may be named in OPTIONS for the purposes of output (SAVE, RESAVE, SAVEDB, CRASH) or restarting (RSO, RSI). We apologize for any confusion during this transition.*

RESTAR can auto-determine whether its data source is a file or folder. Therefore, in the following sections the term "folder" will be used exclusively even though users of the old format will be applying it to a file instead.

Subroutine Name: RESTAR

Description: (See Section 4.7) This subroutine reads in all program variables (temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays and flow variables) from the binary RSI (restart input) CSR folder. Each call to RESAVE will print an associated record number to be used with subroutine RESTAR. Equivalently, the first call to CRASH will print the associated record number to be used in the call to RESTAR.

The exact composition of the data read in depends on argument 'ARGS' used with subroutine RESAVE (or SAVE) when the restart data was written to the RSO folder. If being used to access a crash folder, the folder contains the information equivalent to a call to RESAVE with the argument 'ALL'.

Restrictions and Guidance: Calls to RESTAR require that the RSI be identified with a user's CSR folder name in the OPTIONS DATA block. If any network elements, constants, arrays or array elements have been added or deleted, a collision will be detected and a restart cannot be performed. In this case, an error message will be printed and the program will abort.

Caution: Calls to RESTAR do not affect the dormant/awake status of thermal submodels, and do not affect the location of ties. In other words, the effects of DRPMOD and PUTTIE calls in previous runs are irrelevant. Also, note that the collision checks cannot detect changes in input order, which should be avoided unless purposefully employed by an advanced user.

Caution: If registers and expressions were saved independently of model data (nodal temperatures, hydraulic diameters, etc.), or if model data were saved and registers were excluded, then the restored state may contain inconsistencies (that are assumed purposeful) between register values and the data to which they would otherwise correspond. Furthermore, subsequent calls to the UPREG family of routines (Section 7.12.1) will have no effect unless the registers *themselves* have changed *after* the call to RESTAR. (See also Section 4.9.6.3 and Section 4.9.7.)

Guidance: Values of registers may be overwritten when calling RESTAR. This routine therefore autodetects when it is called from within solutions such as the Solver, reliability engineering modules, and parametric sweeps. In this case, it returns to the original value of the key registers after the restore operation is complete, and propagates these values before returning.

Calling Sequence:

```
CALL RESTAR (NREC)
```

where NREC is the restart record number associated with the data to be read in.

Caution: Although used analogously, exact NREC values in SAVE files do not correspond to NREC values in CSR folders. In other words, if a prior model used a SAVE file to restart, and a new run uses a CSR folder instead, any symbol, register, or logic that references the old NREC value will need to be updated.

Subroutine Name: RESTNC

Description: (See Section 4.7) This subroutine is identical to RESTAR, except that collision checks are not performed. This routine is intended for advanced manipulations by users sufficiently versed in the code to assume responsibility for collision checks. ***Usage of this routine is dangerous and should only be used as a last resort.***

Caution: Calls to RESTNC do not affect the dormant/awake status of thermal submodels, and do not affect the location of ties. In other words, the effects of DRPMOD and PUTTIE calls made in previous runs are irrelevant.

7.6 Utility Subroutines

7.6.1 Finding User Array Locations

Subroutine Name: ARYTRN

Description: This routine returns the absolute location in the A-array of a user array, given the actual number of the array. For numeric arrays, the integer count location is returned. For character arrays, use CRYTRN (Section 7.6.3) instead.

Restrictions and guidance: This routine bypasses the need to consult the array data storage dictionary when relative locations are required for F-type (Fortran) statement logic.

Calling Sequence:

```
CALL ARYTRN('SMN',NUM,L)
```

where:

SMN . . . . . a submodel name enclosed in character delimiters (’)
NUM . . . . . an actual user array number (negative NUM signals to return -1 instead of abort if array is not found)
L. . . . . the returned relative location of the array (integer)

Example:

```
      CALL ARYTRN(’hotstuff’, 10, itest)
F     A(itest+3) = 4.5  $ equiv to “A(10+3) = 4.5”: third element in A10
```

7.6.2 Finding Conductor Value Locations

Subroutine Name: CONTRN, INTCON

Description: This routine returns the relative location in the G-array of a network conductor, given the actual conductor number.

Restrictions & Guidance: This routine bypasses the need to consult the conductor data storage dictionary when relative locations are required for F-type (Fortran) statement logic.

Calling Sequence:

```
CALL CONTRN (’SMN’,NUM,L)
```

where:

SMN . . . . . a submodel name enclosed in character delimiters (’)
NUM . . . . . an actual user conductor number (negative NUM signals to return -1 instead of abort if conductor is not found)
L. . . . . the returned relative location of the conductor (integer)

May alternatively be called as a function:

```
L = INTCON(’SMN’, NUM)
```

7.6.3 Finding CARRAY (Character Array) Locations

Subroutine Name: CRYTRN

Description: This routine returns the index in the UCA array given the actual number of a CARRAY.

Restrictions and Guidance: This routine bypasses the need to consult the CARRAY data storage dictionary when absolute indexes are needed for F-type (FORTRAN) statement logic.

Calling Sequence:

```
CALL CRYTRN ('SMN',NUM,INDEX)
```

where:

SMN	a submodel name enclosed in character delimiters (')
NUM	an actual user CARRAY number (negative NUM signals to return -1 instead of abort if CARRAY is not found)
INDEX.....	the returned integer pointer into the UCA array (integer)

7.6.4 Finding Submodel Name Locations

Subroutine Name: MODTRN, MDLTRN

Description: This routine returns the relative location of a thermal submodel name in the array of submodel names (as defined by the BUILD records).

Restrictions and Guidance: This routine bypasses the need to consult the submodel name data storage dictionary when submodel sequence numbers are needed for F-type (Fortran) statement logic.

Calling Sequence:

```
CALL MODTRN('SMN',L)
```

where:

SMN	a thermal submodel name enclosed in character delimiters (')
L	the returned submodel sequence number (integer)

May alternatively be called as a function (with a different name):

```
L = MDLTRN('SMN')
```

7.6.5 Finding Node Attribute Locations

Subroutine Name: NODTRN, INTNOD

Description: This routine returns the relative location in the T,C and Q arrays of nodal temperatures, capacitor and Q-values, given an actual node number.

Restrictions and Guidance: This routine bypasses the need to consult the node data storage dictionary when relative locations are required for F-type (Fortran) statement logic.

Calling Sequence:

```
CALL NODTRN('SMN', NUM,L)
```

where:

SMN . . . . . a thermal submodel name enclosed in character delimiters (')
NUM . . . . . an actual user node number (negative NUM signals to return -1 instead of abort if node is not found)
L. . . . . the returned relative location of the node in arrays T, C, and Q (integer)

May alternatively be called as a function (with a different name):

```
L = INTNOD('SMN', NUM)
```

7.6.6 Finding User Constant Locations

Subroutine Name: NUMTRN

Description: This routine returns the relative location in the K (and equivalenced XK) array, of a numbered user constant, given an actual constant number.

Restrictions and Guidance: This routine bypasses the need to consult the user constant dictionary when relative locations are required for F-type (Fortran) statement logic.

Calling Sequence:

```
CALL NUMTRN('SMN',NUM,L))
```

where:

SMN . . . . . a submodel name enclosed in character delimiters (')
NUM . . . . . an actual user constant number (negative NUM signals to return -1 instead of abort if node is not found)
L . . . . . the returned relative location of the constant (integer)

7.6.7 Area Integration

Subroutine Name: TRPZDA

Description: This subroutine performs area integration by the trapezoidal rule. It may be used whether or not the DX increment is uniform, but a Y value corresponding to each X value is required. The operation performed is as follows:

```
A = 1/2(Xi - Xi-1)*(Yi + Yi-1)    ,   i = 2,N
```

All values must be floating point numbers except the array length N which must an integer.

Calling Sequence:

```
CALL TRPZDA(N,X(DV),Y(DV),A)
```

7.6.8 Array Integration

Subroutine Name: CRVINT

Description: This subroutine performs an integration of the doublet array, A, and stores the results in doublet array B. The independent variables of the A array are transferred directly to the B array. The dependent variables of the B array are calculated by:

B(2) = 0.0
B(2*N) = B(2*(N-1)) + 0.5*[A(2*N)+A(2*(N-1)]
* [A(2*N-1)-A(2*(N-1)-1)]N=2, NP

where NP = number of pairs of points in the A array (half the integer count)

This subroutine was written primarily for integration of specific heat arrays to obtain enthalpy arrays but could be used for integration of any dependent variable.

Restrictions: Space in B array must be exactly equal to the space in the A array. There must be at least two points in A array (i.e., the integer count must be at least 4).

Calling Sequence:

```
M      CALL CRVINT(A(IC),B(IC))
```

7.6.9 Least Squares Curve Fitting

Subroutine Name: LSTSQU

Description: This subroutine performs a least-squares curve fit to an arbitrary number of X,Y pairs, yielding a polynomial equation of up to order 10. Rather than using a matrix inversion, this subroutine calls the subroutine SIMEQN to obtain a simultaneous solution.

This subroutine is independent of the curve fitting features available in the Solver (Section 5.10).

Restrictions: All values must be floating point numbers except N and M which must be integers. N is the order of the polynomial desired and is one less than the number of coefficients desired. M is the array length of the independent X or dependent Y values.

Calling Sequence:

```
CALL LSTSQU(N,M,X(DV),Y(DV),A(DV))
```

7.6.10 Putting a Submodel in a Boundary State

Subroutine Name: DRPMOD

Description: This subroutine drops a *thermal* submodel out of the current pseudo-compute sequence. Conductors will still be attached to these nodes but their temperatures will no longer be changed. Therefore they will act as boundary nodes until they are replaced by subroutine ADDMOD. In such a dormant state, *calls to a submodel's logic blocks (VARIABLES 0, VARIABLES 1, VARIABLES 2, OUTPUT CALLS) are also suspended. This means updates to parameters owned by other submodels (that are probably not inactive) should be avoided.*

DRPMOD cannot be used to drop fluid submodels.

Restrictions and Guidance: This routine may called *once* for each *active* thermal submodel to be dropped. *It should only be within OPERATIONS or PROCEDURE.* Calls to DRPMOD in any VARIABLES, OUTPUT CALLS, or FLOGIC block are not appropriate. Calls to RESTAR or RESTNC do not affect prior calls to DRPMOD—the dormant/awake status of previous runs is ignored.

Calling Sequence:

```
CALL DRPMOD ('SMN')
```

where SMN is a submodel name (character delineator ’ is required)

7.6.11 Reactivating a Boundary State Submodel

Subroutine Name: ADDMOD

Description: This subroutine replaces a *thermal* submodel in the pseudo compute sequence. ADDMOD cannot be used with fluid submodels.

Restrictions and Guidance: This routine can be called *once* for each thermal submodel *that been dropped*, returning it to active status. *It should only be within OPERATIONS or PROCEDURE.* Calls to ADDMOD in any VARIABLES, OUTPUT CALLS, or FLOGIC block are not appropriate. Calls to RESTAR or RESTNC do not affect prior calls to ADDMOD—the dormant/awake status of previous runs is ignored.

Calling Sequence:

```
CALL ADDMOD ('SMN')
```

where SMN is a submodel name (character delineator ’ is required)

7.6.12 Calculating Transient Energy Balance

Subroutine Name: COMBAL

Description: This subroutine calculates the energy imbalance for transient routines. It takes into account capacitance terms (C*dT/dt), and so can be used to measure the total energy imbalance in the network, which is a measure of solution accuracy. This routine sets the values of EBALSC and EBALNC* for all currently active submodels *based on the last time step taken*, and prints a brief report to the OUTPUT file. (Refer to Section 4.4.1 for a description of these error tolerance parameters.)

The explicit transient routine, FORWRD, does not perform any energy balance checks. Rather, the time step is predicted purely on the basis of stability (and not accuracy) criteria. The system- and node-level energy balance criteria, EBALSA and EBALNA, are not applicable and therefore the corresponding output variables EBALSC and EBALNC are not updated when that routine is used. COMBAL therefore provides the user with the ability to force the calculation of those terms such that an assessment of the current error can be made in energy terms.

* These are the calculated values that are compared (in steady states) with EBALSA and EBALNA, respectively, as described in Section 4.3.2 and Section 4.4.1.

Unlike FORWRD, the preferred transient routine TRANSIENT (“FWDBCK”) is implicit and *does* base its time step predictions on accuracy. However, historically it did not consider energy balance as an accuracy criteria and therefore COMBAL was a useful adjunct to that routine. In modern versions, the control constant FBEBALA (if nonzero and therefore active) does invoke an approximate energy imbalance check, updating EBALNC (but not EBALSA) in the process. Normally, COMBAL should be considered to be a required check when FBEBALA=0.0 and optional when FBEBALA is active (i.e., nonzero). However, the default level of energy control invoked by FBEBALA is modest (10% error in maximum nodal imbalance) and approximate, so COMBAL can serve a useful purpose still since it invokes a full (nonapproximate) calculation and updates EBALSC as well.

Guidance: COMBAL is best invoked from OUTPUT CALLS during a TRANSIENT solution (in which case NSOL=2) or a FORWRD transient (NSOL=3). (See also Section 4.2.1 and Section 4.2.2.) Because of the output it generates, if called from VARIABLES 2 some measure should be taken to limit the number of times it is invoked.

Caution: This routine overwrites the value of the nodal energy imbalance, EBALNC, that is updated in TRANSIENT (“FWDBCK”) when FBEBALA is nonzero. The value produced by TRANSIENT is an estimate, whereas the value produced by COMBAL is a full calculation.

Calling Sequence:

```
CALL COMBAL
```

7.6.13 Net Nodal Heat input

Subroutine Name: NODBAL

Description: This subroutine calculates the current net energy flow into a node, taking into account connections (ties, conductors) and sources. For arithmetic nodes and for diffusion nodes in steady state solutions, the resulting number should approach zero and so is an indication of convergence. For diffusion nodes in transients, the result divided by the capacitance C is the time rate of change of the node (dT/dt = EBAL/C, where EBAL is the returned result of NODBAL). Note that the same value is also available via the NODTAB output routine for many nodes at once.) for transient routines.

Guidance: This routine is best called within OUTPUT CALLS or VARIABLES 2; not all updates to sources, conductors, and ties will have been performed if it called within VARIABLES 0 or VARIABLES 1.

Restrictions: Input errors result in zero value returned along with printed warnings.

Calling Sequence:

```
CALL NODBAL('smn',nid,ebal)
```

where:

`smn`..... thermal submodel name in single quotes
`nid`..... node identifier
ebal ..... returned real value containing the net energy into the node.

Example:

```
CALL NODBAL('grand', 739, etest)
```

7.6.14 Accessing Variables as Double Precision

Subroutine Name: DPTIME

Description: This routine is no longer required. It was used to return time in double precision. Use the variables TIMEO, TIMEM, or TIMEN directly instead. It is documented only to provide explanation to users who inherit an older model.

Calling Sequence:

```
CALL DPTIME('time_name','op_name',val)
```

where:

`time_name` . TIMEO, TIMEM, or TIMEN in single quotes
`op_name`.... PUT or GET in single quotes
`val`..... double precision value or variable to put (set), or location into which double precision variable should be fetched

The first argument, `time_name`, identifies the variable to be fetched or reset in double precision mode. Valid inputs are 'TIMEN', 'TIMEO', or 'TIMEM' in single quotes. The second argument, `op_name`, tells the program whether the user wishes to fetch or reset the time variable named by the first argument. Valid inputs are either 'PUT' or 'GET' in single quotes. The final argument is either the double precision value to set (if `op_name` is 'PUT') or a double precision variable name (if `op_name` is 'GET').

Example: Fetch TIMEN and store the results as a register named DPT

```
CALL DPTIME('TIMEN','GET',DPT)
```

Subroutine Name: DPTEMP

Description: This routine is no longer required. It was used to return time in double precision. Use the variable T directly instead. It is documented only to provide explanation to users who inherit an older model.

Calling Sequence:

```
CALL DPTEMP('smn',nid,'op_name',val)
```

where:

`smn` .....thermal submodel name in single quotes
`nid` .....node identifier
`op_name` ....PUT or GET in single quotes
`val` .....double precision value or variable to put (set), or location into which double precision variable should be fetched

The first argument and second arguments are the thermal submodel name (in single quotes) and the user ID for the node. The third argument, `op_name`, tells the program whether the user wishes to fetch or to reset the temperature of the node named by the first two arguments. Valid inputs are either 'PUT' or 'GET' in single quotes. The final argument is either the double precision value to set (if `op_name` is 'PUT') or a double precision variable name (if `op_name` is 'GET').

Example: Calculate the temperature difference between two nodes in double precision. DPT1 and DPT2 are double precision registers.

```
CALL DPTEMP('WATERY',100,'GET',DPT1)
CALL DPTEMP('WATERY',200,'GET',DPT2)
DTEST_DP = DPT1-DPT2
```

7.6.15 Toggling Double Precision Options On and Off

These routines are no longer required, and they perform no function if called: they are ignored. They are documented only to provide explanation to users who inherit an older model.

Subroutine Name: STOPDP

Description: This routine suspends double precision calculations, if they are active.

Calling Sequence:

```
CALL STOPDP
```

Subroutine Name: RESTDP

Description: This routine reactivates double precision calculations that have been previously suspended via STOPDP.

Calling Sequence:

```
CALL RESTDP
```

7.6.16 Heat Flows and Other Data about a Conductor

For historical reasons, there is no way in SINDA/FLUINT to directly reference the end-point nodes on a conductor, nor to reference the heat rate through a conductor. This routine overcomes these limitations by providing access to that data.

Subroutine Name: CONDAT

Description: This routine returns information about a single conductor including pointers (internal sequence numbers) to the end-point nodes along with the heat rate through the conductor.

Guidance: The returned node pointers are the same values that would be returned by NODTRN or INTNOD. Therefore these routines can be used to tell the nodes apart since the pointers are otherwise not necessarily returned in the same order in which they were input in CONDUCTOR DATA. The returned net heat through the conductor is positive from the first node to the second node, again as returned by CONDAT but not necessarily as input in CONDUCTOR DATA.

Restriction: This routine will fail and cause a processor abort if the referenced conductor is not found, ***or if it and both end-point nodes are not active in the current BUILD configuration.***

Restriction: If the referenced conductor connected more than one node pair (an obsolete, undocumented, and unsupported feature), only one such pair will be returned, and this pair may not correspond to the first pair input in CONDUCTOR DATA.

Calling Sequence:

```
CALL CONDAT(MODNAM, CONNUM, NODE1, NODE2, QTHRU)
```

where:

`MODNAM`	Thermal submodel name containing conductor CONNUM, in single quotes
`CONNUM`	User integer identifier for conductor
`NODE1`.....	Returned *internal* sequence number (pointer) of one end-point node (not necessarily the first node as input in CONDUCTOR DATA)
`NODE2`.....	Returned *internal* sequence number (pointer) of another end-point node
`QTHRU`.....	Returned heat rate through the conductor, positive from NODE1 to NODE2 (real)

Example: Place the heat rate through conductor 10 in submodel MOOKIE in QTEST: Check to see if the first returned node (placed in NTEST) is the first input node (node 405 in the same submodel), and rectify the heat rate if not:

```
CALL CONDAT('MOOKIE', 10, NTEST, MTEST, QTEST)
IF(NTEST .NE. INTNOD('MOOKIE',405)) QTEST = -QTEST
```

Integer Function Name: GETGNUM

Description: This function returns the user-provided "name" or integer ID for a conductor, given its internal location. In many ways, this is the inverse function of INTCON.

Calling Sequence:

```
ITEST = GETGNUM(IDG)
```

where:

IDG.....Input internal sequence location for a conductor (integer)
ITEST .....Returned user ID for the conductor (integer)

Character*32 Function Name: GETGMOD

Description: This function returns the submodel containing a conductor, given its internal location.

Calling Sequence:

```
UCA11 = GETGMOD(IDG)
```

where:

IDG.....Input internal sequence location for a conductor (integer)
UCA11 .....Returned character string containing the conductor's submodel's name (Character*32)

Example:

```
      DO ITEST = 1, NGTOT
            WRITE(NUSER1,19) GETGMOD(ITEST),GETGNUM(ITEST),ITEST
      ENDDO
19    FORMAT(' Conductor ',A32,'.',I8,
     +     ' is stored in the G array in cell ',I8)
```

7.6.17 Heat Flows Between Nodes or Groups of Nodes

QFLOW and its auxiliary routine QFLOWSET represent a flexible means of tracking heat flowing from one or more nodes and into one or more nodes.

Subroutine Name: QFLOW

Description: This subroutine tallies the current heat flow from one set of nodes to another set. A "set" can consist of:

1. a single node
2. a subset of a submodel (i.e., a list of nodes within that submodel)
3. all the nodes in a submodel
4. all active nodes

The routine returns the heat flowing between the sets of nodes via radiation conductors, linear conductors, and both (the total).

Searching a list of nodes and testing for links to other nodes would be computationally slow if QFLOW is called repeatedly (i.e., in a VARIABLES or OUTPUT BLOCK) and if it had no memory of previous calls. Therefore, QFLOW actually consists of two routines: one to create and store sets of nodes (QFLOWSET) and one to calculate the current heat flow for a given numbered set (QFLOW). The user identifies each set with a numeric identifier: a "set ID." Up to 10,000 sets of nodes can be created and stored in any run, and repeated calls to QFLOWSET may be used to redefine a set of nodes for a particular set ID.

QFLOWSET must be called first to define the nodes involved in the set, and to assign this set an integer identifier that can be used to refer to it in later calls to QFLOW. QFLOWSET can be called anywhere, but will most often be called in OPERATIONS following BUILD statements but preceding solution routines or Solver or reliability estimation calls. QFLOW cannot be called without at least one prior call to QFLOWSET.

Calling Sequence:

```
CALL QFLOWSET(setid, smn1, list1, smn2, list2)
CALL QFLOW(setid, qlin, qrad, qtot)
```

where:

`setid`..... user-selected integer identifier of a given set. Must be between 1 and 10000, inclusive.

`smn1`..... the submodel from which heat flows will be tallied, in single quotes. smn1 must be an active (built) submodel. 'ALL' ***cannot*** be used to define smn1.

`list1` .....one of the following:
m .....an integer identifier of a single node m in submodel smn1
0.....(zero) signaling all nodes in submodel smn1
[smn.NAn].a reference to a SINDA array containing an *integer* list of nodes contained within submodel smn1

`smn2`.....the second submodel to which heat flows will be tallied, in single quotes. smn2 must be an active (built) submodel. 'ALL' ***can*** be used to define smn2, meaning all active submodels, in which case the "list2" argument is ignored.

`list2` .....one of the following:
m .....an integer identifier of a single node m in submodel smn2
0.....(zero) signaling all nodes in submodel smn2
[smn.NAn].a reference to a SINDA array containing an *integer* list of nodes contained within submodel smn2

`qlin`.....the returned heat transfer from list 1 to list 2 via linear conductances

`qrad`.....the returned heat transfer from list 1 to list 2 via radiation conductances

`qtot`.....the returned total heat transfer from list 1 to list 2 (qtot = qlin + qrad). Note: heat flow is positive from list 1 to list 2: qtot will be positive if list 1 is losing heat to list 2, and negative if it is gaining heat from list 2.

Examples:

```
c The following example tallies the heat transfer from nodes 101, 110,
c and 121 in submodel ROD to all of the nodes in submodel CONE
c
HEADER OPERATIONS
BUILD CON, ROD, CONE
      CALL QFLOWSET(143,'ROD',ROD.NA43,'CONE',0)
      CALL TRANSIENT
HEADER VARIABLES 2, CONE
      CALL QFLOW(143, CTEST, RTEST, QTEST)
      WRITE(NUSER1,*) ' LINEAR TRANSFER = ', CTEST
      WRITE(NUSER1,*) ' RADIATION TRANSFER = ', RTEST
      WRITE(NUSER1,*) ' TOTAL HEAT TRANSFER = ', QTEST
HEADER ARRAY, ROD
      43 = 101, 110, 121
c
c The following is an example of a series of calls to QFLOWSET and QFLOW to
c tally heat transfer between nodes 101 and 110 and all nodes in
c submodel CONE independently, rather than as a set (per the above example)
c
      CALL QFLOWSET(1,'ROD',101,'CONE',0)
      CALL QFLOW(1, CTEST, RTEST, QTEST)
      ...
      CALL QFLOWSET(1,'ROD',110,'CONE',0)
      CALL QFLOW(1, CTEST, RTEST, QTEST)
```

```
c
c The second call to QFLOWSET overwrote the sets stored in setid=101.
c to avoid this, a separate setid could be defined. For example:
c
      CALL QFLOWSET(1,'ROD',101,'CONE',0)
      CALL QFLOWSET(2,'ROD',110,'CONE',0)
      ...
      CALL QFLOW(1, CTEST, RTEST, QTEST)
      CALL QFLOW(2, XTEST, YTEST, ZTEST)
```

Restriction: A call to QFLOW set for a given *setid* must have been preceded by a call to QFLOWSET defining the node sets for that setid. A repeat call to QFLOWSET overwrites that setid unless a new one is defined.

Restriction: QFLOW assumes that the BUILD statement that was in effect when the corresponding QFLOWSET was called is still valid. *If a new BUILD statement is issued, call QFLOWSET again since the nodal interconnections will have changed.*

Guidance: The returned values are positive if the first list is losing heat to the second list: heat flow is calculated *from* the first list *to* the second list.

Caution: One-way conductors are handled from the perspective of the first list of nodes. This may cause the results to differ if list 1 and list 2 are reversed.

Guidance: The nodes in list 1 can also appear in list 2: they will simply have zero heat transfer to themselves and therefore not affect the resulting tally.

Caution: Do not include decimal points in array entries used to define node lists. SINDA/FLUINT cannot trap this type of error, and the program will abort in an internal call to NODTRN.

7.6.18 Sparse Matrix Memory Utilization Reporting

When MATMET=1 or 2 for a thermal submodel, nodes are solved simultaneously rather than iteratively using the YSMP method. The matrix inversion is performed using as sparse matrix package whose exact memory requirements cannot be predicted ahead of time. (YSMPWK is the equivalent routine for FLUINT matrix inversion.)

The section does *not* apply to MATMET=11 or 12, in which nodes are solved simultaneously using the AMG-CG method. See also Appendix F.

Subroutine Name: YSMPWS*

Description: This output routine is available to aid in the customizing of the YSMP (matrix inversion package) workspace allocation. YSMPWS simply reports on the actual usage (as experienced thus far in the run) of workspace by each thermal submodel employing MATMET=1 or 2. It is called automatically by the processor in the event of an abort due to insufficient allocation, and is available for direct user calls as well.

Calling Sequence:

```
CALL YSMPWS
```

Guidance: Since the utilization of workspace is cumulative, *place the call to YSMPWS at the end of OPERATIONS*. Calling this routine from other locations may result in lower than actual reported usages.

* Workspace is allocated automatically and no user involvement is required: users can ignore this routine, which is documented only for understanding older models.

7.6.19 Temporarily Holding a Node's Temperature

HTRNOD is a utility to temporarily hold a diffusion or arithmetic node's temperature, making it act as if it were a heater node (a special type of boundary node). BDYNOD is similar, except the node acts as if it were a boundary node. (The only difference between heater and boundary nodes is that a heater node's Q is updated by the HNQCAL or HNQPNT routines.) The RELNOD routine reverses this action, freeing the node to respond to changes.

The HTRMOD, BDYMOD, and RELMOD routines (below) provide analogous control to all diffusion and/or arithmetic nodes within a submodel.

Subroutine Name: HTRNOD

Description: This routine makes a diffusion or arithmetic node temporarily act as if it had been input as a heater node (a special type of boundary node for which the heat required to maintain the current temperature can be calculated). The actions of this routine are reversed by the RELNOD command below. There is no penalty for multiple redundant calls.

Guidance: This routine is useful for simulating heaters as boundaries or sized elements in steady states, then releasing the nodes to respond to thermostatic or other control in transients.

Guidance: The heat required to maintain the node is calculated or both calculated and printed using the HNQCAL or HNQPNT routines, respectively. This action temporarily changes the node's Q value. If this change is not desirable, elect BDYNOD instead (below).

Calling Sequence:

```
CALL HTRNOD(MODNAM, NODNUM)
```

where

`MODNAM` .... Thermal submodel name containing node NODNUM, in single quotes
`NODNUM` .... User integer identifier for node to be held

Example:

```
CALL HTRNOD('MOLTRES', 101)
```

Subroutine Name: BDYNOD

Description: This routine makes a diffusion or arithmetic node temporarily act as if it had been input as a boundary node. The actions of this routine are reversed by the RELNOD command below. There is no penalty for multiple redundant calls.

Calling Sequence:

```
CALL BDYNOD(MODNAM, NODNUM)
```

where

MODNAM . . . . . Thermal submodel name containing node NODNUM, in single quotes
NODNUM . . . . . User integer identifier for node to be held

Example:

```
CALL BDYNOD('PARRANDA', 1109)
```

Subroutine Name: RELNOD

Description: This routine reverses the action of a prior call to HTRNOD or ARITNOD, returning the node to its prior status as a diffusion or arithmetic node. There is no penalty for multiple redundant calls.

Calling Sequence:

```
CALL RELNOD(MODNAM, NODNUM)
```

where

MODNAM . . . . . Thermal submodel name containing node NODNUM, in single quotes
NODNUM . . . . . User integer identifier for node to be released

Example:

```
CALL RELNOD('MOLTRES', 101)
```

7.6.20 Temporarily Holding Many Nodes' Temperatures

In order to apply HTRNOD to all of the diffusion and/or arithmetic nodes within a submodel, the HTRMOD routine is available. BDYMOD similarly applies BDYNOD to all diffusion and/or arithmetic nodes in a submodel. The RELMOD routine reverses their actions.

The actions of HTRNOD, BDYNOD, RELNOD, HTRMOD, BDYMOD, and RELMOD are all independent, and can be used to hold customized collections of nodes. For example, to hold all diffusion nodes but one, use a call to HTRMOD followed by a call to RELNOD to release the one node that isn't to be held.

Subroutine Name: HTRMOD

Description: This routine makes all diffusion and/or arithmetic nodes in a submodel temporarily act as if they had been input as heater nodes (a special type of boundary node for which the heat required to maintain the current temperature can be calculated). The actions of this routine are reversed by the RELMOD command below. There is no penalty for multiple redundant calls.

Guidance: This routine is useful for holding all the diffusion nodes in the submodel constant, while letting only arithmetic nodes (and fluid submodels) respond perhaps to a steady state solution.

Guidance: The heat required to maintain the node is calculated or both calculated and printed using the HNQCAL or HNQPNT routines, respectively. This action temporarily changes the nodes' Q values. If this change is not desirable, elect BDYMOD instead (below).

Guidance: Calls to HTRMOD will erase prior actions by ARITMOD or RELNOD or similar routines acting on the same nodes. The last such action will prevail.

Calling Sequence:

```
CALL HTRMOD(MODNAM, EXTENT)
```

where

`MODNAM` .... Thermal submodel name, in single quotes, or 'ALL' to affect all thermal submodels

`EXTENT` .... 'A' to hold all arithmetic nodes in the submodel MODNAM
'D' to hold all diffusion nodes in the submodel MODNAM
'AD', 'DA', or 'ALL' to hold all diffusion and arithmetic nodes in the submodel MODNAM

Example:

```
CALL HTRMOD('COFFING', 'D')
```

Subroutine Name: BDYMOD

Description: This routine makes all diffusion and/or arithmetic nodes in a submodel temporarily act as if they had been input as boundary nodes. The actions of this routine are reversed by the RELMOD command below. There is no penalty for multiple redundant calls.

Guidance: This routine is useful for holding all the diffusion nodes in the submodel constant, while letting only arithmetic nodes (and fluid submodels) respond perhaps to a steady state solution. The DRPMOD and ADDMOD utilities are similar but less flexible.

Guidance: Calls to BDYMOD will erase prior actions by ARITMOD or HTRNOD or similar routines acting on the same nodes. The last such action will prevail.

Calling Sequence:

```
CALL BDYMOD(MODNAM, EXTENT)
```

where

MODNAM . . . . . Thermal submodel name, in single quotes, or 'ALL' to affect all thermal submodels

EXTENT . . . . . 'A' to hold all arithmetic nodes in the submodel MODNAM
'D' to hold all diffusion nodes in the submodel MODNAM
'AD', 'DA', or 'ALL' to hold all diffusion and arithmetic nodes in the submodel MODNAM

Example:

```
CALL BDYMOD('PACHANGA', 'A')
```

Subroutine Name: RELMOD

Description: This routine reverses the action of a prior call to HTRMOD, HTRNOD, BDYMOD, BDYNOD, ARITMOD, ARITNOD, or ARITCSG, returning the nodes to their prior status as diffusion or arithmetic. There is no penalty for multiple redundant calls.

If both arguments are missing, all holds in all thermal submodels are reversed.

Guidance: Calls to RELMOD will erase prior actions by ARITMOD or BDYNOD or similar routines acting on the same nodes. The last such action will prevail.

Caution: This action disrupts any RECESS or ACCRETE calls. Do not use RELMOD with any submodels containing such simulation calls.

Calling Sequence:

```
CALL RELMOD(MODNAM, EXTENT)
```

where

MODNAM . . . . . Thermal submodel name containing node NODNUM, in single quotes, or 'ALL' (or missing) to affect all thermal submodels

EXTENT . . . . . 'A' to release all arithmetic nodes in the submodel MODNAM
'D' to release all diffusion nodes in the submodel MODNAM
'AD', 'DA', or 'ALL' (or missing) to release all diffusion and arithmetic nodes in the submodel MODNAM

Example:

```
CALL RELMOD('COFFING', 'ALL')
CALL RELMOD       $ releases all holds in all thermal submodels
```

7.6.21 Temporarily Making Diffusion Nodes Arithmetic

ARITNOD, ARITMOD, ARITCSG, and ARIT_LIMNODE are utilities to temporarily make one or more diffusion nodes act like arithmetic nodes during a transient solution: their capacitance will be ignored in both temperature and time step calculations. (These routines have no effect on steady state solutions since capacitances are always ignored in those solutions.)

The RELNOD (Section 7.6.19) or RELMOD (Section 7.6.20) routines reverse this action, returning nodes to their normal status.

Note that the NODTAB output utility will list the node type as "ARITNOD" instead of "DIFF" for diffusion nodes that have been converted using any of these utilities.

Restriction: Avoid the use of these routines when the submodel also contains RECESS, RECESS_RATE, or ACCRETE calls, since those routines internally manipulate the status of nodes, causing a potentially severe conflict. If required, separate the submodels containing RECESS, RECESS_RATE, or ACCRETE calls from those requiring calling one of the utilities described in this subsection.

Subroutine Name: ARITNOD

Description: This routine makes a diffusion node temporarily act as if it had been input as an arithmetic node. The actions of this routine are reversed by the RELNOD routine (Section 7.6.19). There is no penalty for multiple redundant calls.

Guidance: This routine is useful for dealing with diffusion nodes whose size becomes negligibly small due to parametric changes or due to simulation actions (e.g., melting, parametric size changes, etc.). Otherwise, a diminishingly small diffusion node causes small time steps.

Guidance: This method is not as computationally efficient as declaring a node to be arithmetic in the first place (permanently).

Restriction: Do not call ARITNOD from within VARIABLES 1.

Restriction: Do not use with nodes that are involved in a RECESS or ACCRETE call.

Calling Sequence:

```
CALL ARITNOD(MODNAM, NODNUM)
```

where

MODNAM . . . . . Thermal submodel name containing node NODNUM, in single quotes
NODNUM . . . . . User integer identifier for the diffusion node to be treated as massless (arithmetic)

Example:

```
CALL ARITNOD('ICEMELT', 1032)
```

Subroutine Name: ARITMOD

Description: This routine makes all diffusion nodes within one or all submodels temporarily act as if they had been input as arithmetic nodes. The actions of this routine are reversed by the RELMOD routine (Section 7.6.20). There is no penalty for multiple redundant calls.

Guidance: This routine is useful for dealing with diffusion nodes whose size becomes negligibly small due to parametric changes or due to simulation actions (e.g., melting, parametric size changes, etc.). It is also useful for re-using a submodel built for a faster time-scale analysis in a slower time-scale case. Otherwise, a diminishingly small diffusion node causes small time steps.

Guidance: This method is not as computationally efficient as declaring a node to be arithmetic in the first place (permanently).

Guidance: Calls to ARITMOD will erase prior actions by HTRMOD or BDYMOD or similar routines acting on the same nodes. The last such action will prevail.

Restriction: Do not call ARITMOD from within VARIABLES 1.

Restriction: Do not use with submodels whose nodes are involved in a RECESS or ACCRETE call.

Calling Sequence:

```
CALL ARITMOD(MODNAM)
```

where

MODNAM . . . . . Thermal submodel name, in single quotes, or ‘ALL’ (or missing) to affect all thermal submodels

Example:

```
CALL ARITMOD('ICEMELT')
CALL ARITMOD      $ all arithmetic nodes in all submodels
```

Subroutine Name: ARITCSG

Description: This routine makes multiple diffusion nodes temporarily act as if they had been input as arithmetic nodes. It chooses which nodes to convert based on their characteristic time scale (in user units of time) or "CSG." CSG is the current nodal capacitance C divided by the current sum of adjacent conductance G: CSG = C/ΣG. Radiation conductances are linearized for this calculation, and the UA of adjacent FLUINT ties are included as well.

Because of the variability of C, G, and UA, some care must be taken to make sure that any relevant temperature-dependent or correlation-dependent (e.g., HTNC or HTP ties) updates have been made *before* calling ARITCSG. Therefore, if it is called only once, it should be called after a steady state or after a partial transient (at least the first time step).

Otherwise, because the CSG of each node can change during the solution, ARITCSG can also be called multiple times instead. It would be disruptive to the solution to call ARTICSG every time step, so if repetitive calls are deemed necessary, OUTPUT CALLS or OPERATIONS would be better calling locations (refer to the guidance below).

The actions of this routine are reversed by the RELMOD routine (Section 7.6.20). There is no penalty for multiple redundant calls.

Unlike ARITNOD and ARITMOD, ARITCSG does *not* affect (and does not override) the simulation status of diffusion nodes that have been placed in heater or boundary state by prior calls to routines such as HTRNOD or BDYNOD. Release those nodes using RELNOD or RELMOD *before* using ARITCSG if you wish ARITCSG to update their simulation status.

Guidance: If time- or temperature-dependent nodal Cs or conductor Gs are used, call ARITNOD after an initial STEADY call, or after an initial period in a transient if no STEADY solution is performed first, or consider repeated calls in a location such as OUTPUT CALLS.

Guidance: This routine is useful for dealing with diffusion nodes whose size becomes negligibly small due to parametric size changes or due to simulation actions (e.g., melting). It is also useful for re-using a model that was originally built for a short time-scale analysis in a longer time-scale case. Otherwise, a diminishingly small diffusion node causes small time steps.

Guidance: ARITCSG is intended to be used to make periodic adjustments to a model, perhaps because:

1. the Gs are volatile (as is often the case for linearized radiation conductances and two-phase fluid ties, or
2. the Cs and Gs are changing (as would happen if a dimension were being adjusted parametrically,) or
3. in the presence of two-phase fluid ties or other important fluid changes (e.g., significant velocity increases or decrease). Large UAs can drive down time steps if the tie is attached to a small diffusion node.

In the first and third cases (volatile CSGs), OUTPUT CALLS would be a logical location for placing the ARITCSG call. In the second case (changing dimensions), OPERATIONS or PROCEDURE or RELPROCEDURE would be recommended locations for the ARITCSG call.

Guidance: See also ARIT_LIMNODE. The node causing the limiting time step is not always the node with the smallest CSG. ARITCSG and ARIT_LIMNODE can be used together.

Guidance: This method is not as computationally efficient as declaring a node to be arithmetic in the first place (in NODE DATA). When the diffusion type was originally chosen, the energy associated with sensible heating of that node was deemed significant: it was presumably not allowed to "default" to a diffusion node just because it didn't limit the model's time step. So unless dimensions are changing during a run, or unless the user is re-using a model designed to answer a different question, ARITCSG should not be used in preference to a carefully crafted model. *This utility is provided as an emergency measure for a user who cannot correct a mesh or otherwise change the model itself.*

Guidance: Calls to ARITCSG will *not* erase prior actions by HTRMOD or BDYMOD acting on the same nodes. ARITCSG should not be used in submodels applying RECESS or ACCRETE.

Guidance: NODTAB reports both the CSG and the status of the node, designated as either DIFF (normal) or ARITNOD (ARITCSG or some other routine has placed this node in the arithmetic mode).

Restriction: Do not call ARITCSG from within VARIABLES 1. Avoid calls from within VARIABLES 2 or take extra caution in doing so, since this may disrupt or otherwise render inefficient the SINDA solution.

Calling Sequence:

```
CALL ARITCSG(MODNAM, CSGLIM)
```

where

`MODNAM` . . . . . Thermal submodel name, in single quotes, or 'ALL' to affect all thermal submodels

`CSGLIM` . . . . . Real value of a CSG (C/ΣG) cut-off for determining which nodes to be treated as massless (arithmetic, instantaneous) versus lagging (diffusion, with mass). Units of CSGLIM are user units of time (e.g., seconds, hours). The sign of CSGLIM determines what ARITCSG should do with nodes *exceeding* this threshold, as explained below.

If CSGLIM is positive, nodes larger than this threshold will be returned to diffusion status (if they are currently in arithmetic states, presumably by prior calls to ARITCSG). In other words, if any such nodes had been changed by prior ARITCSG calls, they will be returned to normal (diffusion node) status.

If instead CSGLIM is negative, the absolute value |CSGLIM| will be used as the threshold, and the negative sign will be used as signal to *not* change any nodes larger than CSGLIM to normal (diffusion node) status, leaving them at whatever status they had before the ARITCSG call. Therefore, repeated calls to ARITCSG with the same negative value of |CSGLIM| will not return converted nodes to normal (diffusion) status.

Caution: High values of |CSGLIM|, which would cause many nodes to be converted to arithmetic nodes, can actually degrade model performance instead of improve it. Start with small values and increase only as needed.

Example:

```
CALL ARITCSG('HUSTLE', THRESHOLD) $ Turns any nodes whose CSG is
      $ smaller than THRESHOLD into arithmetic in submodel HUSTLE
      $ and turns larger ones back to diffusion (normal) status
CALL ARITCSG('ALL',-0.001) $ Turns nodes smaller than 0.001 sec (or
      $ hours etc.) into arit nodes, does NOT change larger ones
      $ back to diffusion status due to the negative sign on CSGLIM
```

Subroutine Name: ARIT_LIMNODE

Description: This utility converts the diffusion node that most recently caused the limiting time step (which is not always the node with the smallest CSG) into an arithmetic node. Usually, this is a node with that is experiencing sharp changes in the source term (Q), or in the G of attached conductors, or in the UA or attached fluid ties.

This routine has the advantage of being much more targeted than ARITCSG in eliminating the *current* cause of the limiting time step. Also unlike ARITCSG, it is intended to be called frequently (perhaps every time step) to catch rapid changes that might happen between output intervals. Also unlike ARITCSG, ARIT_LIMNODE at most changes one node per call.

Unfortunately, there is no automatic means to reset this conversion, and repeated calls to ARIT_LIMNODE will only convert more and more of the model to arithmetic nodes. There are three mechanisms that limit or reset these actions:

1. One argument of ARIT_LIMNODE, DTLOW, prevents any conversions if the time step is above this threshold. In other words, conversions to arithmetic nodes are only performed when the time step has shrunk to some unacceptable level.

2. Another argument, RLIM, prevents too many conversions by imposing a limit on the ratio of the capacitance of converted nodes (by *any* means, including calls to ARITNOD, ARITCSG etc.). A recommended limit is on the order of 0.01 or 1%, meaning that at most 1% of the model's capacitance will be temporarily turned off.

3. Concurrent calls to ARITCSG in OUTPUT CALLS, with positive CSGLIM, are also helpful for resetting the some of the actions of ARIT_LIMNODE, in case nodes that represented an issue at an early point in the transient do not cause time step problems later in the simulation.

   If instead the troublesome nodes are reasonable constant through the run, then either *avoid* the call ARITCSG, or use it with negative CSGLIM and small absolute values. Also, consider small values of DTLOW in this case, to make sure that only the most problematic nodes are converted, since there will be no resetting otherwise and the RLIM limit will quickly be reached.

Caution: High values of DTLOW and/or RLIM, which would cause many nodes to be converted to arithmetic nodes, can actually degrade model performance instead of improve it. Start with small values and increase them only as needed.

The actions of this routine are reversed by the RELMOD routine (Section 7.6.20). There is no penalty for multiple redundant calls.

Calls to ARIT_LIMNODE are ignored except in transients.

Unlike ARITNOD and ARITMOD, ARIT_LIMNODE does *not* affect (and does not override) the simulation status of diffusion nodes that have been placed in heater or boundary state by prior calls to routines such as HTRNOD or BDYNOD. Release those nodes using RELNOD or RELMOD *before* using ARIT_LIMNODE if you wish ARIT_LIMNODE to update their simulation status.

Guidance: This method is not as computationally efficient as declaring a node to be arithmetic in the first place (in NODE DATA). When the diffusion type was originally chosen, the energy associated with sensible heating of that node was deemed significant: it was presumably not allowed to "default" to a diffusion node just because it didn't limit the model's time step. So unless dimensions are changing during a run, or unless the user is re-using a model designed to answer a different question, ARIT_LIMNODE should not be used in preference to a carefully crafted model. *This utility is provided as an emergency measure for a user who cannot correct a mesh or otherwise change the model itself.*

Guidance: Calls to ARIT_LIMNODE will *not* erase prior actions by HTRMOD or BDYMOD acting on the same nodes. ARIT_LIMNODE should not be used in submodels applying RECESS or ACCRETE.

Guidance: ARIT_LIMNODE is best called from VARIABLES 2, but may also be called within OUTPUT CALLS. If ARIT_LIMNODE is called each time step (VARIABLES 2), then it is often helpful to call ARITCSG as well (from OUTPUT CALLS, using a positive CSGLIM).

Restriction: Do not call ARIT_LIMNODE from within VARIABLES 1 or FLOGIC 1.

Calling Sequence:

```
CALL ARIT_LIMNODE(MODNAM, DTLOW, RLIM, RNOW, NODE, IO)
```

where

MODNAM .... Thermal submodel name, in single quotes, to check for the limiting node. This limits corrective actions to specific problem areas. Otherwise, use 'ALL' to check all thermal submodels.

DTLOW..... The input lower tolerable limit on time step. ARIT_LIMNODE will take no action if the current time step is greater than or equal to DTLOW. An input value that is zero or negative prevents *any* action.

RLIM ..... The input maximum fraction of diffusion nodes' capacitance that is allowed to be lost by conversion to arithmetic nodes, whether by this or similar routines (e.g., ARITCSG). A input value of zero prevents *any* action. An illegal value (more than 1.0 or less than 0.0) causes the default of 0.01 (1% of total active capacitance) to be applied.

RNOW ..... Returned fraction of converted to total capacitance of diffusion nodes. RNOW is often returned as zero, even if other nodes have previously been converted. Zero means that RNOW wasn't calculated since something else prevented the current limiting node from being converted (e.g., the time step was greater than DTLOW). If RNOW is greater than RLIM, no node was converted because too many others have already been converted. A message to this effect is printed to the output and message files when applying the verbose mode (see IO below)

NODE ..... Returned integer flag signaling that a node has been converted to arithmetic during this call. If NODE is zero, no node was converted during this call. If it is nonzero, NODE is the returned value of the *internal* sequence number of the converted node.

IO..... An optional integer input signaling whether to use the verbose mode (nonzero) for model debugging, or the quiet mode (zero, which is the default if IO is omitted).

Example:

```
CALL ARIT_LIMNODE('ALL', 1.0E-8, 0.02, rtest, ntest)
      $ checks all active submodels, and if the time step is
      $ smaller than 1.0e-8 and no more than 2% of the capacitance
      $ is neglected,
CALL ARIT_LIMNODE('ALL', 1.0E-8, 0.02, rtest, ntest, 1)
      $ same as the above, but in the verbose mode for diagnostics
CALL ARIT_LIMNODE('TA_MESH', 1.0E-8, 0.001, rtest, ntest)
IF(rtest .gt. 0.001) WRITE(NUSER1,*) 'TOO MANY NODES at ',TIMEN
IF(ntest .gt. 0) WRITE(NUSER1,*) 'NODE CONVERTED (SEQ. NO.) ',ntest
      $ Restricts conversions only to submodel TA_MESH and uses
      $ a smaller limit on the total converted capacitance (0.1%).
      $ Uses custom comments instead of the verbose mode
```

7.6.22 Execution Time Utilities

While the message file reports total processor ("astap.exe") clock time after a run, it is sometimes convenient to measure progress during a run, whether by clock time or CPU time.

For example, use a register as the argument for one of these routines to be able to plot TIMEN as a function of that register (on the X axis). Any steep regions in such a plot indicates a period in which many time steps are being taken, potentially warranting more investigation of those times.

Both of these timers are automatically initiated at the start of a run. A call to START_RUN (no arguments) will reset *both* of these timers if necessary.

SINDA_CLOCK

Description: This routine returns the wall clock time, in seconds, since the processor execution began (or since START_RUN has been last called).

Guidance: Use a real register as the argument to be able to plot TIMEN or other key variables as a function of execution time.

Guidance: Wall clock time will contain I/O time and not just solution time, which can be significant for large models with frequent output intervals. It will also be influenced by other processes running on the same machine.

Calling Sequence:

```
CALL SINDA_CLOCK(time)
```

where

`time` . . . . . A real result containing the wall clock time in seconds, since the processor execution began (or since START_RUN has been last called).

Example:

```
CALL SINDA_CLOCK(clocksec)
```

SINDA_CPU

Description: This routine returns the CPU time, in seconds, since the processor execution began (or since START_RUN has been last called).

Guidance: Use a real register as the argument to be able to plot TIMEN or other key variables as a function of execution time.

Guidance: CPU time does not contain I/O time and is independent of time spent by other processes running on the same machine.

Calling Sequence:

```
CALL SINDA_CPU(time)
```

where

`time` . . . . . A real result containing the CPU time, in seconds, since the processor execution began (or since START_RUN has been last called).

Example:

```
CALL SINDA_CPU(cpu_sec)
```

7.6.23 Print Time Step History List

When trying to determine what portions of a model are causing it to take the smallest time steps, it is helpful to print a list of the reasons for smallest time steps in the last transient.

This utility works in two parts. First, the size of the list to retain is declared in SETTL before a transient is invoked. Then a call to PRINTTL after the transient is used to print the history of the smallest time steps taken.

SETTL

Description: Allocate the size of the time step list for PRINTTL *prior* to a transient solution.

Restriction: SETTL should only be called in OPERATIONS.

Guidance: SETTL is normally only be called once per run. If it is invoked more than once, and if the previous and current list sizes (NUM) are not identical, then an error message will be produced but no further actions will be taken, and the history information already saved will not be reset.

If instead SETTL is called a second time with identical list sizes, then the saved list is deleted. This usage is recommended if a second independent transient is being profiled within the same run (see example in PRINTTL below). Multiple pairs of SETTL and PRINTTL calls can therefore be made for investigating multiple transient integrations, provided the argument to the SETTL calls are all the same.

Calling Sequence:

```
CALL SETTL(NUM)
```

where

NUM. . . . . Integer number of smallest time step points to save for later printing in PRINTTL. Typically, a number in the range of 50 to 100 is adequate.

Example:

```
CALL SETTL(100)
CALL TRANSIENT
CALL PRINTTL(0)
```

PRINTTL

Description: Prints the list of the N smallest time steps taken, and the reason for that limit (where discernible), where N is the list size chosen in the prior SETTL call.

Calling Sequence:

```
CALL PRINTTL(MAPS)
```

where

MAPS. . . . . Integer input, controlling whether or not NODMAPs (Section 7.4.2) and LMPMAPs (Section 7.11.8) are printed for the diffusion nodes or tanks that contributed to the smallest time steps. These maps are printed at the time PRINTTL is called, which might be at a point considerably different from that at which the node or lump caused the time step. The value of MAPS is interpreted as follows:
0 ... Don't print any maps, just the time step list
1 ... Print NODMAPs along with the list
2 ... Print LMPMAPs along with the list
3 ... Print both NODMAPs and LMPMAPs along with the list.

Example:

```
CALL SETTL(200)        $ initialize list size = 200
CALL TRANSIENT
CALL PRINTTL(0)        $ print with no maps
CALL SETTL(200)        $ reset for new list
TIMEO = 0.0
CALL TRANSIENT
CALL PRINTTL(3)        $ this time, print all maps
```

7.7 Applications Subroutines

7.7.1 Heater Node Energy Requirements

Subroutine Name: HNQCAL

Description: This routine calculates the heat necessary to maintain the heater nodes at their set temperatures. The results can be found in the heater node Q source locations. In this context, “heater nodes” includes any diffusion or arithmetic nodes that have been temporarily held as heater nodes using calls to HTRNOD or HTRMOD.

Guidance: May be called from any logic block. For a printout of the q-source data, use subroutine HNQPRT.

Restrictions: There must be at least one heater node in the submodel. Calls to this routine will reset the nodal Q for heater nodes, and should therefore only be called from VARIABLES 2, OUTPUT CALLS, PROCEDURE, and OPERATIONS. Input source data will be overwritten.

Calling Sequence:

```
CALL HNQCAL('SMN')
```

To update all heater nodes, simply use “CALL HNQCAL” (i.e., omit the argument).

7.7.2 Thermostatic Switch with Hysteresis

Subroutine Name: THRMST

Description: This subroutine provides a thermostatic switch with hysteresis. The dead-band (the difference between the upper and lower temperature limit) is specified by the user. The HEATER routine (Section 7.7.3) is more modern and should be used instead if possible. When applied to a nodal Q, THRMST is normally called from within VARIABLES 1.

Restrictions and Guidance: A unique user constant or other real Fortran variable must be defined for each thermostat. The initial value of the variable (0.0 or 1.0) must be initially specified, perhaps in the constants or user data block. The control constant DTMPCA (maximum allowed diffusion node temperature change) for the submodel containing node # would be some fraction of TLOW-THIGH if overshoot is an issue.

Guidance: THRMST is a simple controller. Consider also HEATER (Section 7.7.3) for more sophisticated thermostatic heater control.

Calling Sequence:

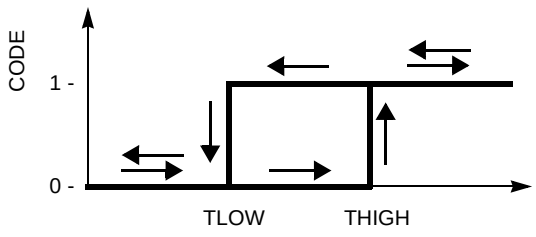
```
M     CALL THRMST (TSEN,TLOW,THIGH,CODE)
```

where:

TSEN. . . . . sensed temperature, usually Tn where n is an actual node number
TLOW. . . . . lower temperature limit
THIGH . . . . . upper temperature limit
CODE. . . . . user variable (e.g., register or user “constant”) unique to this thermostat

Example:

The figure at the right depicts the operation of the thermostatic switch. For TSEN values less than TLOW, CODE returns with a value of zero: for TSEN values greater than THIGH, CODE returns with a value of 1.0. For values between TLOW and THIGH, CODE may have a value of either 0.0 or 1.0.



The switch value CODE may be used in an F or M type statement to directly apply the heating or cooling rates. As an example of a thermostatic switch problem a typical cooling application is as follows: a thermostat is designed to switch on a cooling system rated at 200 BTU/hr at temperatures of 74 degrees and above. The cooling system remains on until 70 degrees is reached. The control thermostat is located at node 10. The cooling load is applied to source location 5. User constant 24 is assigned as the switch location:

```
CALL THRMST (CAB.T10,70.0,74.0,CAB.XK24)
```

To apply the cooling load, the following statement may be used after the above call (recall that summations are required for source values because they are reset and reconstructed each solution step):

```
Q5 = Q5 + CAB.XK24*(-200.0)
```

A typical heating application is as follows: a thermostat is designed to switch on a heater rated at 100 BTU/hr at 70 degrees and below. The heater remains on until 75 degrees is reached. The control thermostat is located at node CMP12.5. The heater load of 100 BTU/hr has been input in user constant CMP12.XK18. User constant CMP12.XK36 is assigned as the switch location.

```
CALL THRMST (CMP12.T5,70.0,75.0,CMP12.XK36)
```

To apply the heating load, the following statement may be used after the above call:

```
Q5 = Q5 + (1.0 - CMP12.XK36) * CMP12.XK18
```

Caution: Nonconvergence and/or small time steps may result if this routine is called from within VARIABLES 1 and NVARB1=1 and TRANSIENT is used. Also, the user should be aware that the signal “CODE” could be changed within each VARIABLES 1 pass if NVARB1=1. While this may help avoid unstable cycling, it does not represent a dead-band control system. A more realistic model would save the value of CODE in VARIABLES 0, and then reinitialize it before the call to THRMST in VARIABLES 1. For example:

```
HEADER VARIABLES 0, DEMO
      CTEST = CODE
HEADER VARIABLES 1, DEMO
      CODE = CTEST
      CALL THRMST(T404, 15.0, 25.0, CODE)
```

This resetting of CODE is not a concern with HEATER, Section 7.7.3, which also has provisions for use in steady state solutions. Use of HEATER is recommended instead of THRMST.

7.7.3 Generalized Heater Controller Simulations

The THRMST routine (Section 7.7.2) can be used to simulate any basic on/off controller with a deadband, while the HEATER and PHEATER routines documented in this subsection are more featured: they apply special control logic in steady states if applicable and they also report extra data relating to duty cycles. For controlling coolers (thermoelectric devices and other refrigerators), refer to Section 7.7.4.

Subroutine Name: HEATER (thermostatic)

Description: This subroutine simulates an electrical heater with a control system which turns the heater on when the sensor temperature falls below the "heater on" temperature TON, and turns the heater off when the sensor rises above the heater off temperature, TOFF. When the heater is on, the input Q value is added to the Q location specified by the user. When the heater is off, no heat is added. When applied to a nodal Q, HEATER is normally called from within VARIABLES 1. HEATER may be used for a fluid lump if it is called in FLOGIC 0 and the QL location of the lump is passed as the Q location, although the special control mode (below) will not apply to FLUINT lumps.

HEATER may also be used to size or simulate an ideal cooler if TON > TOFF and the QHT is negative. However, COOLCON (Section 7.7.4) may be a more appropriate choice if the control variable is not power (Q).

Restrictions: The use of such on/off control logic may cause convergence problems in steady-state solutions. (See also special control mode below.) If this happens, consider using the HTRNOD/RELNOD utilities (Section 7.6.19). Similar problems can occur in TRANSIENT solutions if NVARB1=1 and the call to HEATER is placed in VARIABLES 1, as noted below.

Calling Sequence:

```
 CALL HEATER (TSEN, Q, QHT, TON, TOFF, ONTIME,
+ SWITCH, CODE)
```

where:

TSEN . . . . . the sensed temperature
Q . . . . . the location for storing the heat
QHT . . . . . the heater heat rate
TON . . . . . the heater on temperature
TOFF . . . . . the heater off temperature (TON < TOFF if heater, else if QHT is negative, TON < OFF for cooler)
ONTIME . . . . . a real variable set by HEATER which contains the total time the heater was on. *Use the absolute value—ignore negative values.*
SWITCH . . . . . a real variable set by HEATER containing the number of times the heater was turned on or off.

CODE . . . . . a real variable set by HEATER (user sets CODE for first call):
0.0 if the heater was “on” at last call,
1.0 if the heater was “off” at the last call.
CODE should be a unique variable (e.g., register) for each heater used.

Example:

```
 CALL HEATER(T10, Q10, 500.0, 60.0, 80.0,
+  OTEST, STEST, XK10)
```

Caution: Nonconvergence (LOOPCT>NLOOPT, which may cause repeated time steps) and/or small time steps may result if this routine is called from within VARIABLES 1 and NVARB1=1 and TRANSIENT is used. (With respect to “CODE,” the code internally copes with repeated calls at the same time interval.) This problem will be especially acute if heating rates (QHT) are large, dead-bands (TOFF-TON) are small, and the capacitance of the controlled node is small.

Special Steady State Control Mode: If both “TSEN” and “Q” are passed as direct addresses to a single node’s T and Q cells, as is true in the above example, then HEATER automatically invokes a special simulation mode for steady states. During steady state solutions, the node’s temperature may be automatically controlled using internal calls to HTRNOD and RELNOD (Section 7.6.19), and the heater will be temporarily treated as a proportional heater. The Q rate reported, then, will be within the range of zero and QHT, corresponding roughly to the required duty cycle. This automatic mode is disabled if the arguments passed into HEATER do not directly correspond to a single node. For example, this logic will not function if “T10, Q11” is used, nor “TTEST, Q10” even if TTEST has been set to “T10” via a previous assignment statement or spreadsheet expression. **For this mode to function reliably, make sure all heat loads are updated *before* the HEATER call, including perhaps calling QVTEMP if temperature-variable sources have been applied to the controlled node.**

Subroutine Name: PHEATER (proportional)

Description: This subroutine simulates an electrical heater with a control system which turns the heater fully on (QHT) when the sensor temperature falls below the “heater on” temperature TON, and turns the heater off (or at least to a low user-specified value QHTL) when the sensor rises above the heater off temperature, TOFF. Between these temperature set-points, heat is applied proportionally. When the heater is on, the required heat rate is added to the Q location specified by the user. When the heater is off, no heat is added. When applied to a nodal Q, PHEATER is normally called from within VARIABLES 1. PHEATER may be used for a fluid lump if it is called in FLOGIC 0 and the QL location of the lump is passed as the Q location, although the special control mode (below) will not apply to FLUINT lumps.

PHEATER may also be used to size or simulate an ideal cooler if TON > TOFF and the QHT is negative. However, PCOOLCON (Section 7.7.4) may be a more appropriate choice if the control variable is not power (Q).

Restrictions: The use of such control logic may cause convergence problems in steady-state solutions. (See also special control mode below.) If this happens, consider using the HTRNOD/RELNOD utilities (Section 7.6.19). Similar problems can occur in TRANSIENT solutions if NVARB1=1 and the call to PHEATER is placed in VARIABLES 1, as noted below.

Calling Sequence:

```
 CALL PHEATER (TSEN, Q, QHT, QHTL, TON, TOFF, ONTIME,
+ QAVG)
```

where:

TSEN. . . . . the sensed temperature
Q. . . . . the location for storing the heat
QHT . . . . . the maximum heater heat rate (at or below TON if heater, at or above TON if cooler and QHT is negative)
QHTL. . . . . the minimum heater heat rate (at or above TOFF if heater, and normally zero)
TON . . . . . the heater on temperature
TOFF. . . . . the heater off temperature (TON < TOFF if heater, else if QHT is negative, TON < OFF for cooler)
ONTIME . . . . . a real variable set by PHEATER which contains the total time the heater was on. *Use the absolute value—ignore negative values.* The heater will be on all the time if QHTL is nonzero.
QAVG. . . . . the time-averaged heat rate required
QAVG should be a unique variable (e.g., register) for each heater used.

Example:

```
 CALL PHEATER(T10, Q10, 500.0, 0., 60.0, 80.0,
+  OTEST, XK10)
```

Caution: Nonconvergence (LOOPCT>NLOOPT, which may cause repeated time steps) and/or small time steps may result if this routine is called from within VARIABLES 1 and NVARB1=1 and TRANSIENT is used. This problem will be especially acute if heating rates (QHT) are large, dead-bands (TOFF-TON) are small, and the capacitance of the controlled node is small.

Special Steady State Control Mode: If both “TSEN” and “Q” are passed as direct addresses to a single node’s T and Q cells, as is true in the above example, then PHEATER automatically invokes a special simulation mode for steady states. During steady state solutions, the node’s temperature may be automatically controlled using internal calls to HTRNOD and RELNOD (Section 7.6.19). The Q rate reported, then, will be within the range of QHTL and QHT, corresponding roughly to the required duty cycle. This automatic mode is disabled if the arguments passed into PHEATER do not directly correspond to a single node. For example, this logic will not function if “T10, Q11” is used, nor “TTEST, Q10” even if TTEST has been set to “T10” via a previous assignment statement

or spreadsheet expression. **For this mode to function reliably, make sure all heat loads are updated *before* the PHEATER call, including perhaps calling QVTEMP if temperature-variable sources have been applied to the controlled node.**

7.7.4 Generalized Cooler Controller Simulations

The THRMST routine (Section 7.7.2) can be used to simulate any on/off controller with a deadband.

The HEATER and PHEATER routines (Section 7.7.3) may also be applied to controlling "coolers" too: cases where heat is *extracted* as needed to keep something cool. For this cooler simulation mode to be invoked, TON must be greater than TOFF, *and* the power applied (QHT) *must be negative*. (In PHEATER, the low power level QHTL can be zero but not positive for this cooler control mode to be recognized.)

With negative heat "loads," HEATER and PHEATER might be helpful for idealized refrigeration that neglects heat rejection at the hot side of the cooler, but they are inappropriate for models that include both sides of the thermoelectric cooler (TEC) or refrigeration cycle. For such inclusive models, the need for a more generic "cooler controller" is met by the utilities COOLCON and PCOOLCON, which are directly analogous to HEATER and PHEATER and have essentially the same arguments. Unlike HEATER and PHEATER, however, COOLCON and PCOOLCON assume TON>TOFF and they use an arbitrary variable, not heat, as the control parameter. Because the control variable is not necessarily the extracted power, the steady-state control logic that is part of HEATER/PHEATER is never invoked with COOLCON/PCOOLCON. Therefore, the user should consider use of HTRNOD/RELNOD for stable steady-state solutions.

In summary, use COOLCON or PCOOLCON to control the cold side of a cooler (e.g., the current or voltage on a TEC, or the RPM on a refrigeration compressor, etc.). Use HEATER and PHEATER to control heaters (including hot side control of a cooler), but one can also use those routines to simulate/size ideal coolers that control the (negative) nodal power directly.

Note: If power is applied as Vmax, it must be positive. If the resulting Vout is to be applied as a SINDA nodal power, it must be negated. (See Example 2.)

Subroutine Name: COOLCON (thermostatic cooler controller)

COOLCON is the cooler analog to HEATER ... refer to that routine for more details (Section 7.7.3). Like HEATER, COOLCON should be called in VARIABLES 1.

Calling Sequence:

```
      CALL COOLCON (TSEN, Vout, Vmax, TON, TOFF, ONTIME,
     + SWITCH, CODE)
```

where:

TSEN. . . . . the sensed temperature
Vout. . . . . the returned value: zero or Vmax
Vmax. . . . . the input cooler control parameter such as current, voltage, power, RPM, etc. (positive)
TON. . . . . the cooler on temperature
TOFF. . . . . the cooler off temperature (TON > TOFF)
ONTIME . . . . . a real variable set by COOLCON which contains the total time the cooler was on. *Use the absolute value—ignore negative values.*
SWITCH . . . . . a real variable set by COOLCON containing the number of times the cooler was turned on or off.
CODE. . . . . a real variable set by COOLCON (user sets CODE for first call):
0.0 if the cooler was "on" at last call,
1.0 if the cooler was "off" at the last call.
CODE should be a unique variable (e.g., register) for each cooler used.

Example:

```
         CALL COOLCON(T10, Amps, 50.0, 80.0, 60.0,
        +  OTEST, STEST, XK10)
C
C Example 2: the result is to be applied as a nodal power:
         CALL COOLCON(T45, ptest, ColdPow, TMax, Tmin,
        +  On101, Sw101, XK101)
         Q45 = Q45 - ptest
```

Subroutine Name: PCOOLCON (proportional cooler controller)

PCOOLCON is the cooler analog to PHEATER ... refer to that routine for more details (Section 7.7.3). Like PHEATER, PCOOLCON should be called in VARIABLES 1.

Calling Sequence:

```
         CALL PCOOLCON (TSEN, Vout, Vmax, Vmin, TON, TOFF,
        +    ONTIME, VAVG)
```

where:

`TSEN` . . . . . the sensed temperature
`Vout` . . . . . the returned value: between Vmin and Vmax (inclusive)
`Vmax` . . . . . the input higher end cooler control parameter at (TON) such as current, voltage, power, RPM, etc. (positive)
`Vmin` . . . . . the input lower end cooler control parameter (at TOFF) such as current, voltage, or power (nonnegative, and often zero)
`TON` . . . . . the cooler on temperature
`TOFF` . . . . . the cooler off temperature (TON > TOFF)
`ONTIME` . . . . a real variable set by PCOOLCON which contains the total time the cooler was on. *Use the absolute value—ignore negative values.* The cooler will be on all the time if Vmin is nonzero.
`VAVG` . . . . . the time-averaged V required (output; set by PCOOLCON)
VAVG should be a unique variable (e.g., register) for each cooler used.

Example:

```
         CALL PCOOLCON(T10, Volts, 24.0, 0., 80.0, 60.0,
        +  OTEST, ATEST)
C
C Example 2: the result is to be applied as a nodal power:
         CALL PCOOLCON(T45, ptest, ColdPow, 0., TMax, Tmin,
        +  On101, Pavg101)
         Q45 = Q45 - ptest
```

7.7.5 SINDA-based Heat Exchanger Calculations

This section documents historic utilities that predate the introduction of fluid submodels. Refer to Section 7.11.10 for more modern options. Despite being somewhat outdated, since simple single-phase flows can still be represented using SINDA nodes and one-way conductors on either one or both sides of the heat exchanger, these utilities may still find applications.

Subroutine Name: HXEFF

See also: HXMASTER (Section 7.11.10.3)

Description: This subroutine obtains the heat exchanger effectiveness either from a user constant or from a bivariate curve of effectiveness versus the flow rates on the two sides. The effectiveness thus obtained is used with the supplied flow rates, inlet temperatures and fluid properties to calculate the outlet temperatures. The user may specify a constant effectiveness by supplying a real number or may reference an array number to specify the effectiveness as a bivariate function of the two flow rates. The user supplies flow rates, specific heat values, inlet temperatures, and a location for the outlet temperatures for each of the two sides. The specific heat values may be supplied as a temperature dependent curve or a constant value may be supplied. The user also identifies enthalpy curves for each side which may be generated from the specific heat curve with user subroutine CRVINT.

Restrictions: HXEFF should be called in the VARIABLES 1 block. The value for EFF, the first argument must never be zero. TOUT1 and TOUT2 must be boundary nodes. *This routine is not related to FLUINT network modeling, but may be used in conjunction with it.*

Calling Sequence:

```
   CALL HXEFF (EFF, W1, W2, CP1, CP2, T1N1, T1N2,
+ TOUT1, TOUT2, H1, H2)
```

where:

`EFF` . . . . . (1) the effectiveness if real, (2) a curve number of a bivariate curve of effectiveness versus W1 and W2 if an array

`W1/2` . . . . . the flow rates for side 1 and 2 respectively

`CP1/2` . . . . . the specific heat value for side 1 and side 2 fluid respectively. Constant values may be input or arrays may be used for temperature dependent properties

`TIN1/2` . . . . . inlet temperatures - Usually T(IN1) and T(IN2) where IN1 and IN2 are the inlet nodes on side 1 and side 2

`TOUT1/2` . . . . the outlet temperature locations sides 1 and 2 where the calculated values will be returned. Must be boundary nodes

`H1/2` . . . . . arrays containing enthalpy vs. temperature for sides 1 and 2 respectively

Subroutine Name: HXCNT

See also: NTU2EFF (Section 7.11.10.2) and HXMASTER (Section 7.11.10.3)

Description: This subroutine calculates the heat exchanger effectiveness using the relationships for a counter flow type exchanger. The value of UA used in the calculations may be specified as a constant by supplying a real number or it may be specified as a bivariate function of the two flow rates by referencing an array number. The user supplies flow rates, specific heat values, inlet temperature and a location for the outlet temperatures for each of the two sides. The specific heat values may be supplied as a temperature dependent curve or a constant value may be supplied. The user also identifies enthalpy curves for each side which may be generated from the specific heat curve with user subroutine CRVINT.

Restrictions: HXCNT should be called in the VARIABLES 1 block. The value of UA, the first argument, must never be zero. TOUT1 and TOUT2 must be boundary nodes. *This routine is not related to FLUINT network modeling, but may be used in conjunction with it.*

Calling Sequence:

```
   CALL HXCNT(UA, W1, W2, CP1, CP2, TIN1, TIN2,
+ TOUT1, TOUT2, H1, H2)
```

where:

UA.	(1) the heat exchanger conductance if real, (2) a curve number of a bivariate curve of conductance versus W1 and W2 if an array
W1/2	the flow rates for side 1 and 2 respectively
CP1/2.	the specific heat value for side 1 and side 2 fluid respectively. Constant values may be input or arrays may be used for temperature dependent properties
TIN1/2	inlet temperatures - Usually T(IN1) and T(IN2) where IN1 and IN2 are the inlet nodes on side 1 and side 2
TOUT1/2 . . .	the outlet temperature locations sides 1 and 2 where the calculated values will be returned. Must be boundary nodes
H1/2	arrays containing enthalpy vs. temperature for sides 1 and 2 respectively

Subroutine Name: HXCOND

Description: This subroutine performs thermal analysis of a condensing heat exchanger. The effectiveness may either be supplied as a constant or as a trivariate function of humidity, flow rate of the gas, and flow rate of the coolant. CRVINT may be used to integrate the specific heat curves to produce the enthalpy curves.

Restrictions: HXCOND should be called in the VARIABLES 1 block. The value for EFF, the first argument, must never be zero. TGOUT and TCOUT must be boundary nodes. *This routine is not related to FLUINT network modeling, but may be used in conjunction with it.*

Calling Sequence:

```
 CALL HXCOND(EFF,WG,WC,NHG,NHC,
+ TGIN,TCIN,PSIIN,P,XLAM,XMIMO,PSIOUT,
+ WL,TGOUT,TCOUT)
```

where:

EFF	(1) the effectiveness if real, (2) a curve number of a trivariate curve of effectiveness versus PSIIN, WG, and WC
WG	the flow rate of the gas
WC	the flow rate of the coolant
NHG	the enthalpy curve for the gas
NHC	the enthalpy curve for the coolant
TGIN	the temperature of the incoming gas
TCIN	the temperature of the incoming coolant
PSIIN	the humidity of the incoming gas
P	the total gas pressure
XLAM	the latent heat of vaporization
XMIMO	the molecular weight ratio Mv/Mo
PSIOUT	the outlet humidity
WL	the flow rate of the liquid
TGOUT	the temperature of the outgoing gas
TCOUT	the temperature of the outgoing coolant

Subroutine Name: HXCROS

See also: NTU2EFF (Section 7.11.10.2) and HXMASTER (Section 7.11.10.3)

Description: This subroutine calculates the heat exchanger effectiveness using the relationships for a cross flow type exchanger. The value of UA used in the calculations may be specified as a constant by supplying a real number or it may be specified as a bivariate function of the two flow rates by referencing an array number. Any one of the following four types of cross flow exchangers may be analyzed.

1. Both streams unmixed
2. Both streams mixed
3. Stream with smallest mCp product unmixed

4. Stream with largest mCp product unmixed

The type is specified by the 10th argument in the call statement. The user supplies flow rates, specific heat values, inlet temperatures and a location for the outlet temperatures for both sides. The specific heat values may be supplied as a temperature dependent curve or a constant value may be supplied. The user also identifies enthalpy curves for each side which may be generated from the specific heat curve with user subroutine CRVINT.

Restrictions: HXCROS should be called in the VARIABLES 1 block. The value for UA, the first argument, must never be zero. TOUT1 and TOUT2 must be boundary nodes. *This routine is not related to FLUINT network modeling, but may be used in conjunction with it.*

Calling Sequence:

```
 CALL HXCROS (UA,W1,W2,CP1,CP2,TIN1,TIN2,
+ TOUT1,TOUT2,K,H1,H2)
```

where:

`UA` . . . . . (1) the heat exchanger conductance if real, (2) a curve number of a bivariate curve of conductance versus W1 and W2 if an array

`W1/2` . . . . . the flow rates for side 1 and 2 respectively

`CP1/2` . . . . . the specific heat value for side 1 and side 2 fluid respectively. Constant values may be input or arrays may be used for temperature dependent properties

`TIN1/2` . . . . inlet temperatures - Usually T(IN1) and T(IN2) where IN1 and IN2 are the inlet nodes on side 1 and side 2

`TOUT1/2` . . . the outlet temperature locations sides 1 and 2 where the calculated values will be returned. Must be boundary nodes

`K` . . . . . the code specifying type of cross flow exchanger:
Both streams unmixed: K = 1
Both streams mixed: K = 2
Stream with small m*Cp unmixed: K = 3
Stream with large m*Cp unmixed: K = 4

`H1/2` . . . . . arrays containing enthalpy vs. temperature for sides 1 and 2 respectively

Subroutine Name: HXPAR

See also: NTU2EFF (Section 7.11.10.2) and HXMASTER (Section 7.11.10.3)

Description: This subroutine calculates the heat exchanger effectiveness using the relationships for a parallel flow type exchanger. The value of UA used in the calculations may be specified as a constant by supplying a real number or it may be specified as a bivariate function of the two flow rates by referencing an array. The user supplies flow rates, specific heat values, inlet temperatures

and a location for the outlet temperatures for each of the two sides. The specific heat values may be supplied as a temperature dependent curve or a constant value may be supplied as a temperature dependent curve or a constant value may be supplied. The user also identifies enthalpy curves for each side which may be generated from the specific heat curve with user subroutine CRVINT.

Restrictions: HXPAR should be called in the VARIABLES 1 block. The value of UA, the first argument, must never be zero. TOUT1 and TOUT2 must be boundary temperatures. *This routine is not related to FLUINT network modeling, but may be used in conjunction with it.*

Calling Sequence:

```
 CALL HXPAR(UA,W1,W2,CP1,CP2,TIN1,TIN2,TOUT1,
+ TOUT2,H1,H2)
```

where:

UA . . . . . (1) the heat exchanger conductance if real, (2) a curve number of a bivariate curve of conductance versus W1 and W2 if an array

W1/2. . . . . the flow rates for side 1 and 2 respectively

CP1/2 . . . . . the specific heat value for side 1 and side 2 fluid respectively. Constant values may be input or arrays may be used for temperature dependent properties

TIN1/2 . . . . . inlet temperatures - Usually T(IN1) and T(IN2) where IN1 and IN2 are the inlet nodes on side 1 and side 2

TOUT1/2. . . . the outlet temperature locations sides 1 and 2 where the calculated values will be returned. Must be boundary nodes

H1/2. . . . . arrays containing enthalpy vs. temperature for sides 1 and 2 respectively

7.7.6 Material Conductance and Capacitance

Subroutine Name: GADDi

Description: The GADD family of subroutines will combine materials and conduction paths to form a single node or conductor. They calculate the effective thermal capacitor or conductor value for the given material network. There are 22 conductor network cases and capacity for up to ten materials in a single node. These cases are shown in Figure 7-1.

To use the routines, the standard 3-blank or GEN options should be used to define the composite node or conductor. Then a single call to a GADD subroutine will evaluate the composite conductor or capacitor value.

Restrictions: GADDi should be called in VARIABLES 1.

Calling Sequence:

CALL GADDi(G,NCASE,TA,TB,A_1,F_1,A_2,F_2...,A_i,F_i)

where:

- i ..... Number of conductor elements, or capacitor materials in the network (integer values 2 to 10)
- G ..... The conductor or capacitor being evaluated
- NCASE ..... The conductor or capacitor network case from the following figure
- TA,TB ..... Temperatures used for determining temperature dependent properties. The subroutine uses the average of TA and TB to interpolate. Input 0.0 for both if the material properties are constant. For conductors, TA and TB are the temperatures of the two nodes connected by the composite conductor. For capacitors, TA and TB are both equal to the node temperature.
- A_i ..... Material property array of the i^{th} element in the network. Standard doublet array of property versus temperature. For constant properties input a zero (0.0).
- F_i ..... Multiplying factor for corresponding interpolated A_i value

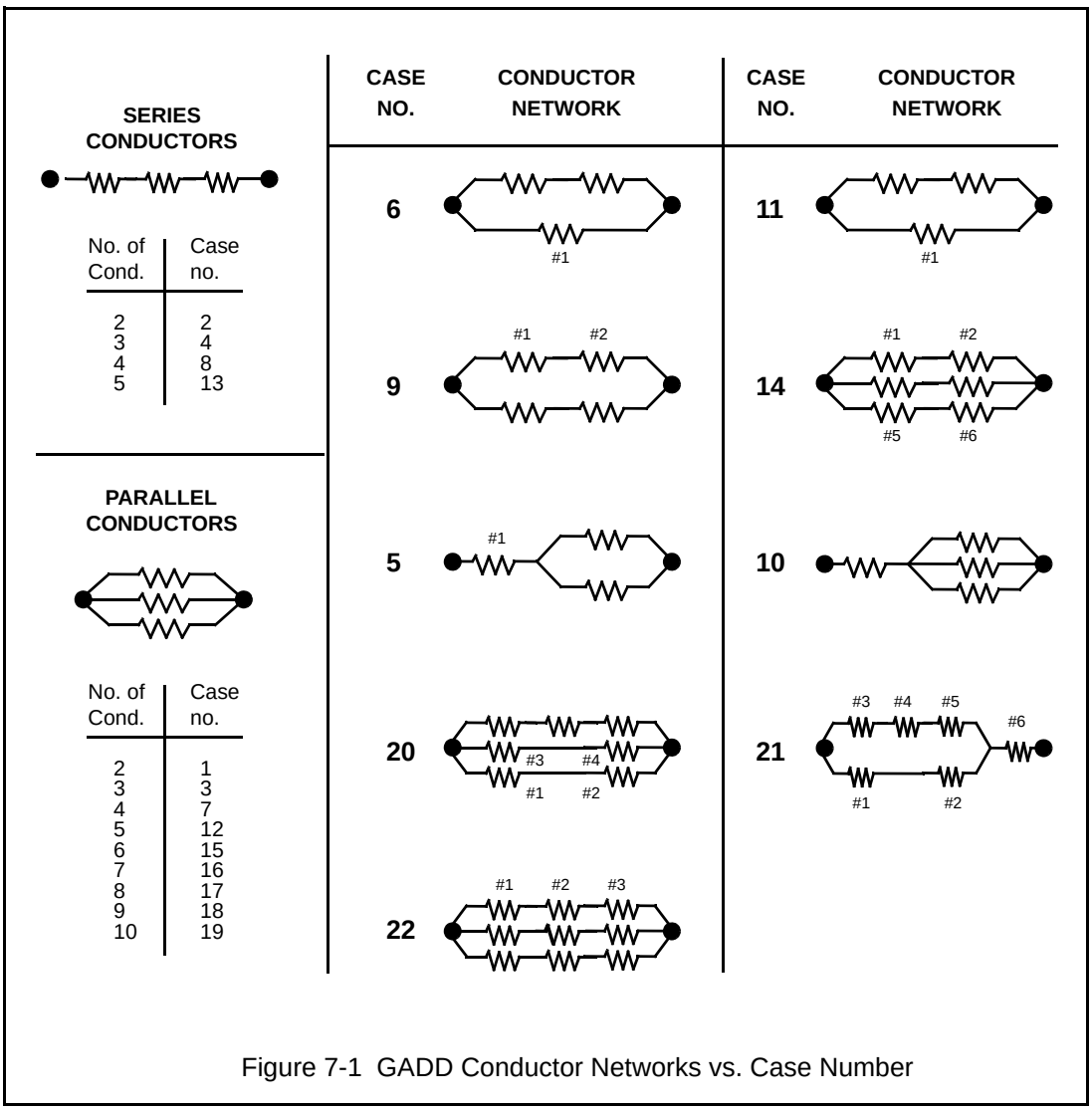


Figure 7-1 GADD Conductor Networks vs. Case Number

7.7.7 Phase Change Simulation

A routine is provided to help simulate phase change materials (PCMs) using SINDA nodes.

Alternatively, the user can consider using one or more FLUINT tanks with stagnant flows to simulate certain classes of phase change. For example, the user might specify a compliant tank and a 9000 series fluid whose Cp curve, when integrated, yields the correct behavior. Because it uses the integrated curve instead of holding Cp constant per solution step as does SINDA, FLUINT is often faster and more accurate, and takes care of the time step internally.

Subroutine Name: FUSION

Description: This subroutine simulates the phase change of a node. The user creates a plain (3-blanks) diffusion node and then calls this routine to control its temperature as needed to simulate phase change.

The energy in or out of the node maybe applied directly, or by any valid conductor or tie. This routine should be called from VARIABLES 2. The capacitance of the phase change node is adjusted accordingly to account for the change in phase, while changing phase temperature of the node is held constant. Freezing and melting cycles are allowed.

This routine does nothing if called from within a steady state, except to reinitialize the user workspace.

The submodel name and node number of the phase change node are supplied as the first two arguments. An array defining phase change properties is supplied as the third argument. This is a singlet array, values in the array are described below. Units must be consistent with the rest of the model.

The last two arguments are returned by the routine for optional use in setting DTIMEH and/or DTMPCA as needed to avoid missing the beginning or ending of the phase change. The program will issue a warning otherwise if the changes are too severe. When phase change begins or ends, a message will be printed.

Calling Sequence:

```
CALL FUSION('SMN', NODE, A(IC), DTH, DTM)
```

where

SMN ..... model name containing the phase change node (enclosed in ')
NODE ..... node number for the phase change node

A. . . . . .user defined singlet array containing property information:
A+1 . . . . . .input phase change material mass
A+2 . . . . . .input specific heat of the phase material when in liquid state
A+3 . . . . . .input specific heat of the phase material when in solid state
A+4 . . . . . .input fusion temperature
A+5 . . . . . .input heat of fusion
A+6 . . . . . .reserved space for internal use. Initialize as 0.0
A+7 . . . . . .reserved space for internal use. Initialize as 0.0
A+8 . . . . . .reserved space for internal use. Initialize as 0.0

DTH. . . . . .returned recommended maximum value of DTIMEH (considering this node's phase change only)

DTM. . . . . .returned recommended maximum value of DTMPCA (considering this node's phase change only)

The returned values of DTH and DTM can be ignored at the user's risk. Otherwise, *for single calls to FUSION and no other logic affecting DTIMEH and DTMPCA*, the call to FUSION might appear as follows:

```
CALL FUSION('MYMOD',45,MYMOD.A50, MYMOD.DTIMEH, MYMOD.DTMPCA)
```

If there is only one phase change node and NO OTHER limitations on DTIMEH and DTMPCA, then the user might call FUSION as shown in the following example:

```
 CALL FUSION('MYMOD', 45, MYMOD.A50, MYMOD.DTIMEH,
+     MYMOD.DTMPCA)
```

Otherwise *each* call should appear as illustrated by the following, which does not disturb other influences on the values of DTIMEH and DTMPCA:

```
CALL FUSION('MYMOD',45,MYMOD.A50, DTEST, TTEST)
MYMOD.DTIMEH    = MIN(MYMOD.DTIMEH, DTEST)
MYMOD.DTMPCA    = MIN(MYMOD.DTMPCA, TTEST)
```

Be sure to reset DTIMEH and DTMPCA at the start of each time step to their original desired values, otherwise the above *example* logic will cause an ever diminishing time step.

Guidance: Multiple calls to FUSION are allowed, however each call needs to have a unique node number and property array.

Guidance: Internally uses DRLXCA as a measure of tolerance in terms of warning the user and controlling DTH and DTM.

Guidance: To turn off warnings, call the routine FUSION_MSG_OFF (no arguments) in OPERATIONS or in any relevant logic block. For example:

```
CALL FUSION_MSG_OFF
```

Calling FUSION_MSG_ON reverses the action of this routine.

7.7.8 PID Controller Simulation Utilities

This section documents a series of subroutines useful for modeling one or more PID (proportional-integral-differential) controllers. The interrelated routines are:

PIDSET. . . . . initializes a PID controller
PIDINIT . . . . . initializes or re-initializes a PID controller
PIDSETW. . . . sets limits on the control variable (CV), and optionally on the wind-up
PID . . . . . models a continuous PID controller in a transient solution
PIDS . . . . . models a PID controller in a steady state solution, or a discrete-interval (finite rate sampling) controller during a transient
PIDGET . . . . . returns performance metrics for a PID controller
PIDTAB . . . . . output tabulation of PID controllers

A PID controller changes a *control variable* (CV) such that a *process variable* (PV) is maintained as close as possible to its *set point* (SP). The operation of a PID controller is completely determined by three factors or *gains*: the proportional gain (G_p), the integral gain (G_i), and the differential gain (G_d). Given that the error is defined as the difference between SP and the current value of PV:

$$E = SP - PV$$

then the action of the PID controller is defined by:

$$CV = G_p*E + G_i*\int E\,dt + G_d*dE/dt$$

To oversimplify, the proportional gain has primary responsibility for returning the process variable to its set point. If that gain is too small (in magnitude) the control may drift. If it is too large the controller may cause an oscillation. The integral gain helps prevent long-term drift (“droop”) off of the set point, and attempts to assure that the error is as much positive as it is negative. The differential gain can help adjust for fast versus slow changes, but can also easily cause oscillations and is susceptible to noise. Choosing the values of the gains to use is called tuning. The Solver (Section 5.1) can be used to analytically tune PID controllers, otherwise the gains are assumed to be derived from tests or are otherwise user-provided.

Up to 100 PID controllers may be in use at any time. They are distinguished from one another by a user-provided integral identifier, similar to a node or lump ID.

PIDSET is called to initialize the gains (P, I, and D factors) on a controller. Although it is intended to be called before calling a solution routine, it may be called at any time to reset the gains for any controller. PIDSETW may be used to set limits on the control variable (CV), as well as to limit the wind-up of the integral term. PIDINT may be used to reset the integral term and other values, perhaps using results from a prior steady-state as the initial conditions for a transient.

PID performs the simulation of the controller. It is intended to be called from either VARIABLES 2 or FLOGIC 2 during a transient.

PID cannot be used in a steady-state solution. PIDS is therefore available as an alternative: it is similar to PID but is set in VARIABLES 0 or FLOGIC 0 during a steady state, and the user must provide a value of "time step" (corresponding to a relaxation step) that is appropriate for the values of the integral and differential gains. PIDS may also be called from OUTPUT CALLS if a discrete-interval controller is needed.

PIDGET can optionally be called to determine the performance of the controller (peak error, integrated error, and RMS error) perhaps as required to tune the gains on the controller.

Subroutine Name: PIDSET

Description: This subroutine is used to set (or reset) the three gains for a PID controller. It is normally called before a solution routine in OPERATIONS, PROCEDURE, or RELPROCEDURE, but can be called more frequently (e.g., in VARIABLES, FLOGIC, or OUTPUT CALLS blocks) if the gains must be reset during a solution.

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in subsequent PID, PIDS, or PIDGET calls when referring to the same controller.

Restrictions: No more than 100 PID controllers can be defined.

Calling Sequence:

```
CALL PIDSET(num, Gp, Gi, Gd)
```

where:

`num` . . . . . unique PID controller integer identifier
`Gp` . . . . . proportional gain (input)
`Gi` . . . . . integral gain (input)
`Gd` . . . . . differential gain (input)

Examples:

```
CALL PIDSET( 100, 10., 0.01, 0.0)
CALL PIDSET( 30212, gainp, gaini, gaind)
```

Subroutine Name: PIDINIT

Description: This subroutine is used to set (or reset) the internally stored integral term based on gains that have already been set (via PIDSET calls) and based on current values of the set point (SP), process variable (PV), and control variable (CV). PIDINIT effectively resets the history for this controller, and is therefore useful at the start of parametric runs, for example. It may be called repeatedly as needed.

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in previous PIDSET calls and in subsequent PID, PIDS, or PIDGET calls when referring to the same controller.

Restrictions: PIDSET must be called before this routine.

Caution: If PIDINIT is ever called for a controller, the automatic resetting of that controller at the start of a new solution routine is thereafter disabled: the user has assumed control.

Calling Sequence:

```
CALL PIDINIT(num, SP, PV, CV)
```

where:

`num` ..... unique PID controller integer identifier
`SP`..... set point (input)
`PV`..... process variable (current value, input)
`CV`..... control variable (current value, input)

Examples:

```
CALL PIDINIT( 106, 100.0, Tnow, ValveP)
CALL PIDINIT( 3121, Tneed, Theater, Qheater)
```

Subroutine Name: PIDSETW

Description: This subroutine is used to set (or reset) the limits of CV for a PID controller, and to also (optionally) limit the wind-up for the integral term. PIDSETW can be called repeatedly as needed.

PIDSETW sets limits on the control variable (CV) value, and can also optionally set corresponding wind-up limits on the integral term. If the last argument (Iwup) is 1 (or nonzero), the internally stored integral term is limited to be between CV_{min}/G_i and CV_{max}/G_i where Gi is the current value of the integral gain (which must be nonzero to use this option). Otherwise, if Iwup is zero, the returned CV values will be limited but not the integral term.

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in previous PIDSET calls and in subsequent PID, PIDS, or PIDGET calls when referring to the same controller.

Restrictions: G_i must be nonzero if Iwup is nonzero. Therefore, PIDSET must be called before this routine, and perhaps PIDINT as well.

Calling Sequence:

```
CALL PIDSETW(num, CVmin, CVmax, Iwup)
```

where:

`num`.....unique PID controller integer identifier
`CVmin` .....minimum value of CV permitted (input)
`CVmax` .....maximum value of CV permitted (input)
`Iwup`.....signal to limit the integral wind-up based on CVmin, CVmax, and the current value of the integral gain, G_i (input). Use Iwup=1 to limit the wind-up, otherwise Iwup=0 signals that no limit is to be applied.

Examples:

```
CALL PIDSETW( 160, 0.1, 1.0, 0)
CALL PIDSETW( 4121, closed, open, 1)
```

Subroutine Name: PID

Description: This subroutine is used to simulate a continuously (rapidly) sampling PID controller during a transient. It is normally called in VARIABLES 2 or FLOGIC 2 (at the end of the time step).

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in a previous PIDSET call in which the gains were specified.

PID adjusts the control variable based on the results of the last time step, and the new value is applied to the next time step. Thus, there is some built-in lag in the simulation and perhaps the time step must be reduced to reduce this effect.

Caution: Although not illegal, it is inadvisable to adjust the set point dynamically during the run.

Restriction: This routine cannot be called during a steady state solution since the time step is unknown. Use the NSOL flag to prevent accidental calls during steady states (as shown in the example below), and consider perhaps using PIDS calls instead during steady states.*

* Note that PIDS, however, should not normally be called from the same location as PID.

Restriction: The internal history inside each controller is reset at the start of a new solution routine as long as PIDINIT has never been used for that controller.

Guidance: The internal history is not retained if a model is restarted from a SAVE file, but is restored if a model is restarted from a CSR Folder.

Restriction: Do not use this routine in conjunction with the BACKUP flag.

Caution: If a real PID controller is being simulated,* care should be taken to avoid sampling faster than the real controller will sample. In other words, if the time step decreases below the sampling interval of the controller, then the action of PID should be delayed and the time interval between actions should be specified (rather than letting PID use the last time step). In this case, consider using PIDS (below) instead of PID, calling PIDS from within OUTPUT CALLS and passing the output interval as the sampling interval (the last argument in PIDS).†

Calling Sequence:

```
CALL PID(num, SP, PV, CV)
```

where:

`num` ..... PID controller integer identifier corresponding to the ID used in the PIDSET call which set the gains for this controller
`SP`..... set point (input)
`PV`..... process variable (current value, input)
`CV`..... control variable (output)

Examples:

```
HEADER FLOGIC 2 ...
      IF(NSOL .GE. 2) CALL PID( 100, 10.0, pl100, fk14)
HEADER VARIABLES 2 ...
      IF(NSOL .GE. 2) CALL PID( 30212, Tset, 0.5*(T3021+T3022), power)
```

Subroutine Name: PIDS

Description: This subroutine is used to simulate a PID controller during a steady state given a user-input artificial time step. It is normally called in VARIABLES 1 or FLOGIC 0 (at the beginning of a relaxation step in STEADY or STDSTL). However, it can alternatively be called in an OUTPUT CALLS blocks to simulate a discrete-interval controller during a transient.

* Versus a “fake” PID controller added strictly to drive the answer to a desired solution.
† A fake submodel can be added whose whole purpose is to provide an OUTPUT CALLS block with the desired sampling rate (and perhaps containing no output operations at all).

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in a previous PIDSET call in which the gains were specified.

PIDS adjusts the control variable based on the results of the last iteration, and the new value is applied to the next iteration. Thus, there is some built-in lag in the simulation.

Caution: Although not illegal, it is inadvisable to adjust the set point dynamically during the run.

Caution: This routine is normally only called within a steady state solution since the time step is user-specified. Use the NSOL flag to prevent accidental calls during transients (as shown in the example below). However, if for some reason (perhaps to model a discrete-interval controller) the user wishes to override the actual time step with a user-imposed one, PIDS can be called in transients instead of PID.

Restriction: The internal history inside each controller is reset at the start of a new solution routine as long as PIDINIT has never been used for that controller.

Calling Sequence:

```
CALL PIDS(num, SP, PV, CV, DT)
```

where:

num	PID controller integer identifier corresponding to the ID used in the PIDSET call which set the gains for this controller
SP	set point (input)
PV	process variable (current value, input)
CV	control variable (output)
DT	artificial time step to use for one STEADY or STDSTL relaxation step (input, should correspond to the units of the G_i and G_d specified for this controller in PIDSET), or output interval if being called during a transient to simulate a discrete-rate controller.

Examples:

```
HEADER FLOGIC 0 ...
      IF(NSOL .LE. 1) CALL PIDS( 100, 10.0, pl100, fk14, DTchar)
HEADER VARIABLES 1 ...
      IF(NSOL .LE. 1) CALL PIDS( 30212, Tset, 0.5*(T3021+T3022), power, 1.)


HEADER OUTPUT CALLS ...
      IF(NSOL .GE. 2) CALL PIDS (102, 2000., rpm, smfr102, outptf)
```

Subroutine Name: PIDGET

Description: This subroutine is used to fetch performance metrics for a PID controller. These metrics include:

peak error . . . . the maximum excursion of PV versus SP since the last solution started (may be positive or negative)

integrated error the integral of error since the last solution started. This is the same number operated upon by the integral gain G_i (may be positive or negative)

RMS error. . . . the root-mean-square error since the last solution started (always positive)

PIDGET is normally called after a solution routine in OPERATIONS, PROCEDURE, or REL-PROCEDURE, but can be called more frequently (e.g., in VARIABLES, FLOGIC, or OUTPUT CALLS blocks) if the performance needs to be monitored during a run.

The controller must be specified by a numeric (integer) identifier. This identifier must be the same as that used in previous PIDSET, PID, and/or PIDS calls when referring to the same controller.

Restrictions: Because the internal history of each controller is reset at the start of a new solution routine, the results of PIDGET reflect the data accrued since the last time the controller was reset.

Calling Sequence:

```
CALL PIDGET(num, Epk, Eint, Erms, Nsteps)
```

where:

`num` . . . . . unique PID controller integer identifier

`Epk` . . . . . peak (maximum or minimum) error (output)

`Eint` . . . . . integrated error (output)

`Erms` . . . . . RMS error (output)

`Nsteps` . . . . Number of steps (time steps or relaxation steps) upon which the above data is based: the number of steps since the start of the last solution routine (output integer)

Examples:

```
CALL PIDGET( 102, Emax, ETEST, OBJECT, NTEST)
```

Guidance: Tuning a PID controller using the Solver. One of the purposes of PIDGET is to return information that can be used to calculate better values of the P, I, and D gains: to help tune the controller. The Solver module (Section 5.1) can be used to automate this process.

To find the gains using the Solver, they must first be declared as registers and then as design variables. The PROCEDURE block consists of a PIDSET call (applying the current values of the design variables as gains), a steady-state or transient (note that gains tuned for one will rarely be applicable for the other), and a PIDGET call to return the objective (OBJECT) and perhaps some constraints.

The peak error is usually highly discontinuous and should not be used as an objective nor a constraint. The integrated error may be used as an objective, setting GOAL to zero, or it may be set to a constraint (defining maximum and minimum limits on the integrated error). The RMS error is useful directly as the objective to be minimized (GOAL=-1.0E-30, the default), but can also be applied as a constraint.

When tuning a “fake” PID controller in a steady state solution, the objective is normally to achieve the fastest converging solution.* Thus, Nsteps might be used as objective:

```
OBJECT= FLOAT(NSTEPS)
```

However, the user is cautioned that the above objective function is discontinuous and that unless substantial variable exists between runs (quasi-continuous), then the Solver may stall.

If a continuous controller (PID inside of FLOGIC 2 or VARIABLES 2 in a transient) is being tuned, then the user is cautioned that the Solver may find a set of unrealistic gains that minimize error at the expense of slow runs (small time steps). Therefore, a hybrid objective might be defined to trade-off a fast run (few time steps) with adequate control:

```
OBJECT= Erms*FLOAT(NSTEPS)
```

The above problems with diminishing time steps caused by the PID controller are not present with a discrete-rate controller (PIDS inside of OUTPUT CALLS in a transient). In that case, simply minimizing the RMS error is often adequate.

If multiple PID controllers are present in a system, then ideally they should all be tuned at the same time (i.e., all gains declared as design variables). However, this means that OBJECT will need to be a composite of the RMS errors of the multiple controllers.

* Unlike other cases where the PROCEDURE is a single steady-state run, in this case a call to SVPART and RESPAR are needed to make sure each steady state run starts from the same conditions such that changes to design variables (gains) can be meaningfully compared.

Subroutine Name: PIDTAB

Description: This tabulation-style output routine lists one line of data (gains, current set points, process and control variable values, etc.) per PID controller.

Calling Sequence:

```
CALL PIDTAB
```

7.7.9 Heat Pipes

This section documents two subroutines, HEATPIPE and HEATPIPE2, that can be used to simulate:

1. constant or fixed conductance heat pipes (FCHPs, CCHPs) without noncondensible gases (NCG)

2. FCHPs (CCHPs) with gas (i.e., degraded via gas generation)

3. gas-loaded variable conductance heat pipes (VCHPs)

These three cases, along with a node and conductor network diagram corresponding to a sample axial subdivision of the pipe, are illustrated in Figure 7-2. That figure corresponds to the one dimensional (1D) routine HEATPIPE, whereas 2D (axially and circumferentially discretized) models are also possible using HEATPIPE2.

The intent of these high-level routines is to simulate the performance of heat pipes within thermal management systems, and not necessarily to contribute to the internal design of the heat pipe itself. Therefore, no attempt is made to explicitly model the liquid and vapor flow within a heat pipe. Because of this top-level treatment, this routine is applicable to a wide variety of heat pipe wick structures (wall wick, slab wick, axially grooved, etc.) and working fluids (cryogens, organics, liquid metals, etc.).

Although unit conventions (specified by UID in global control data, Section 4.3) and fluid property blocks are “borrowed” from the FLUINT side of the program, these routines operate strictly within SINDA thermal networks and do not require that any fluid submodels be used. Note that loop heat pipes (LHPs) and related technologies such as capillary pumped loops (CPLs), on the other hand, *do* require that fluid submodels be used: while HEATPIPE and HEATPIPE2 are inappropriate for such looped devices, FLUINT has been used extensively for modeling them.*

To change from the default unit set for HEATPIPE or HEATPIPE2 inputs, a units utility (HPUNITS) is provided.

The heat pipe simulation routines do not presume a flow direction: heat can be added in the middle of the pipe and rejected at both ends, for example, and the locations of heat acquisition and rejection can change during the analysis. In other words, there is no designation of which segments condense and which evaporate. However, any gas that is present is permitted only on one end or the other of the device: it is cannot be located in the middle of the pipe, nor at both ends simultaneously.

* Contact CRTech for more information.

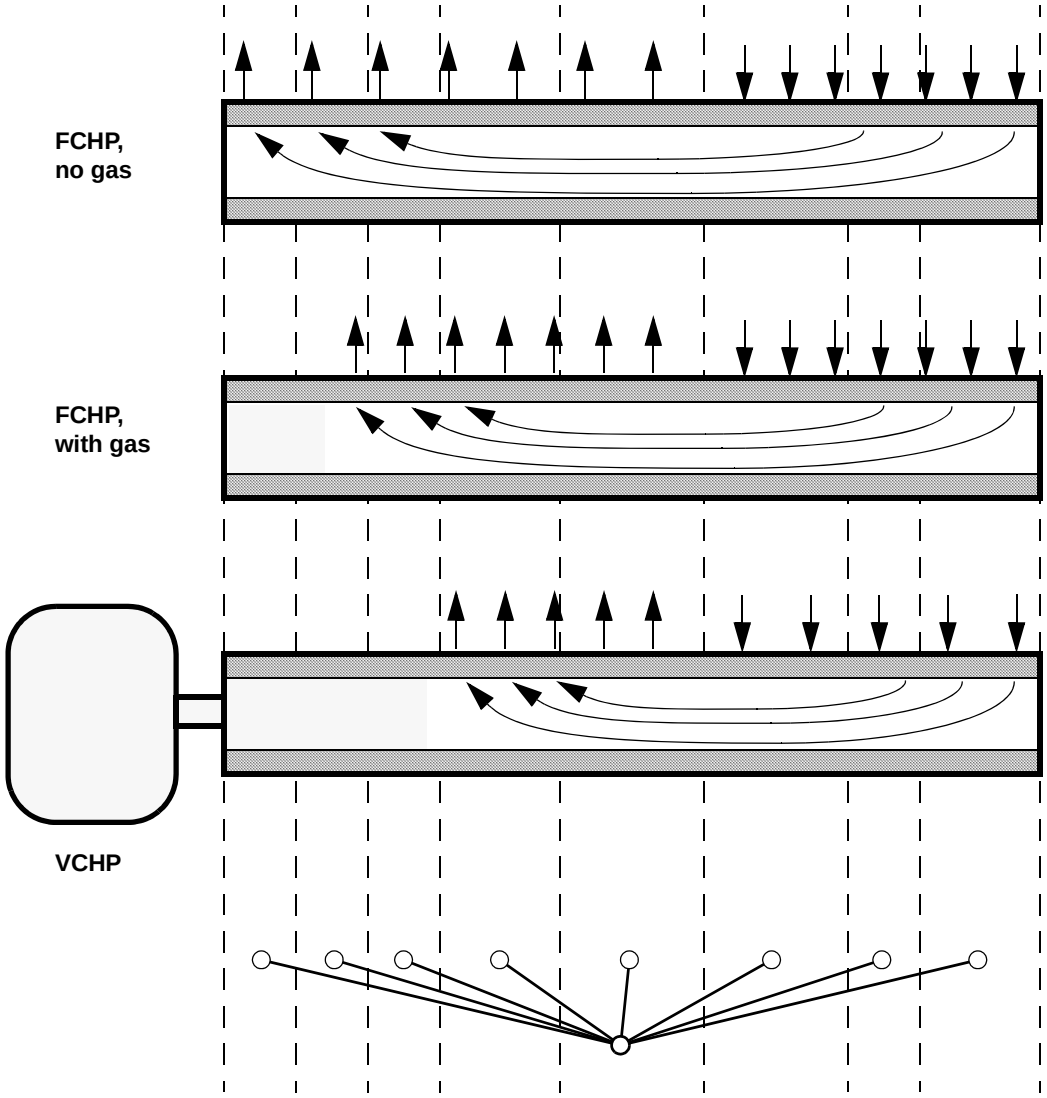


Figure 7-2 Heat Pipe Variations and Typical Network Diagram

Phenomena neglected by these routines include:

1. Wicking limits (including tilt effects), boiling limits, entrainment limits, sonic limits, viscous limits, etc. The exclusion of these limits eliminate any modeling of dry-out or recovery. These routines *do*, however, report the current power-length product (QL_{eff}), and this value can be compared with the pipe's reported capacity at the current temperature, tilt, etc. The power-length value can be used either for sizing heat pipes in preliminary designs, or for verifying adequacy and margin in more mature designs.

2. Freezing or thawing of working fluid.

3. Liquid pooling (condenser blockage due to excess charge).

4. Liquid-trap diode action. Gas-blockage of a VCHP evaporator, however, *is* simulated. To simulate liquid-trap diode action, the user can detect heat flow reversals and disable the pipe (by supplying zero film coefficients) by supplying extra logic. This logic should include criteria for restarting the pipe under favorable temperature gradients.

5. Diffusion, natural convection within the pipe, and dissolution of gases.

6. Radial or axial conduction within the pipe scope of this model terminates at the inner diameter of the pipe (vapor core diameter), and can be added by the user in their thermal network. Circumferential conduction is neglected by HEATPIPE, but may be included using HEATPIPE2 (along with a 2D thermal model of the pipe itself).

In addition, these routines makes the following assumptions:

1. At any instance, each segment is either condensing or evaporating, but not both: no apportionment is used for transitional sections. In HEATPIPE, a “segment” means an axial section. In HEATPIPE2, a “segment” means an axial and circumferential section: the top part of the pipe might be evaporating while the underside might be condensing.

2. In HEATPIPE, each section is one-dimensional: the model assumes axisymmetry. (HEATPIPE2 allows this assumption to be eliminated.)

3. Thermodynamic and hydrodynamic equilibrium is assumed at all times.

4. Gas, if present, is modeled as a flat front that blocks condensation. Blockage is linear, with no constriction resistances etc. per segment. For example, if gas extends 10% into a segment, then the condensation (but not evaporation) in that segment is reduced to 90% of the full value. For a completely blocked segment, a very small but nonzero conductance applies to avoid zero sum of conductors. Not that the flat front assumption applies to both HEATPIPE and HEATPIPE2: the circumferential subdivision in HEATPIPE2 does not mean that a more geometrically complex blockage is assumed.

5. Gas, if any, can only exist at one end of the pipe or the other. It may shift during the simulation, but cannot reside in the middle of the pipe.

6. If a reservoir exists (VCHP), then if working fluid vapor enters into it, a failure mode is assumed. (This treatment precludes use of a reservoir without NCG.) If vapor enters the reservoir, then either the reservoir size should be reduced, and/or the gas charge increased, and/or the temperature of the reservoir increased.

7. If the reservoir is hotter than the rest of the pipe, it is assumed to contain superheated vapor. In this case, the gas must then be contained within the pipe itself.

Note that the routine does *not* assume that noncondensible gases are perfect: real gases may be used as well.

Subroutine Name: HEATPIPE

Description: This subroutine is used to simulate either a constant (fixed) conductance or variable conductance heat pipe. It should be called from Variables 1. Multiple calls (with unique node and conductor inputs) may be made if multiple heat pipes are present.

The subroutine updates the temperature of an arithmetic node representing the vapor temperature. This node is connected to each axial segment via a linear conductor whose conductance value, G, is also updated by the routine as needed to represent vaporization or condensation. These conductors and the arithmetic node are depicted by the thickened network elements at the bottom of Figure 7-2. The nodes representing the wall segments (the thinner circles at the bottom of Figure 7-2) may be any type of node.

The series of conductors representing vaporization and condensation must be added in CONDUCTOR DATA and supplied to the routine by name using an integer SINDA array. Since their G value will be overwritten, a fake value such as “1.0” can be input in CONDUCTOR DATA. These conductors must be input in their geometric order, starting from the reservoir end of the pipe. If no reservoir is present, either end can be used as the starting point as long as the list is in order.

A pipe section length must be associated with each of these conductors. These lengths are supplied to the routine as another SINDA array, this time containing real instead of integer values. The lengths can be unequal, but the ordering of them must correspond to the ordering of the conductor IDs in the previously described name array.

Guidance *Units:* One of two sets of units can be selected via the choice of UID supplied in USER DATA, GLOBAL (Section 4.3). The *default* units for the HEATPIPE parameters are summarized in the right-most columns of Table 7-2. To override these units, use a call to HPUNITS (described below). In addition to these unit choices, the expected value of Tres is expected to be the same as nodal units, per the ABSZRO control constant (Section 4.3).

Table 7-2 Summary of HEATPIPE Parameters and Their Units

Parameter	Description	FCHP?	FCHP w/ gas?	VCHP?	Def. Units UID=ENG	Def. Units UID=SI
smn	name of thermal submodel	Yes	Yes	Yes		
NODE	ID of vapor arithmetic node	Yes	Yes	Yes		
NAc	Array list of conductor IDs	Yes	Yes	Yes		
Al	Array list of corresponding lengths	Yes	Yes	Yes	ft	m
Diam	Pipe vapor core diameter	Yes	Yes	Yes	ft	m
Hcond	Film coefficient when condensing	Yes	Yes	Yes	BTU/hr-ft^2-F	W/m^2-K
Hevap	Film coefficient when evaporating	Yes	Yes	Yes	BTU/hr-ft^2-F	W/m^2-K
Vres	Volume of VCHP reservoir	No	No	Yes	ft^3	m^3
Tres	Temperature of VCHP reservoir	No	No	Yes	R or °F	K or °C
EMgas	Mass of gas (NCG) present	No	Yes	Yes	lb_m	kg
IDwork	ID of working fluid	No	Yes	Yes		
IDgas	ID of gas	No	Yes	Yes		
QLpipe	returned power-length product	Yes	Yes	Yes	BTU-ft/hr	W/m

Guidance *Fluid Properties:* If noncondensible gas (NCG) blockage is to be modeled, then the code requires the ability to calculate both the saturation pressure of the working fluid and the specific volume of the noncondensible gas. To allow cases such as nontrivial saturation temperature-pressure relationships for the working fluids, as well as real gases with complicated PVT surfaces, and to exploit existing fluid property data, these fluid properties are input not by coefficients but rather by indirect reference: by identifying the fluid as one would if using a fluid submodel.

Many working fluids are either built into the code (e.g., specify "717" to mean "ammonia" or see Appendix C.1 for more options) or are available as FPROP DATA blocks (see Section 3.21, or contact CRTech for more options). Similarly any fluid, whether condensible or not, can be used as the "noncondensible" gas provided that vapor exists for the range of temperatures and pressures that result.

However, if no built-in fluids or suitable FPROP blocks are available, a new 8000 series FPROP DATA block can be quickly created to satisfy the requirements of either the working fluid (IDwork) or gas (IDgas), per Section 3.21.4. Note that very few of the property values listed in Section 3.21.4 are normally required, and that most of the ones that *are* required for an 8000 series fluid are not needed by the HEATPIPE routine and can be supplied fake values such as "1."

To create a new HEATPIPE working fluid, attention must be paid only to the TCRIT, PCRIT, and "AT PSAT" inputs. From these two temperature-pressure points, the code will construct an exponential curve fit for the saturation pressure of the gas.

To create a new HEATPIPE gas, attention must be paid only to the MOLW input (or, alternatively, the RGAS input).

Thus, with five data values commonly found in handbooks, adequate fluid descriptions can be created. Refer to the examples at the end of this section.

Note that working fluid properties, as defined in FPROP blocks, need not follow the unit convention of the master model provided that the local set of units for that fluid description has been defined with in the HEADER FPROP DATA subblock.

Guidance *Modeling a FCHP without gas:* In this simple case, many inputs to the routine are unnecessary and can be input as zeros (0.0 or 0 depending on the type): the volume and temperature of the reservoir (Vres and Tres, respectively), the amount of gas (EMgas), and the fluid property identifiers (IDwork, IDgas). These parameters are identified as unnecessary in the "FCHP?" column of Table 7-2. The only variation that can occur in such a heat pipe is a result of any differences between Hcond and Hevap.

Guidance *Modeling a FCHP with gas:* To model a FCHP with degradation due to generated gas, the amount of gas (EMgas) and fluid property identifiers (IDwork, IDgas) must be supplied, but the reservoir volume and temperature can be input as zero (0.0), per the “FCHP w/gas?” column of Table 7-2.

Gas will exist in the coldest end of the pipe, and will partially or completely block condensation in that end, decreasing the overall conductance of the heat pipe. The gas can switch ends if temperatures reverse. Otherwise, the gas volume will grow or shrink depending on the pressure in the pipe (the saturation pressure corresponding to the vapor node temperature) and the temperature of the cold wall in the gas-blocked sections. The partial pressure of gas is calculated as the total pressure minus the saturation temperature of the wall nodes, which are assumed to represent the temperature of the liquid in that region. The user is cautioned that strong temperature gradients may exist in the condensing end of the pipe, perhaps requiring additional resolution (more nodes with smaller segment lengths).

Guidance *Modeling a VCHP:* To model a VCHP employing a gas-charged reservoir, all inputs must be supplied per the “VCHP?” column of Table 7-2, including the volume and temperature of the reservoir. The conductor IDs (integers supplied in array NAc) and the lengths corresponding to these segments (real values supplied in array Al) must be arranged starting at the reservoir end.

Gas will be positioned at the coldest end of the pipe, whether or not the reservoir is located there. Normally, the reservoir is located at the coldest end of the pipe. Unless the reservoir is hotter than the rest of the pipe, gas will exist in the coldest end and will partially or completely block condensation in that end, decreasing the overall conductance of the heat pipe. The gas volume will grow or shrink depending on the pressure in the pipe (the saturation pressure corresponding to the vapor node temperature) and the temperature of the cold wall in the gas-blocked sections. The partial pressure of gas is calculated as the total pressure minus the saturation temperature of the wall nodes, which are assumed to represent the temperature of the liquid in that region. The user is cautioned that strong temperature gradients may exist in the condensing end of the pipe, perhaps requiring additional axial resolution (more nodes with smaller segment lengths).

If there is insufficient gas charge, or if the volume of the reservoir is too big, or if the reservoir becomes too cold, then there will be no gas in the pipe itself: all will be contained in the reservoir. If this happens, the vapor will flow into the reservoir and condense, perhaps overloading any wicking structure intended to return liquid to the pipe. This is considered a failure mechanism, and will generate either a caution in steady state solutions, or an abort in transient solutions. The code does not abort in a steady state solution because the condition may be a temporary result and not relevant

to the final answer. The user is therefore warned to check for cautions, and to assure that the final answer in a steady state case is valid if no subsequent transient is performed. To avoid this abort, signal tolerance of this condition by specifying a negative vapor node ID.

Guidance *Nonconvergence and Transient Oscillations.* When gas is present in the heat pipe, difficulties might arise with steady state convergence and oscillations might occur in transients, causing small time steps. The HEATPIPE and HEATPIPE2 routines necessarily assume that the gas front location is constant over each interval (where "interval means either a steady state relaxation step or a transient time step). In some cases, the gas front artificially oscillates between two or more positions. If this numerical instability is experienced, consider an additional assumption: constant saturation temperature over each interval. This assumption can be added simply by bracketing each HEATPIPE or HEATPIPE2 call by the HTRNOD/RELNOD utilities. For example:

```
 CALL RELNOD('PANEL',100)      $ release vapor node for HEATPIPE
 CALL HEATPIPE('PANEL',100,na101,a102,0.013,Ucond,Uevap
+      ,VolRes,TTEST,gmass,8717,8702,Qtest)
 CALL HTRNOD('PANEL',100)      $ hold constant for rest of network
```

Caution must be applied in transients, however, since the above assumption does not fully conserve energy: a little heat will be gained or lost with each time step if the above approach is applied within a transient solution. Check the Q value of the vapor node (perhaps after an HNQCAL call) to make sure that the magnitude of this heat leak is acceptably small. If it isn't, then a more rigorous but computationally expensive option is to use the NVARB1=1 option to solve fully implicitly in TRANSIENT ("FWDBCK") for the gas front location.

Guidance The FloCAD® option of Thermal Desktop® (see page xliv of the preface) automatically generates HEATPIPE, HEATPIPE2, and HPUNITS calls given user inputs and drawing information. Thermal Desktop users who do *not* use FloCAD to generate the heatpipe calls (e.g., "hand-generated" heat pipes within Thermal Desktop) should be advised that such logic block inputs are not visible to Thermal Desktop, and that extra care should be taken to assure that the global model units are as expected, and that appropriate HPUNITS calls are added as needed.

Guidance If there are strong circumferential gradients, perhaps due to insertion within panels or asymmetric heating or saddles, consider instead the HEATPIPE2 routine (below). HEATPIPE2 should also be considered for noncircular pipes and vapor chamber fins.

Guidance The HPGLOC utility (below) can be used following a HEATPIPE or HEATPIPE2 call to return the gas front location.

Guidance As HEATPIPE and HEATPIPE2 require internal temperature iterations, if there are any temperature-dependent sources on the wall nodes, it may be necessary to call QVTEMP in VARIABLES 1 *before* calling HEATPIPE or HEATPIPE2. Any other updates to Qs on heatpipe nodes should also be performed before the simulation routine is called.

Restriction No single heat pipe can be subdivided into less than 2 nor more than 1000 sections.

Restriction For VCHP modeling, the gas must extend into the pipe itself and cannot be completely contained within the reservoir. This condition will generate a caution during steady states and an abort during transients. See “Modeling a VCHP” above. To avoid this abort, signal tolerance of this condition by specifying a negative vapor node ID.

Restriction Multiple node pairs on not allowed on conductors used for heat pipes. Such pairs are by default illegal, and are only allowed with the NODEPAIR option in OPTIONS DATA.

Calling Sequence:

```
      CALL HEATPIPE('SMN',NODE,NAc,Al,Diam,Hcond,Hevap
     +          ,Vres,Tres,EMgas,IDwork,IDgas,QLpipe)
```

where:

`SMN` ..... thermal model name containing the node and the listed conductors, in single quotes

`NODE` ..... node number for the vapor node, which will also double as the identifier of the heat pipe in any messages. The vapor node must be an arithmetic node. A negative NODE input signals the program to not abort during transients if there is insufficient NCG in a VCHP.

`NAc` ..... reference to a SINDA/FLUINT integer array (NAn or smn.NAn) containing a list of *integer* identifiers of *linear* conductors. The conductors themselves must be in submodel SMN, although the array itself need not be input in that submodel. These conductors must connect the input NODE to the walls of the heat pipe, and must be input in order starting from the reservoir (if any). Up to 1000 segments may be input for any heat pipe. The G values of these conductors will be adjusted by the HEATPIPE routine.

`Al` ..... reference to a SINDA/FLUINT array (An or smn.An) containing a list of *real (floating point)* segment lengths, corresponding to the conductor number in the above array. For example, the third entry in this array defines the axial distance covered (corresponding to) the third linear conductor in the above list. The default units are either feet or meters.

`Diam` ..... heat pipe diameter (vapor core diameter). This dimension will be used both for heat transfer surface area and for gas/vapor volume calculations. The default units are either feet or meters (not inches, centimeters, etc.!).

`Hcond` ..... heat transfer coefficient to be applied to unblocked condensing sections. The default units are either W/m^2-K or BTU/hr-ft^2-R.

`Hevap` ..... heat transfer coefficient to be applied to evaporating sections (which cannot be blocked by gas). The default units are either W/m^2-K or BTU/hr-ft^2-R.

`Vres` ..... the volume of the VCHP reservoir, if any. To model an FCHP, input zero (0.0) volume. If Vres is positive, then the inputs Tres, EMgas, IDwork, and IDgas must also be supplied. The default units are either m^3 or ft^3.

`Tres` ..... the current temperature of the VCHP reservoir, if any. This input is ignored if Vres is zero (FCHP). The units are standard user temperature units (depending on the values of UID and ABSZRO): R, K, °F, or °C, so this input may simply be a reference to the temperature of a SINDA node representing the reservoir, but might also be an averaged value, register, etc.

`EMgas` ..... the mass of noncondensible gas within the pipe. Input zero (0.0) for an FCHP that lacks gas. If EMgas is zero, Vres should be zero too. If EMgas is not zero, both the working fluid (IDwork) and the gas (IDgas) must be specified. The default units are either kg or lb_m: *this a usually very small number!* To convert from kmol into kg or lbmol into lb_m, multiply by the molecular weight of the noncondensible gas. See also HPUNITS below.

`IDwork` ..... the integer identifier of the volatile working fluid, required if EMgas is positive. This might be a library fluid (Appendix C.1) such as “717” for ammonia or “290” for propane. Otherwise, input or INSERT an FPROP BLOCK for a user fluid (Appendix C.2, Section 3.21). Because working fluids must be volatile, 9000 series liquids are not acceptable. A 7000 series (Section 3.21.6) or 6000 series (Section 3.21.7) two-phase fluid can be used as an alternate input if available.

If no appropriate fluids are available, the minimum approach is to specify a 8000 series (Section 3.21.4) gas, but include the critical points (TCRIT, PCRIT) and the AT,PSAT subblock such that saturation pressures can be estimated. Other properties such as energy and transport properties (RGAS, MOLW, CP, K, V, etc.) are not needed by this routine, but are required for an 8000 series fluid, and so fake values (“1.0”) can be supplied if necessary per examples below.

Otherwise, contact CRTech for available fluid property descriptions.

`IDgas` ..... the integer identifier of the noncondensible gas (NCG), required if EMgas is positive. Although this will normally be an 8000 series (Section 3.21.4) gas, any other FPROP BLOCK or even library fluid might also be specified as long as vapor exists over the analyzed ranges of temperatures and pressures.

If no appropriate fluids are available, the minimum approach is to specify a 8000 series (Section 3.21.4) gas, with RGAS or MOLW being the critical parameters for accuracy. Transport and energy properties (CP, K, V, etc.) are not needed by this routine, but are required for an 8000 series fluid, and so fake values (“1.0”) can be supplied if necessary per examples below.

Otherwise, contact CRTech for available fluid property descriptions.

`QLpipe` ..... the *returned* power-length product for the pipe, as currently operating. This

number can be used to compare with the heat transport capacity of the pipe at the current temperature (temperature of NODE), tilt, etc. The default units are W-m (*not Watt-inches!*) or BTU-ft/hr. See the Guidance section below for more details.

Example #1: FCHP (no reservoir), no gas, array and property inputs not shown:

```
      CALL HEATPIPE('PANEL',1000,na10,a20,Dpipe,10000.,5000.
     +      ,0.,0.,0.,0,0,Qtest)
```

Example #2: FCHP (no reservoir), with gas, array and property inputs not shown:

```
      CALL HEATPIPE('PANEL',1000,na10,a20,Dpipe,10000.,5000.
     +      ,0.,0.,1.0E-9,717,8740,Qtest)
```

Example #3: VCHP (both reservoir and gas), array and property inputs not shown:

```
      CALL HEATPIPE('PANEL',1000,na10,a20,Dpipe,10000.,5000.
     +      ,100.0E-6,T3000,5.0E-2/8.314E6,717,8740,Qtest)
```

Example #4: VCHP (both reservoir and gas), all relevant inputs shown:

```
Header Variables 1, Panel
      TTEST = 0.5*(T30 + T40) $ find average reservoir temperature
      CALL HEATPIPE('PANEL',100,na101,a102,0.013,Ucond,Uevap
     +      ,VolRes,TTEST,gmass,8717,8702,Qtest)
      if(qtest .gt. QLlimit/39.37)then $ limit is 12000 W-in
            write(nout,*)' heat pipe overloaded at time ',timen
            timend = timen           $ stop a transient run
      endif
Header Array Data, Panel
C     conductor names from reservoir out, 4 cond segments, 2 evap:
      101   = 10, 20, 30, 40, 110, 120
C           each of these conductors attaches to node PANEL.100
C     lengths corresponding to the above segments
      102   = Cleng/NumC,Cleng/NumC,Cleng/NumC,Cleng/NumC
              Eleng/NumE,Eleng/NumE
Header control data, global
      UID   = SI        $ SI units
      ABSZRO = -273.15  $ Degrees C
```

Guidance QL Product (*QLpipe*, or QL_{eff}). The calculated power-length product, *QLpipe*, allows a user to estimate how far the pipe currently is from its rated capacity at the current temperature and tilt, as determined by the internal limits (wicking, boiling, entrainment, sonic, or viscosity). Since such data ($QL_{eff,max}$ versus temperature and tilt) should be

available from the vendor, there is no need to check these limits explicitly. Assuming the radial heat flow through each (i^{th}) segment is Q_i, the QL_{eff} product is calculated by integration from one end of the pipe to the other, returning the maximum absolute power-length product encountered over each segment:

$$QL_{pipe} = \max_i | [\Sigma_i(Q_i/2 + \Sigma_{j=0,i-1} Q_j) \Delta L_i] |$$

```
Header Register Data
      Cleng = 1.0       $ Length of condenser, m
      NumC  = 4         $ resolution of condenser (for easy changes)
      Eleng = 0.25      $ Length of evaporator, m
      NumC  = 2         $ resolution of evaporator (for easy changes)
      VolRes = 1.0E-3   $ Volume of reservoir, m3
      gmass = 1.0e-6    $ NCG mass, kg
      Ucond = 10000.0   $ U (H) for condensation, W/m2-K
      Uevap = 5000.0    $ U (H) for vaporization, W/m2-K
      QLlimit = 12000.0 $ throughput limit, Watt-inches
C
Header fprop data, 8717, si, 0.0 $ “fake” “volatile” ammonia
c caution: for routine “heatpipe” only: dummy data used
      CP = 1.0, V = 1.0, K = 1.0 $ fake values to satisfy requirements
      TCRIT = 406.8           $ required for vapor pressure curve fit
      PCRIT = 11.6e6          $ required for vapor pressure curve fit
AT,PSAT, 300.0, 1.061E6       $ saturation at 300K for curve fit
C
Header fprop data, 8702, si, 0.0 $ “fake” H2 gas
c caution: for routine “heatpipe” only: dummy data used
      CP = 1.0, V = 1.0, K = 1.0 $ fake values to satisfy requirements
      MOLW  = 2.0159          $ required molecular weight for VSV
```

Subroutine Name: HPUNITS

Unlike FLUINT, SINDA itself does not enforce a unit convention: any units may be used as long as they are self-consistent (e.g., the units of linear G times the units of T should be equal to the units of Q). The same is largely true of the HEATPIPE and HEATPIPE2 routines: the units of the heat transfer coefficients time the units of the areas (or lengths squared) will be assumed to be the same as the units of G, and the QL product will be the units of Q times the units of this same input length, assuming consistent units were chosen by the user.

When gas is present, however, extra calculations are required that depend on a knowledge of the temperature units. Therefore, even if no fluid submodels are present, the UID must be set to SI if degrees C or K are used, and it must be set to ENG if degrees F or R are used. (UID is set in CONTROL DATA, GLOBAL.) ***UID must be correctly set when using nonzero gas quantities in either HEATPIPE or HEATPIPE2.***

When UID is set to SI, the assumed unit for length is meters and the assumed unit for mass is kilograms. When UID is set to ENG, the assumed unit for length is feet, and pounds for mass. To deviate from these assumptions, one or more calls to HPUNITS should be made prior to the HEATPIPE or HEATPIPE2 call:

```
CALL HPUNITS(name,units)
```

where:

name ..... the type of unit to specify: ‘mass’ or ‘length’, in single quotes
units ..... the units to use (in single quotes):
if name = ‘mass’, then select from ‘kg’, ‘gm’, ‘lb’ or ‘lbm’, ‘kg-mole’, ‘lb-mole’, or ‘gm-mole’
if name = ‘length’, then select from ‘m’, ‘cm’, ‘mm’, ‘ft’, or ‘in’

Example:

```
CALL HPUNITS('MASS','GM')
CALL HPUNITS('LENGTH','CM')
```

HPUNITS applies to the next HEATPIPE or HEATPIPE2 invocation, and can be called from anywhere. It could be called repeatedly if multiple heat pipes are present and each use different units. However, if only one heat pipe is employed or else consistent units are used for all heat pipes in the model, then HPUNITS should be called at the beginning of OPERATIONS.

Subroutine Name: HPGLOC

HPGLOC returns the gas front location from the previous HEATPIPE or HEATPIPE2 call:

```
CALL HPGLOC(dist)
```

where “dist” is a real variable that will be returned with the returned distance from the reservoir to the gas front in user length units.

If the returned value of “dist” is negative, then the pipe has reversed, and the distance from the other (nonreservoir) end of the heat pipe to the gas front is of length |dist|. A value of zero is returned if the previous HEATPIPE or HEATPIPE2 call had no gas.

VCHP Error Message Summary

When modeling a variable conductance heat pipe, there are several caution messages that may appear in the output file. When these occur, it is important that the user investigate the cause and determine if there is a problem with the VCHP design. In many cases it may be a transient condition which, depending on the duration, may or may not require a resolution.

**** CAUTION ** VCHP reservoir is warmer than saturation, and contains all vapor. Control authority has been lost for HEATPIPE: *xxx.nnn*. Check reservoir temp, or add more gas.**

The reservoir temperature is greater than the temperature of the vapor node (saturation). When this happens, there is no working fluid vapor in the reservoir and the gas front location can no longer be controlled by the reservoir.

SINDA/FLUINT will solve for a gas front location when this occurs, but the location will most likely be inaccurate. A hot reservoir is normally considered a failure condition in a VCHP.* For a steady state run, this error message often occurs during early iterations. If the message only occurs in early iterations and not during the final iterations, the message can be ignored. During a transient simulation, if the duration is significant, this may indicate a problem in the heat pipe design.

**** CAUTION ** More than one potential gas front location exists, exceeding simulation capabilities for HEATPIPE *xxx.nnn*.**

More than one gas front location has been detected by the code: there are at least two condensing zones that are separated from each other by evaporating zones. This situation can be encountered in a gas-loaded FCHP as well as in VCHPs.

When this condition is found, SINDA/FLUINT will calculate the gas front location as if the second front did not exist, blocking condensing segments from the *coldest* end of the pipe toward the opposite (perhaps condensing) end until all the gas has been allocated. The program does not block off the far end of the pipe (nor any other part that is colder than saturation) past the first evaporation zone that it encounters. Such remote condensing zones (for example, a cold spot in the nominal evaporator) are assumed to have no gas and to be acting like an open condenser. In other words, SINDA/FLUINT assumes that there is no diffusive or convective mechanism that could transport gas through the evaporation zone to the “wrong” side of the pipe.

The occurrence of this event is normally considered a failure mode for a VCHP. In this situation, the code can not properly determine the gas front locations since it cannot choose how much of the available gas should exist in each of multiple regions. If this message is encountered, inspect the temperature profile along the pipe to locate the hot spots and resolve the problem. Most likely the cause is external heating on the condenser causing a local hot spot, but it can also be caused by local

* Excepting perhaps an attempt to shut down the pipe in the reverse mode by purposely heating the reservoir to effect a diode action. This mode is not simulated by the HEATPIPE and HEATPIPE2 routines unless the reservoir is heated above the saturation condition of the pipe, and the condenser is warmer than the evaporator.

cooling on an evaporator. For a steady state run, this should be considered to a design flaw if it persists in the final iterations. For transient runs, assess the duration and time constant of the system to determine if the existence of multiple gas fronts is a problem.

**** CAUTION ** Illegal reservoir or gas sizing for HEATPIPE *xxx.nnn*. Use a smaller reservoir, warmer reservoir, or more gas.**

This message occurs when there is insufficient noncondensible gas in the system. As a minimum, the gas must always fill the reservoir and at least a small amount of gas must extend into the condenser. If there is insufficient gas in the reservoir (and therefore no gas in the rest of the heat pipe), then the working fluid will be pulled into the reservoir via condensation, potentially starving the evaporator of liquid. In steady state runs, this message can occur without issue during early iterations but should not be tolerated in the final iterations. If it does persist, this indicates a design flaw. For a transient run, this error will result in the run being aborted. A negative vapor node ID in the heat pipe call will prevent the run from aborting in this situation. In this case (with abort disabled), the pipe will be assumed to operate in a completely unblocked mode, but with zero condensation in the reservoir: evaporator starvation is not modeled.

The message is sometimes the result of improper gas or reservoir sizing, but is often caused by imposing an environment or heat load on the pipe which is beyond what it was sized to withstand. If the gas mass and reservoir volume has been supplied by a third party (perhaps a vendor or heat pipe designer), find out the minimum and maximum environments and heat loads that were used in the sizing, and then see if the pipe is exceeding this range. If so, resizing may be required. Also be sure to account for all volume in the VCHP including the transition joint between the reservoir and condenser. Again, early steady state iterations may produce this error message. If the message continues throughout a steady state or a transient, review the inputs. Sizing of the gas charge and reservoir volume is a non-trivial process and should be done by the heat pipe designer, but must be verified by system-level analyses.

**** CAUTION ** VCHP has entered diode mode with cool reservoir. Gas cannot be in two places at once for HEATPIPE *xxx.nnn*. Check too if FloCAD Pipe needs to be reversed.**

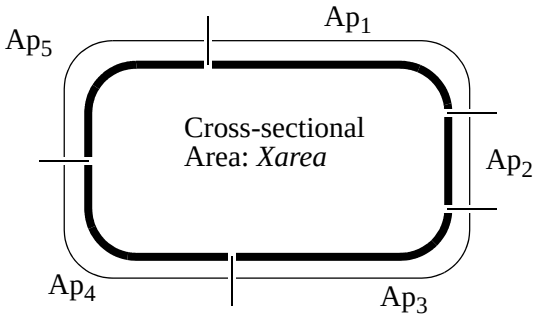
This message occurs when the pipe has reversed (condenser is warmer than the evaporator) and the reservoir temperature is below the vapor temperature of the pipe. This results in gas residing in two locations within the pipe (reservoir and evaporator), which is beyond the simulation capabilities of the HEATPIPE and HEATPIPE2 subroutines. In diode mode (with a condenser hotter than the evaporator), the reservoir must be heated above the vapor temperature of the pipe forcing all gas to the cold end of the pipe to obtain proper shutdown of the pipe.

Subroutine Name: HEATPIPE2

An alternate to HEATPIPE is available to handle circumferentially subdivided heat pipes, non-circular heat pipes, and vapor chamber fins: HEATPIPE2, as defined below:

```
  CALL HEATPIPE2('SMN',NODE,NAc,Ap,Al,Xarea,
+  ,Hcond,Hevap,Vres,Tres,EMgas,IDwork,IDgas,QLpipe)
```

A one-dimensional flat-front assumption still applies, but the user is able to use more than one node to represent an axial segment: the pipe may be subdivided circumferentially into an arbitrary number of nodes, each representing a given length along the periphery of the cross section. For example, the following flattened oval pipe has been divided into 5 sections around the circumference as shown at the right.



The arguments and usage of HEATPIPE2 are nearly identical to HEATPIPE, so the user is assumed to be familiar with application of the HEATPIPE routine. The following are ways in which HEATPIPE2 differs from HEATPIPE:

1. NAc represents the two-dimensional list of conductor numbers, entered as a SINDA integer array in ARRAY DATA. If the pipe has been subdivided into 10 axial segments and 5 circumferential segments, then this list has 10*5=50 entries.

   As with HEATPIPE, conductors should be entered starting with the first axial segment next to the reservoir (if any), with all N of the conductors in that first segment entered first (given a circumferential discretization level of N), followed by the N conductors in the second axial segment, and so forth until all M segments have been entered (assuming M axial segments total), for a total of N*M entries. If Ap is entered as "HP.A100", then in ARRAY DATA for submodel HP, the following should be entered, where NG_{ij} is the conductor number for the conductor in the i^{th} axial segment and the j^{th} circumferential section:

   HEADER ARRAY DATA, HP

100 =	NG_{11},	NG_{12},	NG_{13}, ...	NG_{1N}
	NG_{21},	NG_{22},	NG_{23}, ...	NG_{2N}
	...			
	NG_{M1},	NG_{M2},	NG_{M3}, ...	NG_{MN}

   Note that while this format is very similar to that of a bivariate array, it is purposely not identical: it is missing the initial row size N, since that value is the same as the size of the input array Ap (the next argument in the list).

2. Ap is the linear list of lengths representing the circumference around a typical pipe cross-section, as depicted in the figure above. The heat transfer area for segment i and circumferential section j is therefore given as Al_i*Ap_j. Note that there is an implicit assumption that the cross section of the heat pipe is constant, ***and also that its discretization at any one section is the same as all the others***.

3. Instead of *Diam* that is used in HEATPIPE, HEATPIPE2 requires the vapor cross sectional flow area *Xarea*. This value is used for determining the volume of the pipe, but its value does not affect heat transfer directly.

Guidance When modeling a noncircular heat pipe with no circumferential subdivision, simply use HEATPIPE2 as if it were a call to HEATPIPE, except enter a single value in the Ap array (in ARRAY DATA, not as the argument Ap): the total circumference. Also, note that HEATPIPE2 uses the cross sectional area (Xarea) in place of the vapor core diameter (Diam). For example, for a circular pipe with no circumferential subdivision, HEATPIPE2 will be identical to HEATPIPE if Xarea is input as π*$Diam^2/4$ and the sole member of the Ap array is input as π*Diam.

Caution Vaporization might occur on one side of the same axial segment in which condensation also occurs, such that some vapor (and therefore energy) flows across the pipe as well as along it. However, in such cases the calculated QL product will reflect only net heat flows through that segment, and, depending on the wick design and whether it permits liquid to flow circumferentially, the pipe might reach a wicking limit that is not reflected in the calculated QL product. Consult the heat pipe vendor for specific application details.

Also, the presence of 2D heat transfer (axial and circumferential) does not change the fact that a 1D flat front assumption is employed for calculating gas blockage. In other words, all condensing portions of a single axial segment are blocked equally in the presence of gas. Vaporization cannot be blocked by gas, and so one side of the pipe might be vaporizing when the opposite side is partially or completely blocked by gas from condensing. However, the existence of such 2D effects can render the 1D flat front assumption used in HEATPIPE2 invalid.

Guidance: HEATPIPE2 may also used to represent rectangular vapor chamber fins. However, in that case the 1D flat front assumption for gas is inappropriate: when modeling vapor chamber fins, *EMgas* should be zero, otherwise use extreme caution. Furthermore, HEATPIPE2 may be inappropriate for geometries more complex than rectangular vapor chamber fins. As an alternative, consider Thermal Desktop's (page xliii of the preface) node-to-surface conductors, which can connect arbitrary surfaces but which lacks the ability to make a distinction between *Hcond* and *Hevap*.

7.7.10 Thermoelectric Coolers (TECs) and other Peltier Devices

Routines are available to size and simulate thermoelectric (Peltier) devices. These include:

TECUNITS . . . a units selection utility for all routines in this section
BISTEL . . . . . returns temperature-dependent properties for TECs made with bismuth telluride, a common material for room temperature applications
TECINFO . . . . returns sizing information given material properties (whether output from BISTEL or from another source)
TEC1 . . . . . simulates a single-stage TEC as modeled by two nodes plus a linear conductor interconnecting them: a 1D model
TEC2 . . . . . simulates a single-stage TEC as modeled by multiple pairs of nodes each with a unique linear conductor interconnecting them: a 2D model

All of these routines require attention to units, and in particular to the UID selection (ENG or SI) in Global Control Data (Section 4.3). Temperatures are assumed in user units according to the values of UID and ABSZRO. Default length units are meters if UID=SI, and feet if UID=ENG, but different length scale units can be chosen using the TECUNITS utility. Default *thermal* power units are Watts if UID=SI, and BTU/hr if UID=ENG, and can also be changed using TECUNITS. Electrical power units are *always* Watts.

In particular, ***users of English (US Customary) units, UID=ENG, are cautioned to exercise care*** because of the mixture of electrical and thermal properties.

Subroutine Name: TECUNITS

Unlike FLUINT, SINDA itself does not enforce a unit convention: any units may be used as long as they are self-consistent (e.g., the units of linear G times the units of T should be equal to the units of Q).

However, extra calculations are required for TECs that depend on a knowledge of the units. Therefore, even if no fluid submodels are present, the UID must be set to SI if degrees C or K are used, and it must be set to ENG if degrees F or R are used. (UID is set in CONTROL DATA, GLOBAL.) ***UID must be correctly set when using TEC simulation routines.***

When UID is set to SI, the assumed unit for length is meters and the assumed unit for *thermal* power is Watts. When UID is set to ENG, the assumed unit for length is feet and the assumed unit for *thermal* power is BTU/hr. To deviate from these assumptions, one or more calls to TECUNITS should be made prior to the TEC1, TEC2, or BISTEL call:

```
CALL TECUNITS(name,units)
```

where:

name ..... the type of unit to specify: ‘length’ or ‘tpower’, in single quotes.
units ..... the units to use (in single quotes):
if name = ‘length’, then select from ‘m’, ‘cm’, ‘mm’, ‘ft’, or ‘in’
if name = ‘tpower’, then select from ‘W’, ‘kW’, ‘MegaW’, ‘milliW’, ‘JMIN’ (Joule/min), ‘JHR’ (Joule/hour), ‘BTUHR’ (BTU/hr), ‘BTUMIN’ (BTU/min), or ‘BTUSEC’ (BTU/sec)

Example:

```
CALL TECUNITS('LENGTH','CM')
CALL TECUNITS('TPOWER','BTUMIN')
```

TECUNITS applies to the next TEC1, TEC2, TECINFO, or BISTEL invocation, and can be called from anywhere. It could be called repeatedly if multiple TEC devices are present and each use different units. However, if only one thermoelectric device is employed or else consistent units are used for all such devices in the model, then TECUNITS should be called at the beginning of OPERATIONS.

Note that electrical power is *always* in units of Watts.

Subroutine Name: BISTEL

The routine BISTEL may be used to estimate relevant overall thermal and electrical properties for bismuth telluride couples. These properties (Seebeck coefficient, electrical resistivity, and thermal conductivity) are based on second-order polynomial curve fits.* Although the valid range for these properties was not specified in the original source, BISTEL restricts the range of -100°C to 60°C: average temperatures (where Tavg = 0.5*(Tc+Th)) outside of that range will yield property values at the limits.

The resulting values have been multiplied by 2*Ncp (where Ncp is the number of couples), and may therefore be used directly in the TECINFO, TEC1, or TEC2 routines. (Divide the results of BISTEL by 2*Ncp if the raw material properties are needed.) However, in the case of the TEC1 and TEC2 routines, when using bismuth telluride devices it is best to specify Ncp (the number of couples) directly in those routines since this mode enables them to call BISTEL internally, thereby taking into account temperature variations implicitly within its internal iterations. In other words, *a separate call to BISTEL is neither needed nor recommended when using TEC1 and TEC2.*

Calling sequence:

```
CALL BISTEL (Ncp, Tc, Th, Sm, Rm, Km)
```

* Published by Melcor (www.melcor.com). Ferrotec America (www.ferrotec-america.com) is another good source of property information and sizing relationships. *Each vendor's material properties will differ.*

where:

Ncp. . . . . .Input number of couples (integer*)
Tc . . . . . .Input cold side temperature, user units
Th. . . . . .Input hot side temperature, user units
Sm . . . . . .Returned *effective* Seebeck coefficient (material property times 2*Ncp), V/K or V/R depending on UID.
Rm . . . . . .Returned *effective* resistivity (material property times 2*Ncp), ohm-*length* where *length* is either specified by a TECUNITS call or defaults to meters (UID=SI) or feet (UID=ENG).
Km . . . . . .Returned *effective* conductivity (material property times 2*Ncp), in *tpower*/*length*-degree units depending on the value of UID and any calls to TECUNITS: W/m-K (UID=SI) or BTU/hr-ft-F (UID=ENG) are used by default if no calls to TECUNITS have been made.

Example:

```
CALL BISTEL(71, mymod.T101, mymod.T201, STEST, RTEST, CTEST)
```

Subroutine Name: TECINFO

Given the device aspect ratio (X, the ratio of cross sectional area to thickness for one couple) and, perhaps using a prior call to BISTEL, the overall (effective) device properties Seebeck coefficient (S_m), material electric resistivity (R_m), and thermal conductivity (K_m), TECINFO produces sizing metrics regarding thermal electric cooler performance at the currently specified hot and cold side temperatures. These metrics may be used to select a specific device, the simulation of which can be later performed using the TEC1 routine.

For example, the “maximum current” (the current corresponding to the maximum cold production or maximum temperature gradient) is given by

$$I_{max} = (K_m*X/S_m)\,[(1 + 2*S_m^2*T_h/(R_m*K_m))^{1/2} - 1]$$

for S in V/K, T_h in degrees Kelvin and K_m in W/length-K where “length” is the units used in X. The “optimum current” (the current corresponding to the maximum coefficient of performance, COP = Qc/(V*I), when the device is used for cooling) is given by:

$$I_{opt} = [K_m*X*(T_h-T_c)*(1 + (1 + S_m^2*T_{avg}/(R_m*K_m))^{1/2}]/(S_m*T_{avg})$$

where T_{avg} is the average temperature (in degrees Kelvin): $T_{avg} = 0.5*(T_h+T_c)$

* If a register is used to define this value, be sure to declare it as an integer using the “INT:” prefix in REGISTER DATA.

Note that the values returned by TECINFO are functions of the input hot and cold side temperatures. For the purposes of selecting or sizing a device, several calls to TECINFO (and perhaps BISTEL) may be required at different temperature corresponding to various design cases. Note that S_m, R_m, and K_m are all temperature dependent: for bismuth telluride devices, additional calls to BISTEL will also be needed.

Calling sequence:

```
      CALL TECINFO (Tc, Th, X, Sm, Rm, Km, DTmax,
     .          Imax, Vmax, QcMax, Iopt, Vopt, QcOpt, COPopt)
```

where:

Tc . . . . . Input cold side temperature, user units

Th . . . . . Input hot side temperature, user units

X . . . . . Aspect ratio of one couple: area/thickness, *in length units that are consistent with Rm and Km below and with any prior calls to TECUNITS*. Use X=1.0 if the input Rm and Km values are applied to the whole device (e.g., they have units of ohm and *tpower*/degree respectively). Melcor refers to this parameter as "G" and provides it in units of centimeters, which might need to be converted to meters for TECINFO: X = 0.01*G if Rm is in ohm-m and Km is in W/m-K (the default if UID=SI and no TECUNITS calls have been made).
For example, the total thermal conductance across the device is Km*X, and the total electrical resistance of the device is Rm/X.

Sm. . . . . Input *effective* Seebeck coefficient (material property times 2*Ncp), V/K or V/R depending on UID.

Rm . . . . . Input *effective* resistivity (material property times 2*Ncp), ohm-length where length units depend on the value of UID and any calls to TECUNITS. The default is ohm-m for UID=SI and ohm-ft for UID=ENG.

Km . . . . . Input *effective* conductivity (material property times 2*Ncp), *tpower*/*length*-degree where thermal power and length units depend on the value of UID and any calls to TECUNITS. The default is W/m-K for UID=SI and BTU/hr-ft-R for UID=ENG. This is a real (not integer) value.

DTmax . . . . . Returned maximum temperature lift (at zero cold load), in user units (note that this is *not* the temperature difference at Imax, Vmax)

Imax . . . . . Returned current at maximum net cold production (fixed Th), amps

Vmax . . . . . Returned voltage at maximum net cold production (fixed Th), volts

QcMax . . . . . Returned maximum cold production at Imax and Vmax, in units of *tpower* (this may be less than the maximum cold production possible if Tc equaled Th)

Iopt . . . . . Returned optimum current (maximum COP for cold production), amps

Vopt. . . . . Returned optimum voltage (maximum COP for cold production), volts

QcOpt . . . . . Returned net cold production at Iopt and Vopt, in units of *tpower*

COPopt . . . . . Returned optimum coefficient of performance at Iopt and Vopt

Example:

```
      CALL BISTEL(71, mymod.T101, mymod.T201, STEST, RTEST, CTEST)
      CALL TECINFO(mymod.T101, mymod.T201, 0.0045, STEST, RTEST, CTEST,
     .     Dtmax, Cmax, Vmax, Qnetmax, Copt, Vopt, Qnetopt, COPopt)
```

Guidance Note that the QcMax returned by TECINFO is *not* the same as the nodal Q value applied by TEC1 or TEC2, the thermoelectric cooler simulation utilities. In all of its calculations, TECINFO takes into account thermal conduction through the device, whereas in TEC1 and TEC2 that term is accounted for separately via a SINDA linear conductor. In other words, the magnitude of QcMax will be less than the magnitude of the nodal sink term, |Q(Nc)|, that TEC1 and TEC2 apply for the same temperatures, current, voltage, etc.

Note also that QcMax is not the maximum possible at zero temperature gradient, since Tc and Th may not be equal. Rather, it is the cold production at the current Tc and Th given the maximum current and voltage.

Subroutine Name: TEC1 (one dimensional)

Once a single stage thermoelectric device has been selected, perhaps using the TECINFO routine, the TEC1 utility may be used to simulate it in SINDA.

One node is used to represent the hot side, and another node is used to represent the cold side. These nodes may be of any type (arithmetic, diffusion, boundary). In addition, a linear conductor between these two nodes is used to represent the thermal conduction through the device.* To use TEC1, build this simple network (two nodes and a linear conductor between them) as part of the SINDA model, then call TEC1 within VARIABLES 1, as specified below.

Given the device aspect ratio (X, the ratio of cross sectional area to thickness† for one couple) and *either* the number of bismuth telluride couples (N_{cp}) *or*, for other materials, the overall device properties Seebeck coefficient (S_m), material electric resistivity (R_m), and thermal conductivity (K_m), TEC1 applies the appropriate Q source terms to the nodes to simulate the TEC device. It may internally adjust the temperatures of these nodes in steady states (or in transients if they are arithmetic) as needed.

Usually, the ID of the linear conductor that represents solid conduction through the TEC is also supplied. If this ID (“Ng”) is supplied, then TEC1 also updates the conductance G of that conductor to represent the conduction through the couples themselves. Additional conductors representing other materials or heat transfer paths may be added separately in parallel.

---

* This conduction represents the conduction through the couples themselves, and not any potting compound or other material, which can be added separately (in parallel).

† X has default units of either meters (UID=SI) or feet (UID=ENG), or as specified by the latest TECUNITS call

The TEC electrical state may be specified by inputting one of the following: current, voltage, or power (see “Mode” below). Whichever two values are not specified are calculated by the routine.

Calling sequence:

```
      CALL TEC1 ('smn', Nc, Nh, Ng, Mode, I, V, P,
     .          X, Ncp, Sm, Rm, Km)
```

where:

smn . . . . . thermal submodel containing the nodes (and perhaps the conductor), single quotes

Nc . . . . . ID of cold side node in submodel smn

Nh . . . . . ID of hot side node in submodel smn

Ng . . . . . ID of linear conductor between Nc and Nh representing the back conduction through the device. If Ng is zero, no conductor is updated. Otherwise, the model units are assumed to be *tpower*, *length* depending on UID and any prior TECUNITS calls. If TECUNITS has not been called, then if UID=SI the units are Watts and meters, otherwise if UID=ENG the units are BTU/hr and feet. If these are NOT the correct units, input Ng as zero and update the conductor separately, perhaps using the value of Km returned by either this routine or BISTEL.

Mode. . . . .

Value	Input Parameter	Returned Parameters
1	current (I)	voltage (V) and power (P)
2	voltage (V)	current (I) and power (P)
3	power (P)	current (I) and voltage (V)

A negative sign on “Mode” is used as a flag to signal the use of a “shallow” node update instead of a default “deep” update. See Guidance notes below.

I. . . . . Input current (amps) if Mode = 1, else the returned current. The current is positive for the normal (cooling or heat pumping) direction. Note that this is a real, not integer, argument.

V . . . . . Input voltage (volts) if Mode = 2, else the returned voltage. The voltage is positive for the normal (cooling or heat pumping) direction.

P . . . . . Input electrical power (*always Watts*) if Mode = 3, else the returned power. In P is input, it must be zero or positive. Positive current and/or voltage is assumed.

X . . . . . Aspect ratio of one couple: area/thickness.* Use X=1.0 if the Rm and Km values are applied to the whole device (e.g., they have units of ohm and *tpower*/degree respectively). Melcor refers to this parameter as “G” and provides it in units of centimeters, which must be converted to the correct length units for TEC1: X = 0.01*G if Rm is in ohm-m and Km is in W/m-K (the default if UID=SI and no TECUNITS calls have been made). For example the total thermal conductance across the device is Km*X, and the total electrical resistance of the device is Rm/X.

* X has default units of either meters (UID=SI) or feet (UID=ENG), or as specified by the latest TECUNITS call.

Ncp. . . . . Input number of couples (integer*). If zero, the user must input Sm, Rm, Km. Otherwise, a positive value is taken to mean the number of Bismuth Telluride couples, in which case Sm, Rm, and Km are calculated automatically and become output parameters

Sm . . . . . Input (Ncp = 0) or Output (Ncp > 0) *effective* Seebeck coefficient (material property times 2*Ncp), V/K or V/R depending on UID.

Rm . . . . . Input (Ncp = 0) or Output (Ncp > 0) *effective* resistivity (material property times 2*Ncp), in units of ohm-length where length units depend on the value of UID and any calls to TECUNITS. The default is ohm-m for UID=SI and ohm-ft for UID=ENG.

Km . . . . . Input (Ncp = 0) or Output (Ncp > 0) *effective* conductivity (material property times 2*Ncp), in units of *tpower*/*length*-degree where thermal power and length units depend on the value of UID and any calls to TECUNITS. The default is W/m-K for UID=SI and BTU/hr-ft-R for UID=ENG. This is a real (not integer) value.

Example:

```
Header variables 1, mymod
      call qvtemp('mymod')   $ Just in case nearby nodes are variable
      CALL TEC1('mymod', 101, 202, 101201, -1, Current, Volt, Power,
     .   0.0045, 71, STEST, RTEST, CTEST)
```

Caution Because TEC1 updates nodal temperatures for stability, any other source terms applied to these nodes or to attached nodes should have been updated before TEC1 is called. This may be accomplished by inserting a call to QVTEMP before any TEC1 calls, as shown in the example above.

Guidance Because TEC1 updates nodal temperatures as part of its solution, and because node solutions are sensitive to the sequence of Q updates (see above Caution), if the TEC nodes are adjacent to nodes with Qs that are updated late in the sequence (including heaters and other TEC stages ... see below), then the default "deep" solution may become unstable and need to be replaced with a less aggressive solution that only updates the TEC nodes themselves ("shallow" solution). The shallow solution is signaled via the use of a negative sign on the Mode variable. The upshot: if convergence or stability issues are encountered, try negative Mode values.

Caution TEC1 is meant to simulate single stage devices only. Cascaded (multiple stage) devices may be modeled using multiple TEC1 calls (one per stage), but in that case *the user is strongly advised to use separate nodes between the stages* (e.g., to separate the hot side of one stage from the cold side of the next stage, perhaps with a conductance representing the bond). Also, use negative Mode values (shallow solutions) to keep the order in which nodal Qs are updated from disrupting convergence.

* If a register is used to define this value, be sure to declare it as an integer using the "INT:" prefix in REGISTER DATA.

Guidance For simulating a controller for a TEC, the COOLCON, and PCOOLCON routines (Section 7.7.4) may be used with current or voltage or power as the control parameter. Steady-state solutions may oscillate without the independent use of HTRNOD on one or both TEC nodes.

PID controller simulation routines are also available (Section 7.7.8).

Subroutine Name: TEC2 (two dimensional)

TEC2 is a two-dimensional version of TEC1. Refer to the documentation for TEC1 for additional details.

With TEC1, one node is used to represent the hot side, and another node is used to represent the cold side. With TEC2, any number of nodes may be used to represent the hot side, and an equal number of corresponding nodes may be used to represent the cold side. Linear conductors between these hot and cold node pairs may be optionally used to represent the thermal conduction through the device. Since the TEC has been subdivided, perhaps unevenly, each node pair must be assigned a relative weighting: how much of the total surface area corresponds to each node pair. This weighting is provided by a list of areas corresponding to each node pair.

To use TEC2, build a network (N pairs of nodes and N linear conductors between them) within a single SINDA submodel, list the names of the nodes and conductors in ARRAY DATA (as integer arrays in corresponding order), list the corresponding areas as a real array, then call TEC2 within VARIABLES 1, as specified below.

Given the device aspect ratio (X, the ratio of cross sectional area to thickness* for one couple) and *either* the number of bismuth telluride couples (N_{cp}) *or,* for other materials, the overall device properties Seebeck coefficient (S_m), material electric resistivity (R_m), and thermal conductivity (K_m), TEC2 applies the appropriate Q source terms to the nodes to simulate the TEC device. It may internally adjust the temperatures of these nodes in steady states (or in transients if they are arithmetic) as needed.

Usually, the IDs of the linear conductors that represents solid conduction through the TEC† are also supplied. If these IDs are supplied, then TEC2 also updates the conductance G of those conductors. Additional conductors representing other materials or heat transfer paths may be added separately in parallel.

The TEC electrical state may be specified by inputting one of the following: current, voltage, or power (see “Mode” below). Whichever two values are not specified are calculated by the routine.

* X has default units of either meters (UID=SI) or feet (UID=ENG), or as specified by the latest TECUNITS call.

† This conduction represents the conduction through the couples themselves, and not any potting compound or other material, which can be added separately (in parallel).

NOTE: The use of the TEC2 routine requires careful set-up of an appropriate model. Use of Thermal Desktop® (page xliii in the preface) greatly simplifies these data management tasks.

Calling sequence:

```
   CALL TEC2 ('smn', IDnc, IDnh, IDng, Areas, Mode, I, V, P
.          , X, Ncp, Sm, Rm, Km)
```

where:

smn. . . . . thermal submodel containing the nodes (and perhaps the conductor), single quotes

IDnc . . . . . integer SINDA array containing IDs of cold side nodes in submodel smn

IDnh. . . . . integer SINDA array containing IDs of hot side nodes in submodel smn. This list must be the same size as the code side IDs (IDnc) and must be in the same order: the first cold node is associated with the first hot node, etc.

IDng. . . . . integer SINDA array containing IDs of linear conductors between the hot and cold node pairs representing the back conduction through the device. If zero is input, no conductor is updated. Otherwise, the model units are assumed to be *tpower*, *length* depending on UID and any prior TECUNITS calls. If TECUNITS has not been called, then if UID=SI the units are Watts and meters, otherwise if UID=ENG the units are BTU/hr and feet. If these are NOT the correct units, input IDng as zero and update the conductor separately, perhaps using the value of Km returned by either this routine or BISTEL.

If input, this list must be the same size as the two node ID arrays (IDnc and IDnh) and must be in the same order: the first conductor goes between the first cold node and the first hot node, etc.

Areas . . . . . real (not integer) SINDA array containing the list of surface areas for each node pair. This list must be the same size as the two node ID arrays (IDnc and IDnh) and must be in the same order: the first area listed applies to the first cold node and the first hot node pair. Values must be greater than zero. *Units do not matter for this array as long as the same units are used for each entry in the array.* The "areas" are used strictly to calculate apportioning factors. Thus, if all the nodes are the same size, unit values can be input:

```
  100 = ALLSAME, 24, 1.0    $ 24 identical areas
```

Mode . . . . .

Value	Input Parameter	Returned Parameters
1	current (I)	voltage (V) and power (P)
2	voltage (V)	current (I) and power (P)
3	power (P)	current (I) and voltage (V)

A negative sign on "Mode" is used as a flag to signal the use of a "shallow" node update instead of a default "deep" update. See Guidance notes below.

I. . . . . Input current (amps) if Mode = 1, else the returned current. The current is positive for the normal (cooling or heat pumping) direction.Note that this is a real, not integer, argument.

V . . . . . Input voltage (volts) if Mode = 2, else the returned voltage. The voltage is positive for the normal (cooling or heat pumping) direction.

P . . . . . Input electrical power (*always Watts*) if Mode = 3, else the returned power. In P is input, it must be zero or positive. Positive current and/or voltage is assumed.

X . . . . . Aspect ratio of one couple: area/thickness.* Use X=1.0 if the Rm and Km values are applied to the whole device (e.g., they have units of ohm and *tpower*/degree respectively). Melcor refers to this parameter as "G" and provides it in units of centimeters, which must be converted to the correct length units for TEC2: X = 0.01*G if Rm is in ohm-m and Km is in W/m-K (the default if UID=SI and no TECUNITS calls have been made). For example the total thermal conductance across the device is Km*X, and the total electrical resistance of the device is Rm/X.

Ncp . . . . . Input number of total couples (integer†). If zero, the user must input Sm, Rm, Km. Otherwise, a positive value is taken to mean the number of Bismuth Telluride couples, in which case Sm, Rm, and Km are calculated automatically and become output parameters

Sm. . . . . Input (Ncp = 0) or Output (Ncp > 0) effective Seebeck coefficient, V/K or V/R depending on UID. This value is for the whole device: the material property times 2*Ncp.

Rm . . . . . Input (Ncp = 0) or Output (Ncp > 0) effective resistivity, in units of ohm-length where length units depend on the value of UID and any calls to TECUNITS. The default is ohm-m for UID=SI and ohm-ft for UID=ENG. This value is for the whole device: the material property times 2*Ncp.

Km . . . . . Input (Ncp = 0) or Output (Ncp > 0) conductivity, in units of *tpower*/*length*-degree where thermal power and length units depend on the value of UID and any calls to TECUNITS. The default is W/m-K for UID=SI and BTU/hr-ft-R for UID=ENG. This is a real (not integer) value. This value is for the whole device: the material property times 2*Ncp.

Example:

```
Header array data, mymod
      1 = 1,2,3,4,5           $ cold nodes
      11 = 11,12,13,14,15     $ corresponding hot nodes
      21 = 1,2,3,4,5          $ linear conductors between cold & hot
      100 = 0.12, 0.24, 0.24, 0.24, 0.12$ “areas”
Header variables 1, mymod
      call qvtemp('mymod')   $ Just in case nearby nodes are variable
      CALL TEC2('mymod', na1, na11, na21, a100, -1, Current, Volt, Power,
     .   5.0e-5, 64, STEST, RTEST, CTEST)
```

* X has default units of either meters (UID=SI) or feet (UID=ENG), or as specified by the latest TECUNITS call.

† If a register is used to define this value, be sure to declare it as an integer using the "INT:" prefix in REGISTER DATA.

Caution Because TEC2 updates nodal temperatures for stability, any other source terms applied to these nodes or to attached nodes should have been updated before TEC2 is called. This may be accomplished by inserting a call to QVTEMP before any TEC2 calls, as shown in the example above.

Guidance Because TEC2 updates nodal temperatures as part of its solution, and because node solutions are sensitive to the sequence of Q updates (see above Caution), if the TEC nodes are adjacent to nodes with Qs that are updated late in the sequence (including heaters and other TEC stages ... see below), then the default “deep” solution may become unstable and need to be replaced with a less aggressive solution that only updates the TEC nodes themselves (“shallow” solution). The shallow solution is signaled via the use of a negative sign on the Mode variable. The upshot: if convergence or stability issues are encountered, try negative Mode values.

Caution TEC2 is meant to simulate single stage devices only. Cascaded (multiple stage) devices may be modeled using multiple TEC2 calls (one per stage), but in that case *the user is strongly advised to use separate nodes between the stages* (e.g., to separate the hot side of one stage from the cold side of the next stage, perhaps with a conductance representing the bond). Also, use negative Mode values (shallow solutions) to keep the order in which nodal Qs are updated from disrupting convergence.

Limitation (This discussion only applies to the variable-property option enabled when Ncp > 0.) Thermoelectric coolers are arranged in series electrically, but in parallel thermally: a single value of current exists despite local variations in electrical resistivity, thermal conductivity, temperature gradient, and Seebeck coefficient. Since the resolution of the TEC model is independent of the number of couples, the actual temperature of any one couple is unknown. Therefore the total resistance of the device can only be calculated approximately using area-weighted average temperatures. Local properties are applied to the greatest extent possible, and are used exclusively when current is specified (Mode=1). In other modes (voltage or power specified), average properties must be used to first calculate the current before applying that current to local conditions within each computational cell.

7.7.11 Surface Recession due to Melting or Sublimation

This section describes utilities for one-dimensional modeling of surface recession for a thin layer of material melting off of a surface. (The melted or sublimated phase is considered to be shed from the surface is and no longer part of the model. This is different than the FUSION routine, which assumes the melted phase stays in place and is still considered part of the model.) With additional assumptions,* this utility can be applied to sublimation and ablation. If a large surface has been covered or coated with a material that can melt or sublime, then each node on the discretized model of that surface can be attached to a distinct 1D surface recession model.

The 1D assumption is normally adequate because of the low thermal conductivity of typical materials, coupled with small aspect ratios (thickness divided by distance between adjacent surface nodes); temperature gradients can be assumed to exist only within the thickness dimension of the melting or sublimating material. Of course, adjacent surface locations might melt at different rates, and the substrate can itself be a 2D or 3D model.

Provisions have been made for modeling multiple materials, each of which melt in turn.

7.7.11.1 Theory and Modeling Approach

Surface recession represents special modeling challenges because material is irreversibly lost in the process. There are two basic techniques for handling such effects: shrinking nodes or shedding them.

Shrinking nodes is problematic because the characteristic time for a node (the capacitance over sum of conductance: “CSG” = C/ΣG) reduces by a factor of four for each halving of a node: time steps can grow arbitrarily small. The problems associated with shrinking nodes becomes acute considering that some materials melt completely (e.g., ice off of a steel plate).

The alternative is shedding nodes that have melted and thus retaining the original size of the remaining nodes. However, SINDA only provides mechanisms for nodes to be dropped as part of a higher-level operation: the BUILD operation performed *between* solutions. A more realistic problem with shedding nodes is the need to intelligently rewire heat loads and conductors: the environmental loads and connections must somehow “move” from shed nodes to the remaining nodes. Such environmental effects are typically calculated by other programs (e.g., Thermal Desktop/RadCAD, CFD codes, etc.) which have prepared their inputs for SINDA/FLUINT assuming that surface nodes do not change. Thus, redefining which nodes represent the surface during a run is problematic.

* This utility is limited by use of a single, constant phase change temperature, and a single, constant phase change energy at that temperature. For sublimation, the necessary additional assumptions include constant pressure and rapid sweeping of the vapor away from the surface (such that there is no mass transfer limitation). No actual ablative material, on the other hand, behaves in a way consistent with these limitations. Nonetheless, these capabilities have been used for first-order approximations of *non-charring* ablative materials, collapsing the actual complex material response into a highly approximate “ablation temperature” and “heat of ablation.”

In the past, the above two issues have been avoided by surface recession simulation routines that work “outside” of SINDA: they did not require SINDA nodes even though they used internal methods that are similar to SINDA (i.e., “hidden” nodes and conductors). Such approaches are cumbersome, however, since they must replicate SINDA capabilities such as the flexibility of boundary and initial conditions, time- and temperature-dependent properties, and even postprocessing.

The RECESS routine described in Section 7.7.11.2 takes a different approach: it applies to SINDA nodes and conductors built separately by the user or by a preprocessing program, and therefore RECESS can exploit all SINDA modeling options and postprocessing utilities. Nodes are *partially* shrunk as they melt or sublimate, and are eventually *collapsed* (but not shed). “Collapsing” means that there is no temperature drop nor energy storage associated with the nodes and conductors which represent missing material, nor is any change required to the topology of the network: the surface nodes remain in place.

RECESS internally calls the BDYNOD utility (Section 7.6.19) to keep the surface of the melting material at the phase change temperature. When the material has either cooled below the melt temperature, or has melted completely, the surface node is released (via RELNOD). Internal nodes will be collapsed if they have melted sufficiently. If they are diffusion nodes, they will be treated henceforth as arithmetic nodes using an internal call to the ARITNOD utility (Section 7.6.21). This conversion is necessary in order that the collapsed nodes not limit the time step despite a large adjacent conductance, which itself is necessary to keep temperature drops in the missing material negligibly small. This process is represented graphically in Figure 7-3. (Although diffusion nodes are presented and their use is recommended, arithmetic nodes may alternatively be used to represent the receding material.) The degree of resolution (the number of nodes or “layers” with the receding material) is a choice left to the user.

Notice that the finite differencing scheme used by RECESS places the mass of each layer at the “downstream” edge: each layer within the receding material is represented by a single linear conductor (whose conductance is inversely proportional to the thickness of that layer) and a single node (whose capacitance is proportional to that same thickness).

RECESS does not wait until a node is completely consumed before converting it to an arithmetic node, since to do so would cause repeated cycles of diminishingly small time steps as each layer is melted. Instead, once the currently melting node is reduced to 50%* of the thickness of the subsequent node, it is converted to arithmetic and the control volumes are redrawn: the capacitance and energy associated with the collapsed node are moved to the next node. This redrawing changes the center of thermal mass and therefore the spatial location of the next node, which can cause jaggedness in temperature plots versus time. The default setting of 50% means that the time step will normally not vary by more than a factor of four as each layer is melted. For the final layer, a tolerance is used since there is no place to move the residual mass: to avoid vanishing time steps, a small about of mass† is simply discarded if recession is complete (e.g., all ice has melted). The user may choose to prevent the last layer from melting if the model is not tolerant of complete melt-through.

* See RTLIM in RECESS_SET
† See THKLAST in RECESS_SET

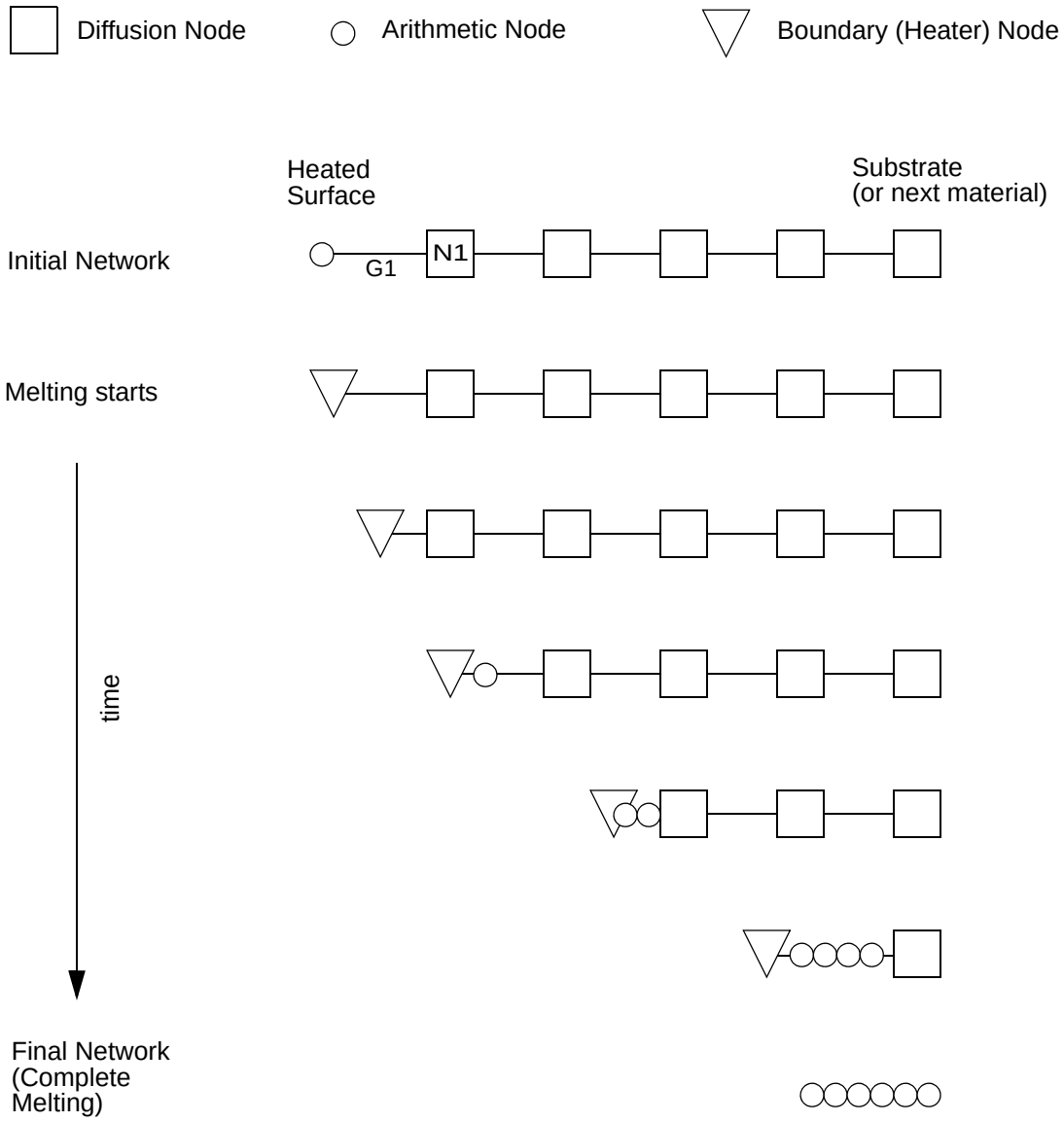


Figure 7-3 Network Changes Invoked by RECESS for Simulating Mass Loss

SINDA does not allow zero distance between collapsed layers since it cannot tolerate zero resistance (infinite conductance). Therefore, a small but finite tolerance* is used for the thickness of each collapsed layer such that the temperature drop through those arithmetic nodes is negligible, but not zero.

* See THKMIN in RECESS_SET

7.7.11.2 Surface Recession Routines

The RECESS routine simulates a melting or sublimating material using a 1D (through-thickness) finite difference model. RECESS may be called multiple times as needed to model different surface locations, or coatings of multiple melting materials on a single surface, or both. *Each call operates on a unique string of nodes and conductors that must be supplied independently by the user.* However, this requirement also means that the user is free to use any SINDA node, conductor, or source options in the construction of the 1D subnetwork that is to represent the melting section.

The operation of RECESS may be influenced not only via the size (resolution) and type of the subnetwork elements, it may also be customized using control variables accessible in a separate routine, RECESS_SET.

This subsection details RECESS and RECESS_SET usage. The next subsection (Section 7.7.11.3) provides a complete example including network and array definitions, along with conversions from the obsolete ABLATS utility.

NOTE: The use of the surface recession routines requires careful set-up of an appropriate model, including specifying array data and appropriate expressions pointing to that data in the node and conductor definitions. Use of Thermal Desktop® (page xliii in the preface) greatly simplifies these data management tasks.

Subroutine Name: RECESS

Description: RECESS operates on a 1D subnetwork (linear string of nodes and conductors) specified separately by the user to simulate irreversible melting, sublimation, or simplified ablation.

The first node and conductor representing the first melt layer in the 1D network (labeled G1 and N1 in Figure 7-3) must both exist in the same submodel, and although there is no requirement for the remaining subnetwork to be in that same submodel, the user will find it convenient that it is. The recession subnetwork must also be built entirely of linear conductors, and no side connections are tolerated. (No such restrictions exist for the rest of the submodel, much less the model.) Refer to the top diagram in Figure 7-3 for an example of a 5-layer melt model plus an arithmetic surface node (which need not be in the same submodel, and neither must it be arithmetic as shown in that diagram).

The nodes may be arithmetic or diffusion nodes of any type (e.g., 3-blank, SIV, etc.).* The surface node may, in addition, be a boundary or heater node, and it might be the last node of another melt material placed outside of the current material. The conductors must be linear, but again they may be any type of linear conductor.

An important input to RECESS is a floating point SINDA array containing the initial thicknesses of each layer. This array is presumed to be ordered from the outside (the first layer to melt) to the inside (the last to melt). The thicknesses need not be equal. RECESS will operate on this array, changing layer thicknesses as needed: the array will therefore contain the current thickness of each

* *If they are diffusion nodes, they all must be diffusion nodes, and if arithmetic, all must be arithmetic.* This restriction does not apply to the surface node, whose type can be specified independently. Similarly, if a negative node ID has been input to signal leaving the last layer unmelted, then the last node may be a different type.

layer. This array forms the "memory" needed by the RECESS routine, and the cells of the array should not normally be adjusted by the user* except using appropriate caution in special circumstances.

For example, assuming a certain segment of melt material in submodel *panel* was to be composed of 10 initially equally spaced nodes (stored in the register *nlayer*), and the total initial thickness was 0.01 cm (stored in the register *ithick*), then the ALLSAME command can be used to fill an array (say #100) appropriately:

```
header register data
  int:nlayer= 10        $ resolution of ice: number of layers
  ithick    = 0.01      $ initial thickness of ice or melt material
header array data, panel
  100 = allsame, nlayer, ithick/nlayer $ nlayer values of ithick/nlayer
```

It is the user's responsibility to use this same array in the definition of the capacitances and conductances of their 1D subnetwork. Updates to those capacitances and conductances will then happen automatically via the spreadsheet/expression system (Section 2.8). For example, if a surface node 100 of area *Asurf* were attached to a string of "3 blank" (constant property) nodes and conductors numbered 101, 102, ... 110, and if the capacitance, density, conductivity, and initial temperature of the material had been defined in registers (say: *CPabl*, *DENabl*, *Kabl*, and *Tinit*, respectively), then the node and conductor data blocks might appear as follows:

```
header node data, panel
  -100, Tinit, -1.0     $ arithmetic (in this case) surface node
  101, Tinit, CPice*DENice*Asurf*A(100+1) $ first node to melt
  102, Tinit, CPice*DENice*Asurf*A(100+2)
    ...
  110, Tinit, CPice*DENice*Asurf*A(100+10)$ last node to melt
header conductor data, panel
  101, 100, 101, Kice*Asurf/A(100+1) $ surface to first melt node
  102, 101, 102, Kice*Asurf/A(100+2)
    ...
  110, 109, 110, Kice*Asurf/A(100+10)
```

A significant shorthand method for defining such relationships can be exploited using the base#increment syntax for expressions (Section 2.8.8.1):

```
header node data, panel
  -100, Tinit, -1.0     $ arithmetic surface node
  GEN 101, nlayer, 1, Tinit, CPice*DENice*Asurf*A(100+1#1)
header conductor data, panel
  GEN 101, nlayer, 1, 100,1, 101,1, Kice*Asurf/A(100+1#1)
```

* Pay particular attention to accidental changes, such as defining the array in terms of a register then changing that register. To continue the example used in this subsection, the "ithick" register should not be changed during program execution or the melt material will be inappropriately restored.

This method can similarly be applied to more complex types of nodes and conductors. For example, if the melt material (ice or solid phase) had used a temperature-dependent conductivity defined in array #99 from submodel *props*:

```
header conductor data, panel
  SIM 101, nlayer, 1, 100,1, 101,1, props.A99, Asurf/A(100+1#1)
```

Restriction: However the nodal capacitances are defined, *the capacitance must be linearly proportional to the thickness*. In other words, a capacitance proportional to "X*Array_value" is legal, but one that is proportional to "Y + X*Array_value" is illegal (but cannot be trapped by SINDA as an error). This restriction means that the entire node must be made of melt material; *a melting material cannot be combined with a non-melting material in the same node*. For example, the DIV option should not be used for a melting node, unless *both* the F1 and F2 factors are proportional to the *same* array value (which represents the total node thickness: the two materials must be in parallel with the direction of recession, not in series).

The one exception to the above rule is the last node, which is often combined (summed capacitance) with a node representing a non-melting back wall. If this is the case, use a negative sign on the conductor ID (idc1), but make sure that there is a nonzero capacitance left in that node in the case of complete melt-through, or else SINDA will err off with a zero capacitance (C) on a diffusion node.

RECESS should only be called in VARIABLES 2. If called from within a steady state solution, it returns immediately without performing any operation, ignoring (for example) nodes that are above the phase change temperature as a result of the steady state analysis.

Use of MATMET>0 is strongly advised (MATMET=12 is the default).

Calling Sequence:

```
      CALL RECESS(smn, idc1, idn1, Athk, Tmelt, HRA,
     +          Xleft, DTfact)
```

where:

`smn`.....submodel name containing all of ice (solid "melt material") nodes and conductors, including surface node (single quotes)

`idc1`.....ID of first conductor (labeled G1 in Figure 7-3 between surface node and first melt material node) (integer). If negative, the minus sign is a flag signaling that the last nodal layer can be completely melted, but if so it is not to be turned into an arithmetic node, but instead is to be left as a diffusion node. This option is useful when the last layer has been summed into a non-melting back wall node.

`idn1` ..... ID of first melt material node (labeled N1 in Figure 7-3) (integer). If negative, the minus sign is a flag signaling that the last nodal layer is to be left unmelted for models that cannot tolerate complete melt-through. Use of this option overrules a negative sign on idc1, since the latter option is only appropriate in the case of complete loss of surface material (melt-through).

`Athk` ..... SINDA real array (e.g, "smn.A200") containing the list of melt material layer thicknesses, from the surface to the substrate

`Tmelt` ..... Input phase change temperature, such as melt or sublimation temperature (user units)

`HRA` ..... Enthalpy of phase change (e.g., heat of fusion or heat of sublimation) times density (R for "rho") times surface area: the energy released per unit thickness melted (e.g., BTU/ft, J/m, consistent with user energy/length units)

`Xleft` ..... Output melt material thickness remaining (user units of length, real)

`DTfact` .... Output recommended time step factor (unitless, real). If DTfact is significantly less than unity, DTIMEH should be reduced and the run repeated.*

Examples (see Section 7.7.11.3 for complete examples):

```
      CALL RECESS('PANEL', 101, 101, A100, TempMelt, Hfus*DENice*Asurf,
     +     XK101, DTEST)
```

The melt temperature (TempMelt) and the HRA product might change during the course of the analysis, but the user is cautioned only to adjust those parameters smoothly and gradually.

Xleft is the tally of actual thickness and may therefore be zero. For coarse discretization, Xleft will not change smoothly due to sensible heating within each layer.

Caution: *Time step control and BACKUP.* DO NOT BACKUP and repeat the time step *after* calling RECESS.

DTfact, if less than unity, represents the fraction by which the previous/current time step would have to be reduced such that no more than 1/2 of one layer were melted during that interval. If the returned value of DTfact is too small, significant error has been accrued and a new run should be made with reduced DTIMEH. No warnings will be issued to this effect.

Guidance: *Parametric runs and restoring melted materials.* When making parametric runs, or performing optimization, correlation/calibration, or statistical analysis, it will be necessary to analytically restore the melted material to its original thickness and state. Array values and nodal states will be reset using RESPAR (as long as they were not excluded in the original SVPART call). Or, if the array cells were defined using registers they can be reset by changing those values.†

* Because the actions of RECESS are irreversible, *BACKUP should not be used after a RECESS call.*

† For the example in this section, the register ithick could be changed. If that value is invariant during each run, the user can force the array to be refreshed by changing ithick to be a new value, calling upreg, then restoring ithick to its original value.

Guidance: *Leaving the last node unmelted.* In some models, the last node cannot be set to an arithmetic mode if melting is complete, or the last conductor cannot be set to a huge value. For these cases (which includes the last melt layer in a Thermal Desktop model), set the node ID (idn1) negative as a signal. The final remaining thickness, which will be inert (capacitance and conductance but no phase change), will then be at least the total thickness divided by the number of layers in the case of even discretization. In fact, this residual thickness can be larger than this value (up to 150%* of the last layer) if diffusion nodes are used since the penultimate layer will have been combined into the ultimate layer before melting terminates.

Guidance: *Letting the last node melt, but not completely.* The last node is often combined (summed capacitance) with a node representing a non-melting back wall. If this is the case, use a negative sign on the conductor ID (idc1), but make sure that there is a nonzero capacitance left in that node in the case of complete melting, or else SINDA will err off with a zero capacitance (C) on a diffusion node.

Guidance: The utility LAYINIT (Section 7.7.12) may be used to set or reset layer thicknesses between solutions (e.g., in OPERATIONS).

Guidance: The REC_QRATE (identical to ACCQRATE) utility, documented in Section 7.7.12, may be used to fetch the net surface heat rate from the last RECESS call.

Subroutine Name: RECESS_SET

Description: RECESS_SET is used to customize operation of subsequent RECESS calls. Three control variables may be set using repeated calls as necessary:

RTLIM The ratio of the thickness of a current melting layer to the next, below which the current node should not shrink. If the current melting layer drops below this relative thickness, then it is collapsed and the control volumes are redrawn. In effect, collapsing means the mass and energy are moved from the current node (which becomes massless or arithmetic) to the next node. This parameter is relevant only if diffusion nodes are used.

RTLIM must be between 0 and 1, exclusive. The default is 0.5: the current melting node will be collapsed and its mass and energy added to the next node when its mass becomes less than 50% of the mass of that next node.†

* See RTLIM in RECESS_SET

† For ACCRETE, the current layer is removed when it is below 0.9*RTLIM (45% by default of the next layer), and the current layer is split when it is above 1+RTLIM (150% by default) of the maximum. In other words, unlike RECESS, since it both melt and thaw, ACCRETE requires some hysteresis (0.9*RTLIM to 1.0*RTLIM) for stability.

Plots of temperatures within a melt material reveal a certain jaggedness as a result of this parameter because control volumes are dynamically redrawn. Smaller values of RTLIM yield smoother curves and slightly more accurate predictions, but halving RTLIM can potentially quarter the time step and therefore quadruple the solution costs.

THKLAST The thickness of the last piece of melt material to discard, in user units of length. This parameter is relevant only if diffusion nodes are used.

The default value for THKLAST is 1.0e-4 (meters presumably, if UID=SI) and 3.2808e-4 (feet presumably, if UID=ENG). The final node in the melt material has no place which can inherit its mass and energy, so it continues to shrink, dropping the time constant. THKLAST is therefore a tolerance: a lower limit to the degree to which the last node can shrink. Smaller values will yield more accuracy but will generate small time steps. For example, halving THKLAST can potentially quarter the time step and therefore quadruple the solution costs.

THKMIN The lowest thickness to which any layer can shrink. This parameter is relevant whether or not diffusion nodes are used. It defines the lowest possible resistance since SINDA cannot tolerate zero resistances (infinite conductances).

The default value for THKMIN is 1.0e-6 (meters presumably, if UID=SI) and 3.2808e-6 (feet presumably, if UID=ENG). Smaller values will yield more accuracy but will lead to numeric difficulties: matrix methods will be required as a minimum. The ideal value results in *reasonably* negligible temperature drops (i.e., a degree or two) in the collapsed (melted) portion. Reductions below that value serve no purpose and could lead to problematic executions.

THKMAX See ACCRETE (Section 7.7.12).

Calling Sequence:

```
CALL RECESS_SET(name,value)
```

where:

name ..... the name of the control variable to specify: ‘RTLIM’ or ‘THKMIN’ or ‘THKLAST’ in single quotes

value ..... the positive value of the control variable to use:
if name = ‘RTLIM’, a unitless value between zero and one (exclusive) is expected.
if name = ‘THKMIN’ or ‘THKLAST’, a positive thickness (user units of length) is expected.

Example:

```
CALL RECESS_SET('RTLIM',0.1)
CALL RECESS_SET('THKLAST', 0.0625/12.)
```

RECESS_SET applies to the next RECESS invocation. It should be called within OPERATIONS or at least before any RECESS call is made. Control values should not normally be changed within a run (e.g., between RECESS calls whether or not they are for the same melt material section), excepting perhaps parametric runs or between groups of greatly differing materials or dimensions (such that each requires unique tolerances).

Subroutine Name: RECESS_RATE

Description: RECESS_RATE works in a manner similar to RECESS, but the rate of recession is specified via an equation of the form $R = A*q_{cw}^{B}$, where R is the linear recession velocity, q_{cw} is the cold wall heat flux, and A and B are experimentally determined rate constants. RECESS_RATE is expected to be used with a surface heat flux that is a function both of time and of the temperature of the surface node. The heat flux at zero temperature is stored in the first column of a bivariate array is used to define the value of Q_{cw}.

One significant difference with RECESS is that the surface node is expected to be a diffusion node up to the time that material loss begins, since using an arithmetic node for the surface temperature tends to be very unstable. Once melting or sublimation starts, the surface node becomes a boundary node as it does in RECESS. If melting stops, the surface node will switch back to be the same type as the first melt material layer node: a diffusion node if the first layer hasn't been depleted yet, and an arithmetic node if it has been depleted.

Like the RECESS routine, an important input to RECESS_RATE is a floating point SINDA array containing the initial thicknesses of each layer. This array is presumed to be ordered from the outside (the first layer to melt) to the inside (the last to melt). The thicknesses need not be equal. RECESS_RATE will operate on this array, changing layer thicknesses as needed: the array will therefore contain the current thickness of each layer. This array forms the "memory" needed by the RECESS_RATE routine, and the cells of the array should not normally be adjusted by the user* except using appropriate caution in special circumstances.

RECESS_RATE should only be called in VARIABLES 2. If called from within a steady state solution, it returns immediately without performing any operation, ignoring (for example) nodes that are above the phase change temperature as a result of the steady state analysis.

Calling Sequence:

```
   CALL RECESS_RATE(smn, idc1, idn1, Athk, Tmelt, Area,
+         Xleft, DTfact, Aqcw, ArateEq, mult)
```

* Pay particular attention to accidental changes, such as defining the array in terms of a register then changing that register. To continue the example used in this subsection, the "ithick" register should not be changed during program execution or the melted material will be inappropriately restored.

where:

`smn` ..... submodel name containing all of ice (solid “melt material”) nodes and conductors, including surface node (single quotes)

`idc1` ..... ID of first conductor (labeled G1 in Figure 7-3 between surface node and first melt material node) (integer) If negative, the minus sign is a flag signaling that the last nodal layer can be completely melted, but if so it is not to be turned into an arithmetic node, but instead is to be left as a diffusion node. This option is useful when the last layer has been summed into a non-melting back wall node.

`idn1` ..... ID of first melt material node (labeled N1 in Figure 7-3) (integer). If negative, the minus sign is a flag signaling that the last nodal layer is to be left unmelted for models that cannot tolerate complete melt-through.

`Athk` ..... SINDA real array (e.g, “smn.A200”) containing the list of melt material layer thicknesses, from the surface to the substrate

`Tmelt` ..... Input phase change temperature, such as melt or sublimation temperature (user units)

`Area` ..... Surface area of the receding surface (user units of length squared, real)

`Xleft` ..... Output melt material thickness remaining (user units of length, real)

`DTfact` .... Output recommended time step factor (unitless, real). If DTfact is significantly less than unity, DTIMEH should be reduced and the run repeated.*

`Aqcw` ..... SINDA bivariate array (e.g. “smn.a400”) containing the cold wall heat flux, used in the rate equation as qcw, stored in the first column of the array. It should be a function of time and temperature.

`ArateEq` ... SINDA real array containing conversion constants, the rate constants A and B, and the heat flux limits used to separate sets of rate constants. The first value in this array converts the velocity of the melt rate equation to the model units. The second value converts the flux units of the qcw value to model flux units. The third value is the A term from the rate equation. The fourth value is the B term from the rate equation. If the rate equation is subdivided into segments for various heat fluxes, more terms can be added to the array by adding the flux breakpoint and another A and B term corresponding to fluxes higher than the last breakpoint. (See example below.)

`mult` ..... multiplier to be applied to the qcw term found in the Aqcw array (real)

As an example, consider two rate equations used to describe the recession rate as two different flux levels:

$$R = 0.01(q_{cw})^{0.05} \text{ m/sec for } q < 20 \text{ J/sec-m}^2$$

$$R = 0.02(q_{cw})^{0.052} \text{ m/sec for } q >= 20 \text{ J/sec-m}^2$$

* Because the actions of RECESS_RATE are irreversible, *BACKUP should not be used after a* RECESS_RATE *call.*

If the model units were feet and hours, then ArateEq array (say, array #1000) would consist of:

```
1000 = 3.2808*3600., 11356.5/3600., .01, .05, 20., .02, .052
```

Use of MATMET>0 is strongly advised (MATMET=12 is the default).

7.7.11.3 Example

A material to be melted off a surface has a density of 15 lb_m/ft^3, a conductivity of 0.05 BTU/hr-ft-R, a specific heat of 0.26 BTU/lb_m-R, an emissivity of 0.9, a melt temperature of 950°F, and a heat of fusion of 5500 BTU/lb_m.

One half inch of this material is exposed to a 2000°F effective radiation source temperature. The material is initially at 70°F. A unit area (one square foot) is used for analysis purposes, and a resolution (number of finite difference layers) of 20 is chosen.

This problem will be demonstrated both using RECESS and, as an aid in model conversion, ABLATS (an unsupported routine used in older SINDA versions for materials shed from a surface).

ABLATS Method

ABLATS is an unsupported and undocumented routine. It uses an internal ("hidden node") representation of the melt material (see Section 7.7.11.1), is unit dependent, and has other deficiencies. Yet it is still present in many SINDA/FLUINT models. Therefore, as an aid in *conversion* (but not *usage*), the ABLATS method of building the above model will be briefly described before showing the preferred method using RECESS.

In ARRAY DATA, a list of values is prepared. This list (which is of mixed type and therefore requires the MIXARRAY flag be set in OPTIONS DATA) consists of an integer ID, the number of layers, the heat rate at the surface, the total thickness, the surface area, the phase change temperature, the heat of phase change, the Stefan-Boltzmann constant, the emissivity, the radiative source temperature, and allocation of extra memory. For the sample problem above, this becomes:

```
HEADER ARRAY DATA, PANEL
      15 = 15, 20, 70.0,0.0,0.5/12.,1.0,950.0,5500.0,SIGMA,0.9,2000.0,
     SPACE,24
```

The call in VARIABLES 2 becomes:

```
HEADER VARIABLES 2, PANEL
      CALL ABLATS(A15, 15., 0.26., 0.05, T1000, C1000)
```

Note that the T and C location of the underlying substrate must be provided, in this case as node number 1000. The second through fourth arguments for ABLATS are the density, specific heat, and conductivity of the material. These could also be references to bivariate arrays.

RECESS Method

First, as usual, appropriate registers are defined for convenience and ease of change:

```
HEADER REGISTER DATA
      nlayer = 20       $ doesn't really need to be INT
                        $ except if referenced in logic
      ithick = 0.5/12.0 $ same names as above for ease of explanation
      Asurf = 1.0
      CPD   = 0.26*15.0 $ Cp times density
      Ksolid= 0.05      $ conductivity of solid (pre-melt) material
      HAD   = 5500.0*15.$ Heat of fusion or sublimation times density
      Thot  = 2000.0    $ source temp
      Tinit = 70.0      $ Initial temp
      emis  = 0.9       $ emissivity
```

Next, an array is added to contain the layer thicknesses:

```
HEADER ARRAY DATA, PANEL $ any submodel name is OK of course
      1000 = allsame, nlayer, ithick/nlayer
```

The next step is to build the network. In the following example, the surface node will be assumed to be located in a different submodel: surf.11. The numbering for the melt material subnetwork is chosen to be 1 through 20 for both nodes and their corresponding conductors:

```
HEADER NODE DATA,SURF
      -1, Thot, -1.0    $ a heater (bdy) node for the source
      11, Tinit, -1.0   $ surface node (arithmetic in this case)
HEADER CONDUCTOR DATA, SURF
      -11, 1, 11, emis*asurf  $ connect to source
HEADER NODE DATA, PANEL
      gen 1, nlayer, 1, Tinit, CPD*Asurf*A(1000 +1#1)$ melt nodes
HEADER CONDUCTOR DATA, PANEL
      1, surf.11, 1, Ksolid*Asurf/A(1000 +1)$ first conductor
      gen 2,nlayer-1,1, 1,1, 2,1, Ksolid*Asurf/A(1000 +2#1)  $ the rest
```

Finally, the call to RECESS is made, and the run terminated when all material has been depleted:

```
HEADER VARIABLES 2, PANEL
c
      call recess('panel',1,1, A1000, 950., HAD*Asurf, xtest, dtest)
c
      if(xtest .le. 0.0) timend=timen  $ terminate run if done
c issue warning to user file if time steps were too large
      if(dtest .lt. 0.1) write(nuser1,*) ' should rerun this with dtimeh ='
     &  , dtest*dtimeu
```

7.7.12 Ice Accretion and Melting

RECESS assumes that all material shed from a surface is irrecoverably lost. However, in some cases either the reverse is the concern (ice or frost accumulation), or the surface can both lose material when heated then regain it when cold (for example, a surface submerged in a liquid pool).

ACCRETE is a utility very similar to RECESS (and in many ways, simply a bi-directional version of RECESS). It is available for modeling the growth of ice on surfaces (including on the inside of pipes). Because of the similarity, this utility ("ACCRETE") will be documented without repeating the RECESS description. Instead, the differences between RECESS and ACCRETE will be listed in this section. In other words, Section 7.7.11 is prerequisite reading.

Subroutine ACCRETE

ACCRETE, a variation of RECESS (Section 7.7.11.2), is available for modeling the accumulation or melting of ice or frost. The calling sequence is:

```
      CALL ACCRETE(smn, idc1, idn1, Athk, Tmelt, HRA,
     +         Xleft, DTfact)
```

ACCRETE is almost identical to RECESS, except as noted:

1. If the outer surface is cooled, ice will accumulate in the outermost active layer in proportion to the cooling rate. In other words, ACCRETE is like RECESS except that it is reversible. In RECESS, material disappears irreversibly, whereas it can disappear or reappear (melt/sublimate or freeze) when using ACCRETE. *The material added to the ice layer is assumed to be readily available, such that the energy released by freezing is the only limiting factor (and not mass transfer limitations).*

2. Multiple stacks of layers (i.e., more than one set of layers on top of each other and controlled by separate ACCRETE calls and arrays, perhaps using different freeze points and thermal properties) are legal. However, multiple stacks should be used with caution because inner stacks are not aware of the existence of outer layers: ice may melt or accumulate *within* inside stacks.

3. For ice forming or sublimating in air, use the heat of sublimation (from vapor to ice phase) instead. For ice melting in air, use the heat of fusion (from liquid to solid phase) if the droplets fall off without vaporizing. For ice forming from or melting to liquid, use the heat of fusion. ($\Delta H_{sub} = \Delta H_{fg} + \Delta H_{fus}$ at the triple point.)

4. When forming frost on a cold surface from a moist air environment, or when subliming from solid ice directly to vapor, the water vapor must diffuse through the air. A heat transfer coefficient based on condensation to a thin liquid film or freezing to an ice layer is an overestimate because the air acts as a barrier to diffusion.

   The HTUDIF or HTFDIF utilities may be used to assist in the mass transfer calculations (see example below), noting that they is not directly applicable* but that they nonetheless produces outputs that can be used for further calculations. Also note that at reduced humidity and low external convection (e.g., diffusion-limited accretion), the ice surface temperature will drop below the triple point, whereas ACCRETE assumes a constant phase transition temperature (freeze point). In other words, ***ACCRETE methods are not applicable to frost formation in air unless both humidity and convection coefficients are very high.***†

5. "Melting" is not always the same as "freezing in the reverse direction." When forming frost from a moist air environment, the aforementioned effects of diffusion-limited mass transfer and decrease in surface (freeze point) temperature may occur. Furthermore, melting (unlike sublimation) will not experience these mass transfer-limiting effects if the droplets that form fall away from the surface instead of accumulating. In that case, the heat of fusion would be used for melting, whereas the heat of sublimation is normally be used for ice accretion in an air environment.

6. Similarly, "melting" is not the exact opposite of "freezing." Unlike melting, freezing should *ideally* include an advection term carrying the energy to the surface node that is associated with the flow of vapor or liquid that is freezing. This advection can be modeled using a linear (perhaps one-way) conductor whose conductance G is the product of the mass flow rate of ice being deposited times the specific heat of the vapor (if HRA includes the heat of sublimation) or liquid (if HRA includes only the heat of fusion). This term is often negligible since the mass flow rates are typically low, and also since the temperature difference between the freestream and freeze temperature is also usually small. The ability to frequently neglect this term is fortunate since the ice accretion rate is normally an *output*, not an *input*. The example below shows one method for including this term if it cannot be neglected, employing the utility routine ACCQRATE to report the mass flow rate of ice accumulation.

* HTUDIF is solves for an overall heat transfer coefficient from gas/vapor to a subcooled liquid or wall, across a liquid/vapor interface at which phase change occurs. However, this value may be used to calculate a liquid/vapor temperature which can in turn be used to correct gas/vapor-side heat transfer.

† In theory, it should be possible to reduce the freeze point gradually during a transient, but such methods have not yet been developed.

7. When modeling the freezing or melting of ice on the inside pipe wall of a fluid submodel, the ice mass flow rate is usually negligible compared to the flow rate through the pipe.* However, the constriction associated with the growth of the ice layer should be reflected by updating the appropriate geometric parameters (e.g., DH, AFI, AFJ).

When modeling the accretion of an ice layer, the starting point may be an ice layer of zero thickness: an initial condition corresponding to "no ice present." Since zero thickness results in zero capacitances and infinite conductances, the THKMIN parameter (see RECESS_SET) applies as a lower limit. Do not initialize the values in the thickness array to zero (e.g., "allsame, *nlayers*, 0.0") unless the conductance and capacitance values upon which they are based can tolerate zero values. Instead, initialize the values of the thickness array to any nonzero value and use the LAYINIT utility in OPERATIONS (see below) to reset the thicknesses to THKMIN. LAYINIT will apply the ARITNOD utility to the collapsed layers as needed.

As ice forms within such a collapsed stack, it will start forming in the "last" layer (the one farthest from the surface, and listed last in the thickness array). This "last" layer will grow in thermal resistance but will not expand to a diffusion node (per RELNOD) until its thickness has exceeded THKLAST, assuming it was defined as a diffusion node.

If the layers have been initialized to some value larger than THKMIN, then ice will not form in those inner layers. Instead, it will form in the outermost or "first" layer (nearest the surface: the first value listed in the thickness array).

A RECESS_SET control parameter is available that is relevant only to ACCRETE calls: THKMAX. This parameter defines the largest thickness to which any layer can grow. Thus, the maximum thickness for all of the ice becomes N*THKMAX, where N is the number of layers. If ice is present initially, THKMAX should be set large enough such that the existing layers are not thicker than this limit. If the ice grows thicker than N*THKMAX, a larger value of THKMAX should be considered in future runs. If modeling the growth of ice layers on the inside of a duct, THKMAX should be calculated such that the residual flow passage dimensions are larger than zero since FLUINT does not permit zero diameters and flow areas.

If the current layer grows thicker than (1+RTLIM)*THKMAX, it is reduced to THKMAX and RTLIM*THKMAX is moved to the next outer layer, which then continues accumulating ice.

Subroutine LAYINIT

LAYINIT is a utility to set or reset a stack of RECESS or ACCRETE layers. Specifically, the total current thickness is specified, and LAYINIT will reset the array values containing the layer thicknesses and, if appropriate, collapse or expand active layers. LAYINIT is intended to be used from within either OPERATIONS, PROCEDURE, or RELPROCEDURE.

* In the rare cases where this isn't true, MFRSET connectors (whose SMFR is set with the help of ACCQRATE results) can be used to extract or inject the appropriate mass flow rate to or from plena.

One of the primary purposes of LAYINIT is to initialize a stack of zero thickness in preparation for later ice accumulation. Therefore, zero is a value input for overall thickness. When zero thickness is specified, all layer nodes will be initially collapsed (per ARITNOD) and the layer thicknesses will be set to their minimum value: THKMIN.

If the input thickness is greater than THKMIN times the number of layers, then each layer will be initialized as active.

LAYINIT may be invoked as follows:

```
CALL LAYINIT('smn',IDc1,IDn1,Athk,THKtotal)
```

THKtotal is the total stack thickness to apply. If IDc1 is negative, then the last node is not collapsed to a RECESS node, per RECESS and ACCRETE usage. IDn1 may be negative to parallel the corresponding RECESS or ACCRETE call, but only the absolute value is used internally.

Note that the first 4 arguments of LAYINIT are intentionally identical to the first 4 arguments of the corresponding RECESS or ACCRETE call, to facilitate copying and editing of the text inputs.

Subroutine ACCQRATE

ACCQRATE is a reporting utility used to retrieve the value of the net heat rate (Watts or BTU/hr) for the last ACCRETE call.*

```
CALL ACCQRATE(Qnet)
```

Qnet is defined as positive for melting (or sublimation or ablation), and negative for ice formation.

One of the main purposes of ACCQRATE is to calculate the net mass flow rate to the surface, to be used for the minor advection term described above. The net mass rate being added to the surface is:

```
Emdot = -Qnet/DeltaH
```

where DeltaH is either the heat of sublimation or the heat of fusion, depending on what was used to define the HRA call in the ACCRETE (or RECESS) call.

* ACCQRATE will report the same term for the previous RECESS call. An alternative, REC_QRATE, is also available (same function, just an alternative name as a memory aid).

7.7.12.1 Example: Frost Formation on a Pipe

At time zero, a horizontal 3cm OD stainless steel pipe carrying 80K nitrogen is exposed to blowing air at 300K and 100% relative humidity. The ACCRETE utility is employed to model the accumulation of frost on the pipe wall. The complete input file is listed in Table 7-3.

Frost grows as tiny ice crystals, yielding a rather porous structure. Thus, while the density of water ice is about 920 kg/m3, and effective density of 100 kg/m3 is assumed for frost in this example. The thermal conductivity of frost is estimated from text book values, with a rough estimate of conductivity at the freeze point based on the density ratios. Note that this treatment is highly simplified, and that real frost can be extremely complex. The actual behavior is time-dependent, since the porosity will continue to reduce as water vapor penetrates, and as the ice sublimes in outer layers and refreezes in inner layers.

Another important simplification is the assumption that the surface of the frost can be assumed constant at the triple point of water 273.15K, such that the ACCRETE routine is applicable without modification (without a variable freeze point or heat of sublimation, for example). This assumption is clearly not valid for the first few seconds as the surface temperature jumps from 80K to 273.15K, but is also not valid if the relative humidity and/or the external film coefficients are too low, as will be described later. In other words, diffusion is not assumed to be a limiting factor.

The ID of the nitrogen pipe is assumed to be a boundary node (frost.100) at 80K, whereas the thermal mass of half the pipe is placed at the OD as a diffusion node (of volume *VolumSS*).

The freestream temperature is assumed to be boundary node (frost.200).

The 10 ice layers themselves are represented by nodes frost.1 through frost.10, with linear conductors representing the frost thermal resistance as conductors #1 through #10. Despite the fact that the ACCRETE thickness layers are defined from the outside in, the node and conductors are numbered from the inside out: node 1 freezes first, and node 10 freezes last. Array #1 contains the list of layer thicknesses, which are all initialized to a very small but nonzero value of 1.0e-6 meters.

Node 1 represents both first-to-form (last listed) ice layer, as well as the outer half of the stainless steel pipe. Thus, a DIV (two-material) option will be used for this node (#1), whereas other ice layers will be represented by SIV (single-material) nodes (#2 through #10). To prevent node frost.1 from being collapsed (per the internal ARITNOD call), a negative sign will be used as a signal on both the ACCRETE and LAYINIT calls on the *idc1* argument as a signal.

The minimum (collapsed) layer thickness (THKMIN) will be left at the default value of 1.e-6 meters (for SI models). The value of THKMAX will be set (via a RECESS_SET call) to 0.001 meters, meaning that the total ice thickness that will be accumulated will be 10*0.001=0.01m or 1cm. Once the frost reaches that thickness, accumulation will cease and the ice will grow cold instead, so THKMAX should be set large enough to accommodate the expected ice thickness (per preliminary runs), but not so thick such that the resolution of thermal gradients within the variable-conduction ice is lost. Finally, the starting thickness of the first layer formed (THKLAST) will be set to be 10% of THKMAX, although this protection against too-low time steps is not very applicable in this model because the first layer formed (the last layer in the thickness array) is composed of ever-present stainless steel as well as ice.

Actually, given the hours-long accumulation event (only the first hour of which is simulated in this run), using diffusion nodes to represent the ice layers is overkill. A variation using arithmetic nodes for the ice (see Table 7-4) achieves identical results in less time.

The nodal capacitances and frost conductances could be based on simple linear relationships if the frost thicknesses could be assumed small. Just in case they are not small, and for demonstration purposes, nonlinear calculations of volume ($\pi*(R_o^2-R_i^2)$) and resistance ($2\pi KL/\ln[R_o/R_i]$) are made using the current thicknesses as they are updated. In other words, the cylindrical geometry is respected. Unfortunately, the values within the thickness arrays (e.g., "frost.a(1+5)" for the fifth thickness from the surface) are *processor variables*: their value is not known to the code before processor execution begins. The default random values used within the preprocessor (between 1 and 2) cause mathematical problems in the nonlinear cylindrical conductance and volume expressions. Therefore, these expressions cannot be specified fully in the preprocessor, and are instead set in the processor using CHGREG calls in OPERATIONS. Refer to Section 2.8.8 for more details on processor variables used within data field formulas.

Use of a thin-layer approximation eliminates this considerable complexity, and is therefore recommended. Use more ice layers if necessary to help make this assumption more applicable, especially if they can also be assumed to be arithmetic (which is almost always true, and which helps eliminate the need to define volume/capacitance expressions).

The HRA term in the ACCRETE call is based on the heat of sublimation, since during the freeze event vapor water will transition to solid ice. Note that the surface area in the HRA term is permitted to grow as the ice builds up: the surface area (*SurfArea*) is adjusted based on the ice thickness (*Xleft*) reported by prior ACCRETE calls.

Node 11 is the arithmetic surface node, which is connected to the environmental boundary (frost.200) by a convection term (register *HNCeff* for the film coefficient), a radiation term (emissivity *emissIce*), and an advection term for the flow of vapor to the surface (register *EmdotFrz*).

The latter flow rate (EmdotFrz) will be calculated via a call to ACCQRATE placed after the ACCRETE call. The flow rate from the prior time step is applied to a subsequent time step, but this method is justifiable not only because the flow rate is slow to vary, but because this advection term is small to begin with. The primary purpose of the flow rate calculation is not, in fact, to model advection. Rather, this calculation is performed in order to compare with an estimation of the diffusion-limiting flow rate as described next.

The convection term (governed by the register *HNCeff*) deserves more thought and discussion. If the pipe were surrounded by 100% water vapor with no noncondensible gas, then an extremely high drop-wise or film-wise condensation coefficient (estimated to be on the order of 60,000 W/m^2-K) would apply. The presence of noncondensible gas (air, in this case) slows the effective convection, as described in Appendix B.5. A more realistic calculation for stagnant humid air is supplied by the NCHCT routine for natural convection to or from a horizontal cylinder. The resulting coefficient (about 6 W/m^2-K, or 4 order of magnitude smaller than the condensation coefficient) turns out to be too small to enable the use of ACCRETE because the resulting diffusion-limited flow rate (*Em-*

dotDif) is then always smaller than the power-limited ice accumulation rate (*EmdotFrz*). A value of 100 W/m^2-K, representative of a forced convection environment, is used instead such that the ACCRETE call is appropriate.*

The HTUDIF routine could be used to calculate the effective film coefficient, corrected for mass transfer effects, from freestream to just before the ice layer (assuming a thin liquid film present, which is implied by the assumption that the freeze point does not drift below the triple point). However, this coefficient would transverse the vapor-liquid transition, which is contained within the ACCRETE control volume. Instead, the results of the HTUDIF calls are used to calculate the corrected heat rate to the surface (the basis of *HNCeff*) and the corresponding diffusion-limited mass flow rate (*EmdotDif*), based on the equations presented in Appendix B.5. At high relative humidity, the film coefficient predicted by NCHCT could be used directly, but at low humidity it should be reduced.

Comparison of *EmdotDif* with the predicted accumulation rate *EmdotFrz* shows that, except for the first few seconds, the freeze rate is less than the diffusion limit (*EmdotFrz*>*EmdotDif*). The assumption of a constant freeze point is therefore valid since diffusion is not limiting frost accumulation.

At low humidities, however, the diffusion rate will become a limiting factor, and the surface temperature will drop below freezing. Unfortunately, neither HTUDIF nor ACCRETE would be directly applicable in that case, since HTUDIF assumes a liquid phase present (however thin) and ACCRETE assumes a constant freeze point temperature. Methods for dealing with diffusion-limited and variable freeze point ice accretion are beyond the scope of this document.

Table 7-3 Complete Input File for Pipe Frost Accretion

```
header options data
title ice build up on horizontal pipe
  output = ice_pipe.out
  save = ice_pipe.sav

header control data,global
   uid = si,  sigma = sbconsi
   abszro = 0.0
   timend = duration
   matmet = 12
c for better properties:
cinsert f6070_water.inc

header node data, frost
    DIV 1,TN2,a200,Volum1*frostDen,A400,0.5*VolumSS*SSden
    SIV 2,TN2,a200,Volum2*frostDen
    SIV 3,TN2,a200,Volum3*frostDen
    SIV 4,TN2,a200,Volum4*frostDen
    SIV 5,TN2,a200,Volum5*frostDen
    SIV 6,TN2,a200,Volum6*frostDen
    SIV 7,TN2,a200,Volum7*frostDen
```

* In Table 7-3, the NCHCT call is commented out and replaced, such that the user can verify that a lower *Hextcon* coefficient would render invalid the key assumption of this analysis.

```
    SIV 8,TN2,a200,Volum8*frostDen
    SIV 9,TN2,a200,Volum9*frostDen
    SIV 10,TN2,a200,Volum10*frostDen
    -100,TN2,0.0    $ LN2
    11,TN2,-1.0     $ surface
    -200,Tair,0.0   $ Environ. air

header conductor data,frost
    SIV 100,100,1,a300,GtermSS
    SIV 1,1,2,a100,Gterm1
    SIV 2,2,3,a100,Gterm2
    SIV 3,3,4,a100,Gterm3
    SIV 4,4,5,a100,Gterm4
    SIV 5,5,6,a100,Gterm5
    SIV 6,6,7,a100,Gterm6
    SIV 7,7,8,a100,Gterm7
    SIV 8,8,9,a100,Gterm8
    SIV 9,9,10,a100,Gterm9
    SIV 10,10,11,a100,Gterm10
c convection/diffusion from air to OD
       200,200,11, SurfArea*HNCeff
c radiation to OD
       -201,200,11, SurfArea*emissIce
c advection
       202,200,11, CPvap*EmdotFrz

insert air8c.inc
header flow data,wetair,fida=8729,fidw=6070
lu def, pl!= 101325.
       tl = Tair
       xl = 1.0
       xgw = 0.00001
       xga = 0.99999
lu plen,1

header registers
       Xallow = 0.01       $ allow 10mm max ... if hits this, raise it.
       Tair = 300.0
       TN2 = 80.0
       relhum = 1.0  $ at less than 100% rel humidity, surface drops in temp
       Hextcon = 100.0     $ set in Logic: external convection coef.
       Hfilmc = 60000.0    $ dropwise, guestimate W/m2-K
       Heff = 1.0          $ see HTUDIF call in logic
       HNCeff = 1.0        $ see HTUDIF call in logic
       Diam = 0.03
       Ttriple = 273.15
       Tinter = 0.0
       CPvap = 0.0
       EmdotDif = 0.0
       EmdotFrz = -aqrate/Hsublim
       SurfArea = pi*(Diam+2*Xleft)*Length
       frostDen = 100.0  $ frost density kg/m3. Guessed! changes with time
       iceDen = 920.0      $ ice density at 0C
       Length = 1.0        $ unit length
C
c preprocessor can't survive random values for thicknesses,
c so use CHGREG in OPERATIONS.
c      Rad10 = Rad9 + frost.a(1+1)
c      Rad9 = Rad8 + frost.a(1+2)
```

```
c      Rad8 = Rad7 + frost.a(1+3)
c      Rad7 = Rad6 + frost.a(1+4)
c      Rad6 = Rad5 + frost.a(1+5)
c      Rad5 = Rad4 + frost.a(1+6)
c      Rad4 = Rad3 + frost.a(1+7)
c      Rad3 = Rad2 + frost.a(1+8)
c      Rad2 = Rad1 + frost.a(1+9)
c      Rad1 = Rad0 + frost.a(1+10)
c
       Rad10 = Rad9 + 1.e-6
       Rad9 = Rad8 + 1.e-6
       Rad8 = Rad7 + 1.e-6
       Rad7 = Rad6 + 1.e-6
       Rad6 = Rad5 + 1.e-6
       Rad5 = Rad4 + 1.e-6
       Rad4 = Rad3 + 1.e-6
       Rad3 = Rad2 + 1.e-6
       Rad2 = Rad1 + 1.e-6
       Rad1 = Rad0 + 1.e-6
       Rad0 = Diam/2
       RadID = Rad0 - SSthick

       SSthick = Diam*0.1
       SScond = 12.0
       SScp = 460.
       SSden = 7800.
c
       VolumSS = pi*(rad0^2-radID^2)*Length
       Volum1 = pi*(rad1^2-rad0^2)*Length
       Volum2 = pi*(rad2^2-rad1^2)*Length
       Volum3 = pi*(rad3^2-rad2^2)*Length
       Volum4 = pi*(rad4^2-rad3^2)*Length
       Volum5 = pi*(rad5^2-rad4^2)*Length
       Volum6 = pi*(rad6^2-rad5^2)*Length
       Volum7 = pi*(rad7^2-rad6^2)*Length
       Volum8 = pi*(rad8^2-rad7^2)*Length
       Volum9 = pi*(rad9^2-rad8^2)*Length
       Volum10 = pi*(rad10^2-rad9^2)*Length
c
       GtermSS=2.*pi*Length/ln(rad0/radID)
       Gterm1 =2.*pi*Length/ln(rad1/rad0)
       Gterm2 =2.*pi*Length/ln(rad2/rad1)
       Gterm3 =2.*pi*Length/ln(rad3/rad2)
       Gterm4 =2.*pi*Length/ln(rad4/rad3)
       Gterm5 =2.*pi*Length/ln(rad5/rad4)
       Gterm6 =2.*pi*Length/ln(rad6/rad5)
       Gterm7 =2.*pi*Length/ln(rad7/rad6)
       Gterm8 =2.*pi*Length/ln(rad8/rad7)
       Gterm9 =2.*pi*Length/ln(rad9/rad8)
       Gterm10 =2.*pi*Length/ln(rad10/rad9)
c
       Hfus = 334.e3$ J/kg
       HFG = 2.5e6$ J/kg ... will call VHFG in OPERATIONS
       Hsublim = HFG + Hfus
       HRA = Hsublim*iceDen*SurfArea
       Xleft = 0.0
       DTfact = 0.0
       Duration = 3600.0
       emissIce = 0.9
```

```
header array data, frost
c from Cryogenic Heat Transfer, Barron 1999, P62. Table 203
c conductivity of frost, with estimated ice value used at 273K
  100 =80.,   162.7e-3
       90.,   98.3e-3
       100.,  65.1e-3
       110.,  40.5e-3
       120.,  23.9e-3
       130.,  20.8e-3
       140.,  23.4e-3
       160.,  41.4e-3
       180.,  59.2e-3
       200.,  77.0e-3
       273.,  1.88*frostDen/IceDen
c ice Cp ... from www.engineertoolbox.com
  200 = 173.,1.389e3
       223.,  1.751e3
       273.,  2.050e3
c Stainless K (Barron Fig 1-4)
  300 = 4., 0.1
       10., 0.8
       20., 2.
       40., 5.
       80., 8.
       100.,9.
       200.,13.
       400.,17.
c Stainless Cp (just need for DIV function)
  400 = 100., SScp
        300., SScp
c Thickness array:
   1 = allsame,10,1.0e-6

header output calls, frost
       call nodtab('all')
       call gprint('all')
       call regtab
       call save('all',0)

header variables 2, frost
       call accrete('frost',-10,10,frost.a1,Ttriple,HRA,Xleft,DTfact)
       call accqrate(aqrate)

header operations
build all
buildf all
defmod wetair
c
c set register expressions. (Not needed if use thin-walled approx.)
c Couldn't withstand unknown processor variables in preprocessor:
c
       call chgreg('Rad10','Rad9 + frost.a(1+1)')
       call chgreg('Rad9','Rad8 + frost.a(1+2)')
       call chgreg('Rad8','Rad7 + frost.a(1+3)')
       call chgreg('Rad7','Rad6 + frost.a(1+4)')
       call chgreg('Rad6','Rad5 + frost.a(1+5)')
       call chgreg('Rad5','Rad4 + frost.a(1+6)')
       call chgreg('Rad4','Rad3 + frost.a(1+7)')
```

```
       call chgreg('Rad3','Rad2 + frost.a(1+8)')
       call chgreg('Rad2','Rad1 + frost.a(1+9)')
       call chgreg('Rad1','Rad0 + frost.a(1+10)')

       vtest  = vsv(vps(Ttriple,fi6070),Ttriple,fi6070)
       HFG    = vhfg(vps(Ttriple,fi6070),Ttriple,vtest,fi6070)
       CPvap  = vcpv(vps(Ttriple,fi6070),0.5*(Ttriple+Tair),fi6070)

c if 1cm thick total, max.: 0.01 = 0.001*10 layers
       call recess_set('thkmax',Xallow/10.)
c  10% of THKMAX for min. starting thickness
       call recess_set('thklast',0.1*Xallow/10.)
       call layinit('frost',-10,10,frost.a1,0.0)
       Xleft  = 0.0
       call upreg
c
c get coef for dry air:
c      CALL NCHCT(Hextcon, utest, dtest ,Diam, pl1, tl1, 1.0,
c     +     Ttriple, fi8729)
c too low! use forced convection estimate instead:
       Hextcon = 100.0

       call humr2x('wetair',1,relhum,xtest)
       call chglmp('wetair',1,'xgw',xtest,'pl')
       call prepmc('wetair',1)

       CALL HTUDIF (Heff,Hfilmc,Hextcon,Ttriple,pl1,tl1,wetair.FI)
       Tinter = (Heff/Hfilmc)*(tl1-Ttriple) + Ttriple
       HNCeff = Hextcon*(tl1-Tinter)/(Tair-Ttriple)
C      m"*hfg = Hfilmc*(Tinter-Ttriple) - Hextcon*(tl1-Tinter)
C      m"*hfg = Heff*(tl1-Ttriple) - Hextcon*(tl1-Tinter)
       etest  = Heff*(tl1-Ttriple) - Hextcon*(tl1-Tinter)
       EmdotDif= SurfArea*etest/hfg
       call transient

endofdata
```

The duration of the ice-building event is extremely slow compared to the time constant of the ice layer: identical answers can be found in a fraction of the run time by using arithmetic nodes instead of diffusion nodes, as shown in Table 7-4. If more arithmetic layers (Nlayer>10) were used, the thin-walled cylindrical conduction approximation would become increasingly valid, which would considerably simplify the model.

Table 7-4 Alternate NODE DATA for Arithmetic Ice Stack

```
header node data, frost
    SIV 1,TN2,A400,0.5*VolumSS*SSden
      gen,2,9,1,TN2,-1.0
    -100,TN2,0.0  $ LN2
    11,TN2,-1.0   $ surface
    -200,Tair,0.0 $ Environ. air
```

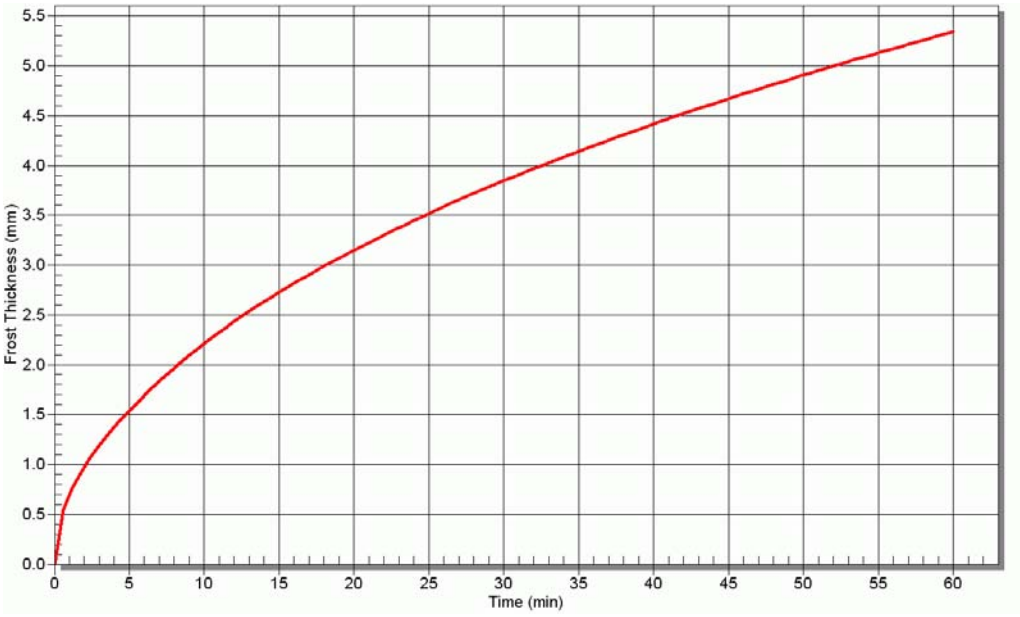


Figure 7-4 Frost Thickness versus Time

7.7.12.2 Preliminary Access for Thermal Desktop Users

A user interface for ACCRETE is not yet available in Thermal Desktop, but the existence of the RECESS routine can be exploited. To use ACCRETE in Thermal Desktop, add a recession material instead, then *globally* replace all RECESS calls with ACCRETE calls by inserting the following into SUBROUTINES:

```
(HEADER SUBROUTINES)
fstart
      subroutine recess(name,int1,int2,real1,real2,real3,real4,real5)
      character *(*) name
      call accrete(name,int1,int2,real1,real2,real3,real4,real5)
      return
      end
fstop
```

Users of the Intel Visual Fortran (IVF) product will need to add "/FORCE" to the link command found in the four infon*.lnk files in the /bin subdirectory of the SINDA/FLUINT install location. If you re-install SINDA/FLUINT, you will need to edit these link command files again as well.

To initialize the layers to zero thickness, make a preliminary run, then copy the generated RECESS calls in VARIABLES 2 from the *.cc file, and paste them into OPERATIONS before a transient is called. Rename "RECESS" to "LAYINIT" using an editor, and strip the last 3 arguments off, replacing them with a zero value (for zero thickness).

For example, convert:

```
call recess('frost',-10,10,frost.a1,Ttriple,HRA,Xleft,DTfact)
```

to be:

```
call layinit('frost',-10,10,frost.a1,0.0)
```

7.7.13 Standard Atmospheric Temperature and Pressure

Subroutine Name: STDATMOS

Description: This routine can be used to calculate density, temperature, and pressure changes with altitude based on the 1976 US Standard Atmosphere.* Perfect gas behavior is assumed. The reference temperature at sea level is 15°C (288.15K, 518.67R).

The units of altitude are always kilometer (km), independent of the choice of UID in CONTROL DATA, GLOBAL. The resulting temperature and pressure ratios should be applied to absolute units (e.g., K, R, psia).

Restrictions: No input checks are made. Values of altitude beyond 86km should not be used.

Calling Sequence:

```
CALL STDATMOS (Alt, Dratio, Pratio, Tratio)
```

where:

`Alt` ..... input altitude above sea level, in kilometers (real).
`Dratio` .... output ratio of density to that of sea level (real)
`Pratio` .... output ratio of absolute pressure to that of sea level (real)
`Tratio` .... output ratio of absolute temperature to that of sea level (real)

Example:

```
CALL STDATMOS (1.6093, dtest, ptest, ttest)  $ Alt is mile high
Temp = 518.67*ttest + abszro $ “Temp” in degrees F (UID=ENG here)
Pres = 14.7*ptest        $ “Pres” in psia
```

Guidance: The ratio of speed of sound to that of sea level can be calculated as SQRT(Tratio), assuming a perfect gas.

* U.S. Standard Atmosphere, 1976, U.S. Government Printing Office, Washington, D.C., 1976. See http://nssdc.gsfc.nasa.gov/space/model/atmos/us_standard.html

7.8 Arithmetic Subroutines

Caution Most of the routines documented in this section apply to data values, "A(DV)," within arrays, and not to arrays themselves, "A(IC)." This means they should normally reference the first data cell in the array, not the array itself. In other words, these routines have no knowledge of the SINDA storage format, and just operate on sequential array cells as if they all contained the same type of data. *They should not be used with bivariate or trivariate arrays.* See also Section 2.11.4.

7.8.1 Add Two Arrays

Subroutine Name: ADDARY

Description: This subroutine will add the corresponding elements of two specified length arrays to form a third array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ADDARY (N, AI1(DV), AI2(DV), AO(DV))
```

where:

`N` ..... integer size of arrays
`AIn` ..... input arrays
`AO` ..... output array

Examples:

```
CALL ADDARY (NTEST, A(101+1), A(102+1), A(103+1))
CALL ADDARY (33, SUM1(1), SUM2(1), RESULT(1))
```

7.8.2 Add a Constant to Elements of an Array

Subroutine Name: ARYADD

Description: This subroutine will add a constant value to every element of an array to form a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ARYADD (N, AI(DV), C, AO(DV))
```

where:

N. . . . . integer size of arrays
AI . . . . . input array
C. . . . . a floating point constant
AO . . . . . output array

7.8.3 Divide Corresponding Elements of Arrays

Subroutine Name: DIVARY

Description: This subroutine will divide the corresponding elements of one array into another array to form a third array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL DIVARY (N, AI1(DV), AI2(DV), AO(DV))
```

where:

N. . . . . integer size of arrays
AIn . . . . . input arrays
AO . . . . . output array

7.8.4 Divide a Constant into Elements of an Array

Subroutine Name: ARYDIV

Description: This subroutine will divide a constant value into every element of an array to form a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ARYDIV (N, AI(DV), C, AO(DV))
```

where:

N..... integer size of arrays
AI..... input array
C..... a floating point constant
AO..... output array

7.8.5 Multiply Two Arrays

Subroutine Name: MPYARY

Description: This subroutine will multiply the corresponding elements of two specified length arrays to form a third array.

Restriction and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL MPYARY (N, AI1(DV), AI2(DV), AO(DV))
```

where:

N..... integer size of arrays
AIn ..... input arrays
AO..... output array

7.8.6 Multiply the Elements of an Array by a Constant

Subroutine Name: ARYMPY

Description: This subroutine will multiply every element of an array by a constant value to form a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ARYMPY (N, AI(DV), C, AO(DV))
```

where:

N. . . . . integer size of arrays
AI . . . . . input array
C. . . . . a floating point constant
AO . . . . . output array

7.8.7 Subtract Corresponding Elements Of Arrays

Subroutine Name: SUBARY

Description: This subroutine will subtract the corresponding elements of one array from another array to form a third array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL SUBARY (N, AI1(DV), AI2(DV), AO(DV))
```

where AO(DV) = AI1(DV) - AI2(DV) and:

N. . . . . integer size of arrays
AIn . . . . . input arrays
C. . . . . a floating point constant
AO . . . . . output array

7.8.8 Subtract a Constant from Elements of an Array

Subroutine Name: ARYSUB

Description: This subroutine will subtract a constant value from every element of an array to form a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ARYSUB (N, AI(DV), C, AO(DV))
```

where AO(DV) = AI1(DV) - C and:

N . . . . . integer size of arrays
AI . . . . . input array
C . . . . . a floating point constant
AO . . . . . output array

7.8.9 Sum an Array of Floating Point Values

Subroutine Name: SUMARY

Description: This subroutine will sum the specified elements of an array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer.

Calling Sequence:

```
CALL SUMARY (N, AI(DV), C)
```

where:

N . . . . . integer size of array
AI . . . . . input array
C . . . . . a floating point output sum

7.8.10 Invert Array Elements

Subroutine Name: ARYINV

Description: This subroutine will invert the specified elements of an array in place.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array will be overlaid into the input array.

Calling Sequence:

```
CALL ARYINV (N, AIO(DV))
```

where:

N. . . . . integer size of array
AIO . . . . . input/output array

7.8.11 Divide an Array into a Constant

Subroutine Name: ARINDV

Description: This subroutine will divide every element of an array into a constant value to form a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL ARINDV (N, AI(DV), C, AO(DV))
```

where AO(DV) = C/AI(DV) and:

N. . . . . integer size of arrays
AI . . . . . input array
C. . . . . a floating point constant
AO . . . . . output array

7.8.12 Inverse of Sum of Inverses

Subroutine Name: ADARIN

Description: This subroutine calculates one over the sum of inverses of an array of values. This subroutine is useful for calculating the effective conductance of series conductors.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer.

Calling Sequence:

```
CALL ADARIN (N, AI(DV), C)
```

where:

N . . . . . integer size of array
AI . . . . . input array
C . . . . . a floating point output

7.8.13 Array Difference and Multiplication

Subroutine Name: QFORCE

Description: This subroutine multiplies an array by the difference between two arrays to make a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL QFORCE (N, AI1(DV), AI2(DV), AI3(DV), AO(DV))
```

where AI4(DV) = AI3(DV) * (AI1(DV) - AI2(DV)) and:

N . . . . . integer size of arrays
AIn . . . . . input arrays
AO . . . . . output array

7.8.14 Floating Difference and Multiply

Subroutine Name: QMETER

Description: This subroutine multiplies a floating point number by the difference of two other floating point numbers.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The answer may be overlaid into one of the input variables.

Calling Sequence:

```
CALL QMETER (C, D, B, A)
```

where A = B * (C - D) and A through D are floating point constants or variables. A must be a variable.

7.8.15 Array Difference and Multiply

Subroutine Name: QMTRI

Description: This subroutine multiplies an array by the difference between adjacent elements of another array to make a new array.

Restrictions and Guidance: All data values to be operated on must be floating point numbers. The array length N must be an integer. The answer array may be overlaid into one of the input arrays.

Calling Sequence:

```
CALL QMTI (N, AI1(DV), AI2(DV), AO(DV))
```

where AO(DV) = AI2(DV) * (AI1(DV) - AI1(DV+1)) and:

N. . . . . integer size of arrays
AIn. . . . . input arrays
AO . . . . . output array

7.8.16 Floating Point Calculation

Subroutine Name: RDTNQS

Description: This subroutine adds 460.0 to two floating point numbers, takes each to the 4th power, does a difference and multiplies this by a third floating point number

Restrictions and Guidance: All data values to be operated on must be floating point numbers.

Calling Sequence:

```
CALL RDTNQS (D, C, B, A)
```

where A = B * ((C+460.0)**4 - (D+460.0)**4) and A through D are floating point constants or variables. A must be a variable.

7.9 Co-solved (Auxiliary) Differential Equations

Many of SINDA/FLUINT's network elements (diffusion nodes, tanks, tubes, etc.) correspond to differential equations in time, customized for heat transfer and fluid flow problems. Occasionally, the user must also include other differential equations into the solution set, including those describing transient mechanical translations and rotations as well as certain control systems.

Utilities are available to include co-solve differential equations, to customize their time-step control, and to generate text output:

DIFFEQ1 . . . . First order equation integrator: B*dX/dt = C*X + D
DIFFEQ1I. . . . First order equation initialization
DIFFEQ1G . . . First order equation: fetch current derivative
DIFFEQ2 . . . . Second order equation integrator: $A*d^2X/dt^2 = B*dX/dt + C*X + D$
DIFFEQ2I. . . . Second order equation initialization
DIFFEQ2G . . . First order equation: fetch current first and second derivatives
DIFFEQSET . . Customize time step controls
DEQNTAB . . . Tabulate current equations

The term *auxiliary* means that these equations are co-solved *along with* the thermal/fluid solutions, but these equations are not solved *implicitly* together. Specifically, there is a one time step lag between the update of the thermal/fluid solution (using the previous value X_i in the i^{th} differential equation) and the update of the differential equation (likely using the results of the thermal/fluid solution). Both the thermal/fluid solution and the differential equation solutions track and automatically adjust time steps for stability and accuracy. In general, close agreement with closed-form solutions is achieved. Often, the lag increases stability. However, the user should note that the possibility exists for tightly coupled thermal/fluid/auxiliary solutions to diverge and become unstable, just as arbitrary user logic adjustments are not always tolerated.

The differential equation solutions can affect the overall time step taken by the model. Two conditions are checked: the rate of change of X (based on the dX/dT at the start of the time step) and the estimation of truncation error (based on the performance of the *previous* time step*). The user can control how these decisions are made (using the DIFFEQSET utility) and can see the results of these decisions as part of the output of the routine DEQNTAB.

Each equation† is assigned a unique identifier by the user and this identifier should be used consistently in subsequent calls: each time a new identifier is detected a new independent equation is created. Thus, these identifiers are analogous to node or lump identifiers. All equations must be uniquely identified; first and second order equations cannot share the same identifier.

---

* If some sudden change were to occur that induced a large error, that error will remain uncorrected and will cause subsequent time steps to be reduced. The differential equation integration is performed after a thermal/fluid time step has been completed, and therefore cannot invoke a back-up to redo the previous time step. As with any unforeseeable sudden changes to parameters in logic, the user should cap the time step during such events.

† Currently, up to 1000 differential equations may be added.

7.9.1 First Order Differential Equations

Equations of the form:

$$B\frac{dX}{dt} = CX + D$$

can be co-solved, where X is a function of time t and perhaps other variables. The coefficients B, C, and D are assumed to be constant over a single time step, but need not be constant otherwise: they can depend on the thermal/fluid solution and vary each time step. Similarly, the resulting value X may be used within the thermal/fluid solution.

This equation is useful for solving equations such as m*dV/dt = F_{net} (Newton's law for velocity), or the rotational equivalent I*dω/dt = T_{net}.

To integrate X numerically, the initial X must be known. Furthermore, the initial dX/dt, if known, helps initiate time step control on the first and second steps and improves accuracy (since a second order implicit integration method is used). Therefore, two routines are available: one to set the initial conditions (or to reset the current point), and one to perform a time step integration: DIFFEQ1I and DIFFEQ1, respectively. DIFFEQ1I is normally called from OPERATIONS, whereas DIFFEQ1 must be called from either FLOGIC 2 or VARIABLES 2. If the call to DIFFEQ1I is omitted, the initial X and dX/dt are assumed zero.

The current values of X and its derivative (dX/dt) may be fetched using DIFFEQ1G any time after an equation has been defined.

Subroutine Name: DIFFEQ1I

Description: DIFFEQ1I initializes a new first order differential equation, or resets an existing one. It is used to set the current point and derivative with respect to time. If this routine is omitted, DIFFEQ1 will begin integration assuming both the initial point and derivative are zero.

This routine is normally called in OPERATIONS, but may be called any time the current point needs to be changed.

Calling Sequence:

```
CALL DIFFEQ1I(ID, XI, DXDTI)
```

where:

ID.....	Unique integer identifier for this equation. This ID should be matched in subsequent DIFFEQ1 calls, as well as repeated calls to DIFFEQ1I affecting the same equation. ID must be nonzero. Each unique value will generate a new equation.
XI.....	The input real initial (or current) value of X (perhaps equal to -D/C)
DXDTI.....	The input real initial (or current) value of dX/dt, the derivative with respect to time.

Subroutine Name: DIFFDP1I

Description: DIFFDP1I is the double precision version of DIFFEQ1I. This is an obsolete option: use DIFFEQ1I instead.

Subroutine Name: DIFFEQ1

Description: DIFFEQ1 performs the integration of the first order differential equation:

$$B\frac{dX}{dt} = CX + D$$

Each call to DIFFEQ1 integrates over the last time step, returning the current value of X. The initial value of X (as well as dX/dt) was assumed to have been set either in the prior call to DIFFEQ1 for the previous time step, or in a call to DIFFEQ1I.

This routine *must* be called from either FLOGIC 2 or VARIABLES 2. No action is taken if called during steady states: the initial condition (as set in DIFFEQ1I) is simply returned as X.

Calling Sequence:

```
CALL DIFFEQ1(ID, X, B, C, D)
```

where:

ID.....	Unique integer identifier for this equation. This ID should be matched in subsequent DIFFEQ1 calls, as well as in any prior calls to DIFFEQ1I affecting the same equation. ID must be nonzero. Each unique value will generate a new equation.
X.....	The returned (next) real value of X
B.....	The input real dX/dt coefficient (see above equation). B must be nonzero.
C.....	The input real X coefficient (see above equation)
D.....	The input real “constant” coefficient (see above equation)

Subroutine Name: DIFFDP1

Description: DIFFDP1 is the double precision version of DIFFEQ1. This is an obsolete option: use DIFFEQ1 instead.

Subroutine Name: DIFFEQ1G

Description: DIFFEQ1I fetches the current values for an *existing*[*] first order differential equation if this information is required by the user, otherwise its use is completely optional.

This routine may be called any time after the appropriate equation has been defined by a DIFFEQ1 or DIFFEQ1I call.

Calling Sequence:

```
CALL DIFFEQ1G(ID, XC, DXDTC)
```

where:

`ID` . . . . . Unique integer identifier for requested equation, as defined by prior DIFFEQ1 or DIFFEQ1I call.

`XC` . . . . . (Returned.) The current real value of X

`DXDTC` . . . . . (Returned.) The current real value of dX/dt, the derivative with respect to time.

Subroutine Name: DIFFDP1G

Description: DIFFDP1G is the double precision version of DIFFEQ1G. This is an obsolete option: use DIFFEQ1G instead.

* Invocation before the corresponding equation has been defined will result in a fatal error.

7.9.1.1 Example: Pump Start-up

A liquid water loop consists of 2.5cm diameter line that is 1000 meters long. A centrifugal pump completes the circuit, and is initially off (zero shaft speed). A head-flow rate curve is available for the pump at a reference speed of 1000 rpm, and the pump similarity rules are applied for other speeds (using a FLUINT PUMP). Assume that "speed" is a register and therefore RSPD=1000 and SPD=speed for the PUMP connector.*

At time zero, power is applied to the pump. The pump must overcome both resistance and inertia in the line (which is covered by FLUINT tubes), as well as the slight compressibility of the liquid (which is covered by the COMP factor in FLUINT tanks, perhaps as supplied via the COMPLQ utility).

However, the pump must also overcome the rotational inertia of its vanes (and the liquid between them) as well as the windings (if an electric motor), and perhaps shaft and gear box (if a mechanical motor), friction, etc. These latter effects will be included via a co-solved first order differential equation.

Assume that the raw mechanical torque available to the pump is 13 N-m at zero speed (register "Traw = 13"), and that it degrades linearly to 10 n-m at 1000 rpm. A new register can be then set describing the available mechanical torque:

Tmech = Traw - 0.003*speed

Friction losses within the pump itself contribute 2 N-m of torque at 1000 rpm:

Tfrict = 0.002*speed

As the pump spins up, an assumption will be made that the head-flow rate curve is obeyed at every speed: it will be used to predict off-design performance lacking better data. The hydraulic torque produced by the pump will be estimated based on the current flow rate and pressure drop. If path 1 in fluid submodel "water" represents the pump, lump 1 the inlet, lump 100 the outlet, and Rimp is a register containing the outer blade radius of the pump, then a crude estimate for the hydraulic torque is:

Thydr = water.fr1*sqrt(2.0*max(0.,water.pl100-water.pl1)/water.dl1)*Rimp

Alternatively, if the efficiency EFFP of the pump were known, this could be specified as an input and the program would calculate the hydraulic torque as the variable TORQ1:

Thydr = water.torq1

Thus, the net torque on the pump available for accelerating or decelerating it is:

Tnet = Tmech - Tfrict - Thydr

* The pump similarity rules lead to infinite flow resistance at zero speed. To overcome this, a LOSS element can be placed in parallel with the PUMP. The K-factor for this loss should increase quickly as speed increases and the similarity rules take over. For example: FK = Kinit + min(1.0E10,speed**3).

If the rotational inertia of the pump/blade/windings/shaft etc. can be estimated as “Inert” (a real register), then the speed of the pump is given by:

$$Inert*d\omega/dt = Inert*d(speed/2\pi)/dt = Tnet$$

The above is, of course, a first order differential equation with respect to time. This equation will be called equation #1. To initialize it, a call is made in OPERATIONS before calling TRANSIENT:

```
CALL DIFFEQ1I( 1, 0., 0.)
```

If time steps are too small as the pump starts from zero, the XLOW parameter can be increased to reasonable rpm values (say, 1 rpm, versus the default 1.0e-5 rpm) using the DIFFEQSET routine:*

```
CALL DIFFEQSET('XLOW',1.0)
```

Then, assuming that a register “twopi” has be set to “2*pi”, within FLOGIC 2:

```
CALL DIFFEQ1( 1, stest, Inert, 0.0, Tnet)
speed = stest*twopi
```

Recall that this resulting speed value was used to define the SPD parameter of a PUMP connector that is being solved concurrently within the FLUINT submodel.

Note that the C term (coefficient times current value of X, or speed coefficient) is zero. This means that the Tnet term changes each time step as the speed changes. While this is legal, it is desirable to shift some of the D term into the C term: $Inert*d(speed/2\pi)/dt = C*(speed/2\pi) + D$.

While a bit more elaborate, this effort is rewarded by larger time steps:

```
c collect friction (0.002) plus degradation in power (0.003), and
c convert to radians/sec:
      ctest = -0.005*twopi
c remaining term is the raw mechanical torque less hydraulic:
      dtest = Traw - Thydr
c now call for integration
      CALL DIFFEQ1( 1, stest, Inert, ctest, dtest)
      speed = stest*twopi
```

Note that while the hydraulic torque term also depends on “speed,” this dependence is difficult to extract and so is it is left in the D term, which is modified each time step as the flow rate and pressure differential change as a result of the concurrent FLUINT solution.

* Note that if there were multiple differential equations, the XLOW value would have been reset for all of them: only one such global value is available.

7.9.2 Second Order Differential Equations

Equations of the form:

$$A\frac{d^2X}{dt^2} = B\frac{dX}{dt} + CX + D$$

can be co-solved, where X is a function of time t and perhaps other variables. The coefficients A, B, C, and D are assumed to be constant over a single time step, but need not be constant otherwise: they can depend on the thermal/fluid solution and vary each time step. Similarly, the resulting value X may be used within the thermal/fluid solution.

This equation is useful for solving equations such as $m*d^2x/dt^2 = F_{net}$ (Newton's law for position), or the rotational equivalent $I*d^2\theta/dt^2 = T_{net}$.

To integrate X numerically, the initial X as well as dX/dt must be known. Furthermore, the initial d^2X/dt^2, if known, helps initiate time step control on the first and second steps and improves accuracy (since a third order implicit integration method is used). Therefore, two routines are available: one to set the initial conditions (or to reset the current point), and one to perform a time step integration: DIFFEQ2I and DIFFEQ2, respectively. DIFFEQ2I is normally called from OPERATIONS, whereas DIFFEQ2 must be called from either FLOGIC 2 or VARIABLES 2. If the call to DIFFEQ2I is omitted, the initial X, dX/dt, and d^2X/dt^2 are assumed zero.

The current values of X and its derivatives (dX/dt, d^2X/dt^2) may be fetched using DIFFEQ2G any time after an equation has been defined.

Subroutine Name: DIFFEQ2I

Description: DIFFEQ2I initializes a new second order differential equation, or resets an existing one. It is used to set the current point, first derivative with respect to time, and second derivative with respect to time. If this routine is omitted, DIFFEQ2 will begin integration assuming the initial point and derivatives are all zero.

This routine is normally called in OPERATIONS, but may be called any time the current point needs to be changed.

Calling Sequence:

```
CALL DIFFEQ2I(ID, XI, DXDTI, D2XDT2I)
```

where:

`ID` . . . . . Unique integer identifier for this equation. This ID should be matched in subsequent DIFFEQ2 calls, as well as repeated calls to DIFFEQ2I affecting the same equation. ID must be nonzero. Each unique value will generate a new equation.

`XI` . . . . . The input real initial (or current) value of X (perhaps equal to -D/C)

`DXDTI` . . . . . The input real initial (or current) value of dX/dt, the derivative with respect to time.

`D2XDT2I` . . . . The input real initial (or current) value of d^2X/dt^2, the second derivative with respect to time.

Subroutine Name: DIFFDP2I

Description: DIFFDP2I is the double precision version of DIFFEQ2I. This is an obsolete option: use DIFFEQ2I instead.

Subroutine Name: DIFFEQ2

Description: DIFFEQ2 performs the integration of the second order differential equation:

$$A\frac{d^2X}{dt^2} = B\frac{dX}{dt} + CX + D$$

Each call to DIFFEQ2 integrates over the last time step, returning the current value of X. The initial value of X (as well as dX/dt and d^2X/dt^2) was assumed to have been set either in the prior call to DIFFEQ2 for the previous time step, or in a call to DIFFEQ2I.

This routine *must* be called from either FLOGIC 2 or VARIABLES 2. No action is taken if called during steady states: the initial condition (as set in DIFFEQ2I) is simply returned as X.

Calling Sequence:

```
CALL DIFFEQ2(ID, X, A, B, C, D)
```

where:

ID..... Unique integer identifier for this equation. This ID should be matched in subsequent DIFFEQ2 calls, as well as in any prior calls to DIFFEQ2I affecting the same equation. ID must be nonzero. Each unique value will generate a new equation.

X..... The returned (next) real value of X

A..... The input real d^2X/dt^2 coefficient (see above equation). A must be nonzero.

B..... The input real dX/dt coefficient (see above equation).

C..... The input real X coefficient (see above equation)

D..... The input real "constant" coefficient (see above equation)

Subroutine Name: DIFFDP2

Description: DIFFDP2 is the double precision version of DIFFEQ2. This is an obsolete option: use DIFFEQ2 instead.

Subroutine Name: DIFFEQ2G

Description: DIFFEQ2I fetches the current values for an *existing** second order differential equation if this information is required by the user, otherwise its use is completely optional.

This routine may be called any time after the appropriate equation has been defined by a DIFFEQ2 or DIFFEQ2I call.

Calling Sequence:

```
CALL DIFFEQ2G(ID, XC, DXDTC, D2XDT2C)
```

where:

ID..... Unique integer identifier for requested equation, as defined by prior DIFFEQ2 or DIFFEQ2I call.

XC..... (Returned.) The current real value of X

DXDTC..... (Returned.) The current real value of dX/dt, the derivative with respect to time.

D2XDT2C... (Returned.) The current real value of d^2X/dt^2, the second derivative with respect to time.

* Invocation before the corresponding equation has been defined will result in a fatal error.

One common usage of DIFFEQ2G is to fetch the first derivative of a second order equation: dX/dt. If a piston is modeled and the position X is the equation parameter, dX/dt may also be needed in order to set the VDOT of a corresponding FLUINT tank: VDOT = dX/dt * A_p where A_p is the piston cross-sectional area. In such a case, calls to routine DIFFEQ2 would be followed by an immediate call to DIFFEQ2G (as DIFFDP2 would be followed by DIFFDP2G):

```
fnet = (PL101 - PL102)*Apiston
CALL DIFFEQ2(101, xtest, totmass, frict, spring, fnet)
CALL DIFFEQ2G(101, xtest, velocity, acceler)
vdot101 = velocity*Apiston
vdot102 = -vdot101
```

Subroutine Name: DIFFDP2G

Description: DIFFDP2G is the double precision version of DIFFEQ2G. This is an obsolete option: use DIFFEQ2G instead.

7.9.2.1 Example: Valve Stem Motion

A fictitious simplified valve is depicted in Figure 7-5. A spring keeps the valve seated until the pressure force overcomes it, then it opens gradually (e.g. a pressure relief valve or a check valve with an offset) and perhaps with some oscillation. The flow resistance of the partially opened valve can be modeled many ways as a function of Xpos: the size of the gap (also the distance of the valve stem from the resting position). The position of the valve will vary transiently as a function of its mass, the spring force, sliding friction, and the pressure differential.

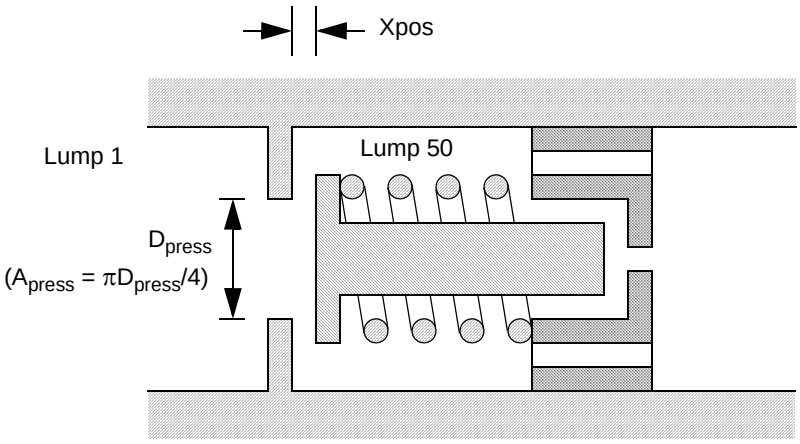


Figure 7-5 Geometry for DIFFEQ2 Valve Stem Example

If the valve area is Apress, the pressure force is ΔP*Apress. If the upstream pressure is lump 1 in submodel “air” and the downstream pressure is lump 50 in the same submodel, then the pressure force may be expressed by the following register:

Fpress = (air.PL1 - air.PL50)*Apress

Similarly, if the spring rate is Kspring and the zero position is Xzero (negative in this case to assure a closing force at zero pressure gradient), then the spring force is:

Fspring = Kspring*(Xpos - Xzero)

The friction force is more difficult to characterize since one must know the current velocity, but if that velocity (d(Xpos)/dt) is known as is the friction coefficient Cfrict (force per velocity):

Ffrict = Cfrict*d(Xpos)/dt

The net force on the stem is then:

Fnet = Fpress - Ffrict - Fspring

The position of the stem whose mass is Mstem is then a second order differential equation:

$$Mstem * d^2Xpos/dt^2 = Fnet$$

It is both convenient and efficient (in terms of larger time steps) to leave Fnet decomposed into its components, which are functions of Xpos and its derivative (velocity):

$$Mstem * d^2Xpos/dt^2 = - Cfrict*d(Xpos)/dt - Kspring*Xpos + (Fpress + Kspring*Xzero)$$

This equation will be called equation #22. To initialize it, a call is made in OPERATIONS before calling TRANSIENT:

```
CALL DIFFEQ2I( 22, Xpos, 0., 0.)$ Xpos = 0.0 initially
```

Then within FLOGIC 2:

```
ptest       = Fpress + Kspring*Xzero
CALL DIFFEQ2( 22, Xpos, Mstem, -Cfrict, -Kspring, ptest)
```

Note that while Mstem, Cfrict, and Kspring are constant, Fpress (and therefore ptest) is a function of pressure. The returned value Xpos is used to calculate the flow resistance and throat area of the valve for the next time step.

The above model does not take into account the fact that Xpos cannot be negative. Similarly, it cannot be too large or it will hit a stop (assume the maximum value of Xpos is given by register Xmax). Such limits can be accounted for by appending the following to the FLOGIC 2 logic immediately following the DIFFEQ2 call:

```
if(Xpos .le. 0.0)then
      Xpos = 0.0
      call diffeq2i(22,Xpos,0.0,0.0)
elseif(Xpos .ge. Xmax)then
      Xpos = Xmax
      call diffeq2i(22,Xpos,0.0,0.0)
endif
```

Note the use of DIFFEQ2I calls to both reset the current value of Xpos and to zero the first and second derivatives as needed to indicate a motionless valve stem.

Also note that the above correction does not model the elastic or perhaps inelastic impact of the stem against its seat or stops, it merely limits the possible range of Xpos.

7.9.3 Customized Time Step Controls

For first order equations, which are integrated using second order methods, the truncation error (as listed in DEQNTAB) is the absolute value of the third order term in the Taylor series normalized to the current value of X: $(d^2X/dt^2)*\Delta t_{last}^2/2X$ where the second derivative is evaluated numerically as a change in dX/dt over the last interval (Δt_{last}). For a second order equation (solved using third order methods), the truncation term is the normalized fourth term in the Taylor series: $(d^3X/dt^3)*\Delta t_{last}^3/6X$.

The time step might be limited based on the truncation error from the *previous* step, as needed to keep the relative truncation error below 0.1% each step. This value may be adjusted using the ETARGET variable (defaulted to 0.001) in DIFFEQSET.

Similarly, a limit on the size of change allowed in the independent variable X is imposed based on the dX/dt value from the previous time step (or as set in an initialization routine). X is allowed to change by no more than 10% each time step, with a lower limit of 1.0e-5 applied to X to prevent diminishingly small time steps when X is very small. These values may be adjusted using the XLOW value (defaulted to 1.0e-5) and the DXSIZE variable (defaulted to 0.1) in DIFFEQSET.

In pseudo-code, the time step control is summarized as follows:

Δt_{max} = min{ max(XLOW,X)*DXSIZE/dXdt , Δt_{last}*ETARGET/ERR_{last}}

Subroutine Name: DIFFEQSET

Description: DIFFEQSET allows the user to reset time step control parameters for all differential equation routines.

Increase XLOW to be a reasonably negligible value of X: a value of X below which a small time step should not be invoked. This is useful if the value X starts from zero or goes to zero. The default XLOW is 1.0e-5. Increasing XLOW will yield larger time steps as X gets diminishingly small, at the expense of poorer accuracy.

Increasing DXSIZE and ETARGET will yield larger time steps at the expense or poorer accuracy. Decreasing these values has the reverse effect. ETARGET is a better determinant of accuracy of the differential equation's integration, whereas DXSIZE exists primarily because the returned value X is often used within the thermal/fluid solution (see examples), which may not tolerate large changes to X each interval.

This routine can be called in any logic block.

Calling Sequence:

```
CALL DIFFEQSET(name, value)
```

where:

name . . . . . The name of the control parameter to be changed, in single quotes:
'DXSIZE'... to change the size of ΔX allowed per time step
'XLOW' ... to change the minimum value of X to be considered in the above
'ETARGET' ... to change the target value for truncation error

value . . . . . The input real value of the parameter to be used for subsequent differential equation controls. Nonzero positive values are required.

Examples:

```
call diffeqset('xlow', 1.0)   $ X beneath 1 is negligible
call diffeqset('ETARGET', 1.0E-4) $ more accuracy is required
```

7.9.4 Output Tabulation

Subroutine Name: DEQNTAB

Description: DEQNTAB tabulates data for all current differential equations, as defined in prior calls to DIFFEQ1I, DIFFEQ1, DIFFEQ2I, and DIFFEQ2. Data tabulated includes the current values of X, dX/dt, d^2X/dt^2 (if a second order equation, otherwise this is zero), as well as the time step limits and last truncation error (see DIFFEQSET). The DT CHG column refers to the DX limit (governed by DXSIZE and XLOW) and the DT ERR column refers to the truncation error time step limit (governed by ETARGET).

If a differential equation controlled the last time step, this will be noted. However, since DEQNTAB is normally called from OUTPUT CALLS and the last time step is often adjusted to hit an output interval (and not due to simulation accuracy criteria), this feature will normally be of limited value. Therefore, when debugging time steps or other problems consider calling DEQNTAB from another location (perhaps FLOGIC 2) or setting the output interval to be each time step (ITEROT or ITROTF=1).

Calling Sequence:

```
CALL DEQNTAB
```

7.10 Root and Minimum Finder Utilities

Additional equations must be often be co-solved along with the SINDA/FLUINT thermohydraulic solution. The final solution might not only need to satisfy energy, mass, and momentum considerations, but also perhaps additional user-imposed conditions. For example, a set of utilities is available for simulating PID controllers (see Section 7.7.8), but they are only marginally applicable to steady-state solutions such as STEADY (aka, "FASTIC"), which is often used for setting initial conditions. Similarly, co-solved ODEs (see Section 7.9) are used for simulating mechanical motions (e.g., valve stems and pump shafts) in transients, but they are not applicable to steady-state solutions. The Solver (see Section 5) is an ultimate solution to such co-solution needs and more (e.g., sizing), but it can take time to learn ... perhaps more time than is warranted if the user is only concerned with simple problems.*

A set of utilities is available to fill this gap. These routines are specifically designed to handle simple co-solved user equations during steady-state solutions, perhaps in preparation for PID and ODE simulations in subsequent transient solutions. These utilities include:

1. A root finder: a utility to solve for x such that F(x) = 0. The parameter x is varied iteratively along with SINDA/FLUINT solutions, with the goal of achieving both convergence in x and in SINDA/FLUINT temperatures, flow rates, etc. by the end of the STEADY call. Many common co-solution needs can be met if posed as finding the root of F(x). For example, x might be the RPM of a turbocharger shaft and F(x) is the net torque (turbine output torque minus compressor input torque). Or x might be the position of a valve stem and F(x) is the difference between a downstream pressure and the desired set point of a pressure regulator.

2. A minimum finder: a utility to solve for x such that F(x) is minimized over some interval in x. Again, the parameter x is co-solved along with SINDA/FLUINT STEADY solutions. As an example, F(x) might be the negative of the efficiency of a turbine (such that minimizing the negative maximizes the efficiency), and x is the setting on a by-pass valve.

If these tasks had been posed to the Solver, then the Solver would have evaluated the function F(x) at the *end* of a converged steady-state solution, estimated a new value x', then invoked another steady-state solution to find the new value of F(x'). The Solver's reliance on converged SINDA/FLUINT solutions allows it to evaluate multiple functions and adjust multiple parameters, and to include complex interrelationships and constraints.

However, it is often convenient to adjust a single parameter *during* a single steady-state solution. This co-solution approach requires some compromises. First, only one or two such parameters can be adjusted without causing unresolvable interactions. Second, the function F(x) might not be known precisely because it is being calculated concurrently with a SINDA/FLUINT solution that has not yet converged. The ramifications of this second compromise can be subtle, as explained next.

* This statement refers to the *user's* time. For some problems, total *computational* time might be reduced using the Solver despite its use of multiple steady-state solutions, since each steady-state solution (with a constant value of x, the *design variable*) will normally require fewer iterations to converge.

The algorithms underlying the root finder and minimum finder converge very fast for continuous analytic functions, whether linear or nonlinear. They are tolerant of *some* computational noise in the functions, but they can be fooled into premature convergence if the function evaluations are made using preliminary (unconverged) SINDA/FLUINT solutions. Computational noise in F(x) misleads the finders by indicating false trends. Adjusting the parameter x every steady-state solution would exacerbate these false trends. Therefore, it is best to allow a few SINDA/FLUINT iterations to complete *between* adjustments to the parameter x, allowing the thermohydraulic network react to the previous change. Ideally, one should wait long enough for SINDA/FLUINT to fully converge without exiting the current STEADY solution. Therefore, the root and minimum finder utilities prevent SINDA/FLUINT convergence if they are still manipulating the parameter x (i.e., if they themselves have not also converged on a solution).

If the SINDA/FLUINT solution were perfectly linear (meaning no radiation, no fluid flow, constant thermophysical properties, and trivial co-solved user logic), then only one or two iterations would be needed to solve the network. In more realistic cases, perhaps 10-100 iterations are required. It is important to note that the solution is often approached asymptotically: a nearly-complete answer is achieved in 3-30 iterations, and a consistent trend toward the final solution is evident in only 2-10 iterations. So the “trick” in adding another co-solved solution (the details to which the SINDA/FLUINT solution is largely oblivious) is to make adjustments every 2-10 iterations: frequently enough for computational speed, but not so frequently that the interim solutions are so untrustworthy that they cause the root finder or minimum finder to return an inaccurate or erroneous solution. A slightly larger lag of between 5 and 20 iterations is typically required the first time (i.e., before any changes to x are made), since the SINDA/FLUINT solution is usually started from rough or guessed initial conditions.

In simple terms, *the speed of convergence of the SINDA/FLUINT and auxiliary solutions should be matched such that they approach the final solution together.* The settings for achieving this match are very problem dependent: some experimentation will be required.

For example, consider replacing a DPRVLV pressure regulator device with a simple LOSS device that achieves the same result. The DPRVLV would have set the path resistance as needed to achieve the PSET (downstream set-point pressure) specified by the user, so a LOSS device that achieves the same objective requires the user to adjust its K-factor (FK) iteratively. This K-factor becomes the x for the root or minimum finders: x=FK. Assume that the register *setP* is the desired pressure at the downstream lump, which is lump number 100. The regulation of the LOSS’s K-factor can be achieved by finding the root of the equation F(x) = F(FK) = setP-PL100, or by finding the minimum of either the function $(\text{setP-PL100})^2$ or |setP-PL100|. If FK is reset by the root or minimum finder, then the fluid network will start to adjust to the new value immediately, but it will approach a new solution in 3-4 iterations and will converge on the new solution in perhaps 5-10 iterations. If the root or minimum finder makes FK adjustments every SINDA/FLUINT iteration, it will be doing so based on still-in-progress solutions and will therefore make erroneous decisions based on excessive computational noise. If the finder waits more than 5-10 iterations before making the next adjustment, the SINDA/FLUINT solution may have achieved convergence (not expecting any further FK adjustments to be made). For such a system, an initial lag of 10 is about right, meaning 10 SINDA/FLUINT relaxation iterations (LOOPCT) will be calculated based on the initial K-factor before any change is made, such that the “PL100” value is roughly correct. Thereafter, the K-factor

can be adjusted every 5th iteration: frequently enough to minimize total iterations, but infrequently enough for the last K-factor to propagate through the solution such that the changes to PL100 are *approximately* correct. (As will be shown later, these settings correspond to LAGI=10 and LAGN=5 in the minimum and root finders.)

Just as SINDA/FLUINT applies some criteria for determining convergence based on how fast the solution is changing (specifically: ARLXCA, DRLXCA, RERRF), the root and minimum finders have their own tolerance settings for changes in both x and F(x). Unlike SINDA/FLUINT tolerances, these root and minimum finder tolerances (TOLX and TOLF, as will be shown later) can be set to zero.

A summary of the routines is provided below. Each such routine has its own initialization utility.

MIN_FIND . . . Find x such that F(x) is minimized over an interval x_{low} to x_{high}. This routine is placed in FLOGIC 0 or VARIABLES 1. It is initialized (or reset) using the routine MIN_FIND_I, which is usually called from OPERATIONS.

ROOT_FIND . Find x such that F(x) =0. Though it is not required, it is common to restrict the range of x over an interval x_{low} to x_{high} since most models cannot tolerate any value of x (e.g., zero or negative x). This routine is placed in FLOGIC 0 or VARIABLES 1. It is initialized (or reset) using the routine ROOT_FIND_I, which is usually called from OPERATIONS.

7.10.1 Co-Solved Minimum Finders

Subroutine Names: MIN_FIND_I, MIN_FIND

Description: MIN_FIND locates the minimum of a user-evaluated function: it finds x for which F(x) is minimized.

MIN_FIND is normally called in FLOGIC 0 or VARIABLES 1 during a STEADY solution (in which case NSOL=0). MIN_FIND_I initializes MIN_FIND if it has never been called within the current run, and resets it if it has been called previously, so MIN_FIND_I is normally called from within OPERATIONS.

MIN_FIND is intended to be used to co-solve for the minimum of function F(x) along with the SINDA/FLUINT STEADY or, less frequently, the STDSTL solution. The user starts by setting limits of x to search from XLO to XHI in OPERATIONS, along with iteration delay or “lag” factors (LAGI and LAGN) and x and F(x) tolerances (TOLX and TOLF), using the initialization routine MIN_FIND_I.

MIN_FIND_I also returns the first value of x (XOUT) that should be used to evaluate F(x) before the first call to MIN_FIND, which is located in either FLOGIC 0 or VARIABLES 1. The function F(x) should therefore be evaluated by the user before calling MIN_FIND. MIN_FIND will then return the next value of x (XOUT) at which F(x) should be evaluated:

```
HEADER OPERATIONS
      ...
      CALL MIN_FIND_I( Xlo, Xhi, Xout, ... )
      CALL STEADY
      ...
HEADER FLOGIC 0, ...          $ OR VARIABLES 1
      <update to F(Xin), perhaps as part of prior SINDA/FLUINT solution>
      CALL MIN_FIND( Xin, FatX, Xout )
```

MIN_FIND does not perturb x every iteration. Rather, it changes x at intervals prescribed by LAGI and LAGN to allow the SINDA/FLUINT solution a chance to take account of the last adjustment. While it is still adjusting x, MIN_FIND disables convergence of SINDA/FLUINT's steady-state solutions (usually STEADY). However, if STEADY does signal convergence while x is still being adjusted, MIN_FIND will skip the remaining delay and proceed with the next change in x. ***Therefore, large values of LAGI and LAGN are safe but potentially expensive.***

When the output value (XOUT) of MIN_FIND equals the input value (XIN) within the tolerance TOLX, the search has converged:*

$$|XOUT - XIN| < TOLX$$

Ideally, this convergence happens just before the SINDA/FLUINT solution converges. Refer to the discussion in the introduction section for more details on synchronizing solutions using the lag factors LAGI and LAGN.

To find the maximum of a function, minimize its negative: -F(x).

MIN_FIND can only search one function at a time. If multiple functions are required, and if the interactions between them are small, then consider using separate use of the independent routine DMIN_FIND (the double precision version of MIN_FIND).

Calling Sequence:

```
CALL MIN_FIND_I(xlo, xhi, xout, lagi, lagn, tolx, tolf)
```

* Also possible but less common, convergence will be assumed if |F(XOUT) - F(XIN)| < TOLF. Note that both TOLX and TOLF are absolute (unit dependent) and not relative or fractional.

where:

`xlo` ..... input value of the lower limit on x (real). This limit cannot later be adjusted during the search. The minimum might be found at this limit, but values lower than this will not be searched.

`xhi` ..... input value of the upper limit on x (real). This limit cannot later be adjusted during the search. The minimum might be found at this limit, but values greater than this will not be searched.

`xout` ..... output (returned) value of the first x that should be evaluated before calling MIN_FIND, which is normally called from FLOGIC 0 or VARIABLES 1.

`lagi` ..... input value of the initial delay in adjusting x (integer). If LAGI is input as nonpositive, then a default value of 5 is used (which should be considered a minimum). MIN_FIND will not adjust x until up to LAGI iterations have been completed. Specifically, the first adjustment of x usually will not be made until LOOPCT=LAGI+1, providing a chance for the SINDA/FLUINT solution to approach a reasonable value of $F(x_1)$ where x_1 is the XOUT value returned by MIN_FIND_I.

`lagn` ..... input value of the continuing delay in adjusting x after the initial delay of LAGI (integer). If LAGN is input as nonpositive, then a default value of 2 is used (which should be considered a minimum). MIN_FIND will adjust x up to every LAGN iterations. For example, if LAGI = 5 and LAGN = 3 then adjustments will be made at LOOPCT = 6, 9, 12, 15, ... etc. as shown below (Table 7-5) presuming SINDA/FLUINT does not converge on an interim value of x.

`tolx` ..... input (real) tolerance on change in x for achieving convergence. Zero values (meaning “as accurate as possible”) is legal, otherwise positive values will be taken as absolute (not relative or fractional) change in x. Negative values signal the default of 0.01 to be used, but again note that TOLX=0.01 does not mean a 1% change, but rather an absolute change in x of 0.01 units. If x is typically a large number (say an RPM of 10000.0), then a large TOLX should be supplied (say 10.0 or 100.0 for this RPM example).

`tolf` ..... input (real) tolerance on change in F(x) for achieving convergence. Zero values (meaning “as much accuracy as possible”) is legal, otherwise positive values will be taken as absolute (not relative or fractional) change in F(x). Negative values signal the default of 0.01 to be used, but again note that TOLF=0.01 does not mean a 1% change, but rather an absolute change in F(x) of 0.01 units. If F(x) is typically a large number, as large value of TOLF should be supplied. Convergence of MIN_FIND will occur if *either* TOLX *or* TOLF is achieved.

Calling Sequence:

```
CALL MIN_FIND(xin, fatx, xout)
```

where:

`xin`.....input (real) value of x at which function F has been evaluated (e.g., the value corresponding to FATX meaning "F(x)"). XIN should be equal to either the initial value of XOUT returned from MIN_FIND_I, or the XOUT value returned from a prior call to MIN_FIND (in a previous iteration). User adjustments to XIN should not be made.

`fatx`.....input (real) value of the function F evaluated at XIN: F(XIN).

`xout`.....output (real) value of x to use for the next evaluation of F(x). When MIN_FIND is converged, XOUT will equal XIN within the tolerance TOLX (presuming TOLF was not first satisfied). XIN and XOUT can be the same variable (e.g., register) if the user does not wish to check for convergence or monitor changes.

Table 7-5 Illustration of MIN_FIND_I and MIN_FIND Usage with LAGI=5 and LAGN=3

LOOPCT	XIN to MIN_FIND	FATX to MIN_FIND	XOUT from MIN_FIND	Comment
0	N/A	N/A	x1	XOUT from MIN_FIND_I in OPERATIONS
1	x1	N/A	x1	First call: SINDA/FLUINT not yet solving ...
2	x1	ignored	x1	First time F(x1) was evaluated before MIN_FIND
3	x1	ignored	x1	
4	x1	ignored	x1	
5	x1	ignored	x1	
6	x1	F(x1)	x2	LAGI F(x1) solutions completed: go to next x value
7	x2	ignored	x2	
8	x2	ignored	x2	
9	x2	F(x2)	x3	LAGN F(x2) solutions completed: change x again
10	x3	ignored	x3	
11	x3	ignored	x3	
12	x3	F(x3)	x4	LAGN F(x3) solutions completed: change x again
13	x4	ignored	x4	
14	x4	ignored	x4	
15	x4	F(x4)	x5	and so on until convergence (XIN=XOUT)

Guidance: If an inaccurate value of F(x) is found, then consider using larger values of LAGI and LAGN and trying again. Print out or plot the values of x and F(x) being passed to MIN_FIND. If the SINDA/FLUINT solution is still adjusting F(x) significantly when x is next adjusted, then MIN_FIND will be making decisions based on an inaccurate value of F(x), and the delays should be increased. On the other hand, if run times are excessive and if messages such as "CONVERGENCE DISALLOWED BECAUSE MIN_FIND HAS NOT YET CONVERGED" are frequent, then the delays can be decreased.

Guidance: If x varies by many orders of magnitude (i.e., if XLO << XHI), and if XLO and XHI are positive, then consider substituting x' = log(x) and evaluating F(exp(x')) to diminish the apparent range of variation. See example below.

Guidance: MIN_FIND is intended for co-solutions involving the STEADY (or perhaps STDSTL) routines since it accesses the control constant LOOPCT. If called during a transient, no lags will be used: x will be adjusted every time step. To call MIN_FIND in OPERATIONS (outside of a SINDA/FLUINT solution), set NSOL=2 and use LOOPCT as an iteration variable (perhaps in a DO WHILE loop). Note that NSOL=0 for STEADY, and NSOL=1 for STDSTL, and NSOL=2 for TRANSIENT (FWDBCK).

Examples:

```
c Find the maximum pump efficiency. Pump is path 404 in submodel POPT.
c The shaft speed of the pump, SPD404, is adjusted as the
c independent variable X, and the resulting pump efficiency EFFP is
c calculated by SINDA/FLUINT from the previous iteration.
c SINDA/FLUINT solves up to LAGI=10 times using the initial value of SPD,
c then SPD is adjusted up to every LAGN=5 iterations thereafter.
c The pump RPM is limited to be from 1000 to 5000
c The tolerance on the speed is set to 10 rpm (TOLX=10.0),
c and the tolerance on the efficiency is left as the default 0.01
c
HEADER OPERATIONS
...
      CALL MIN_FIND_I (1000., 5000., POPT.SPD404, 10, 5, 10.0, -1.0)
      CALL STEADY
...
HEADER FLOGIC 0, POPT
...
      CALL MIN_FIND( SPD404, -EFFP404, SPD404)
```

```
c Example 2 is a similar example to the above, using a parameter whose
c large variation requires a different treatment:
c Find the maximum pump efficiency. Pump is path 404 in submodel POPT.
c Resistance of a by-pass valve, LOSS path 505, is adjusted as the
c independent variable X, and the resulting pump efficiency EFFP is
c calculated by SINDA/FLUINT from the previous iteration.
c SINDA/FLUINT solves up to LAGI=15 times using the initial value of FK, then
c FK is adjusted up to every LAGN=10 iterations thereafter.
c FK is limited from 0.01 to 1.0e6.
c Because of this huge range, the logarithm of the K-factor
c is used as X rather than the actual K-factor itself.
c The tolerance on the log of the K-factor is set to zero (TOLX=0.0),
c and the tolerance on the efficiency is left as the default 0.01
c In case other solution routines are also called, the adjustments
c are limited to STEADY (FASTIC), where NSOL = 0
c
HEADER OPERATIONS
...
      CALL MIN_FIND_I (ALOG(0.01), ALOG(1.0E6), FTEST, 15, 10, 0.0, -1.0)
      POPT.FK505 = EXP(FTEST)
      CALL STEADY
...
HEADER FLOGIC 0, POPT
...
      if(nsol .eq. 0) then      $ only during STEADY (FASTIC)!
       FTEST= ALOG(FK505)
       CALL MIN_FIND( FTEST, -EFFP404, GTEST)
       IF(GTEST .EQ. FTEST) THEN
            WRITE (NUSER1,*) 'MIN_FIND CONVERGED AT LOOPCT = ', LOOPCT
       ENDIF
       FK505 = EXP(GTEST)
      endif
```

Subroutine Names: DMIN_FIND_I, DMIN_FIND

Description: These are double-precision variations of MIN_FIND_I and MIN_FIND. They are obsolete: MIN_FIND_I and MIN_FIND should be used instead.

7.10.2 Co-Solved Root Finders

Subroutine Names: ROOT_FIND_I, ROOT_FIND

Description: ROOT_FIND locates a root (zero crossing point) of a user-evaluated function: it finds x for which F(x)=0.

ROOT_FIND is normally called in FLOGIC 0 or VARIABLES 1 during a STEADY solution (in which case NSOL=0). ROOT_FIND_I initializes ROOT_FIND if it has never been called, and resets it if it has been called previously, so ROOT_FIND_I is normally called from within OPERATIONS.

ROOT_FIND is intended to be used to co-solve for the nearest root of function F(x) along with the SINDA/FLUINT STEADY or, less frequently, the STDSTL solution. The user optionally sets limits of x to search from XLO to XHI in OPERATIONS using the initialization routine ROOT_FIND_I (otherwise the finder might search the whole range of x). At the same time, the user can set iteration delay or "lag" factors (LAGI and LAGN) and x and F(x) tolerances (TOLX and TOLF).

ROOT_FIND_I also returns the first value of x (XOUT) that should be used to evaluate F(x) before the first call to ROOT_FIND, which is located in either FLOGIC 0 or VARIABLES 1. The function F(x) should therefore be evaluated by the user before calling ROOT_FIND. ROOT_FIND will then return the next value of x (XOUT) at which F(x) should be evaluated:

```
HEADER OPERATIONS
      ...
      CALL ROOT_FIND_I( Xlo, Xhi, Xout, ... )
      CALL STEADY
      ...
HEADER FLOGIC 0, ...          $ OR VARIABLES 1
      <update to F(Xin), perhaps as part of prior SINDA/FLUINT solution>
      CALL ROOT_FIND( Xin, FatX, Xout )
```

ROOT_FIND does not perturb x every iteration. Rather, it changes x at intervals prescribed by LAGI and LAGN to allow the SINDA/FLUINT solution a chance to take account of the last adjustment. While it is still adjusting x, ROOT_FIND disables convergence of SINDA/FLUINT's steady-state solutions (usually STEADY). However, if STEADY does signal convergence while x is still being adjusted, ROOT_FIND will skip the remaining delay and proceed with the next change in x. ***Therefore, large values of LAGI and LAGN are safe but potentially expensive.***

When the output value (XOUT) of ROOT_FIND equals the input value (XIN) within the tolerance TOLX, the search has converged:*

$$|XOUT - XIN| < TOLX$$

Ideally, this convergence happens just before the SINDA/FLUINT solution converges. Refer to the discussion in the introduction section for more details on synchronizing solutions using the lag factors LAGI and LAGN.

ROOT_FIND works in one of two modes: either searching a bounded region of x, or an unbounded one. For well-behaved functions, setting a search range (from XLO to XHI) in ROOT_FIND_I can cost a few extra steady state iterations† that might not otherwise be necessary. Usually, however, it is well worth this extra computational expense since most models cannot tolerate any value of x to be tested. For example, x might be a shaft speed and testing zero, negative, or huge speeds might cause solution problems including aborts. As another example, x might be a valve position, and without limits a negative or too-large position might be returned by ROOT_FIND, causing problems in the next SINDA/FLUINT iteration. If limits *are* set, then ROOT_FIND will not test values outside of XLO to XHI. However, the user must assure that F(XLO)*F(XHI) < 0 ... that is, *that function F has opposite signs at the extremes of the valid range of x, such that a root exists between the two limits.* Otherwise, if both F(XLO) and F(XHI) are either positive or if both are negative, then ROOT_FIND will resort to searching an unlimited range of x values and modeling problems might ensue. To assure appropriate signs at the limits of x, the user should make sure that F is calculated correctly, and that LAGI is large enough such that the returned values of F(XLO) and F(XHI) are accurate (or at least of the correct sign).

ROOT_FIND can only search one function at a time. If multiple functions are required, and if the interactions between them are small, then consider using separate use of the independent routine DROOT_FIND (the double precision version of ROOT_FIND).

Calling Sequence:

```
CALL ROOT_FIND_I(xlo, xhi, xout, lagi, lagn, tolx, tolf)
```

* Also possible but less common, convergence will be assumed if |F(XOUT) - F(XIN)| < TOLF. Note that both TOLX and TOLF are absolute (unit dependent) and not relative or fractional.

† Specifically, up to an extra LAGI solutions may be required.

where:

`xlo` ..... input value of the lower limit on x (real). This limit cannot later be adjusted during the search. The root might be found at this limit, but no smaller value than this will be tested (providing F(XLO) is opposite in sign from F(XHI)).

`xhi` ..... input value of the upper limit on x (real). This limit cannot later be adjusted during the search. The root might be found at this limit, but no larger value than this will be tested (providing F(XLO) is opposite in sign from F(XHI)). If no such range is applicable (i.e., if any value of x might be valid), then set XHI equal to XLO.

`xout` ..... output (returned) value of the first x that should be evaluated before calling ROOT_FIND, which is normally called from FLOGIC 0 or VARIABLES 1. This will be equal to XLO in all cases, and is present as an argument only for consistency with MIN_FIND.

`lagi` ..... input value of the initial delay in adjusting x (integer). If LAGI is input as nonpositive, then a default value of 5 is used (which should be considered a minimum). ROOT_FIND will not adjust x until up to LAGI iterations have been completed. Specifically, the first adjustment of x is usually not be made until LOOPCT=LAGI+1, providing a chance for the SINDA/FLUINT solution to approach a reasonable value of $F(x_1)$ where x_1 is the XOUT value returned by ROOT_FIND_I (the value of F(XLO) is being sought if XLO ≠ XHI). If XHI ≠ XLO, than at this point (LOOPCT=LAGI+1), ROOT_FIND will return XHI for the next LAGI iterations so that F(XHI) can be evaluated.

`lagn` ..... input value of the continuing delay in adjusting x after the initial delay of LAGI or 2*LAGI (integer). If LAGN is input as nonpositive, then a default value of 2 is used (which should be considered a minimum). ROOT_FIND will adjust x up to every LAGN iterations. For example, if LAGI = 4 and LAGN = 3 (and XLO ≠ XHI) then adjustments will be made at LOOPCT = 9, 12, 15, 18, ... etc. (See Table 7-6) presuming SINDA/FLUINT does not converge on an interim value of x

`tolx` ..... input (real) tolerance on change in x for achieving convergence. Zero values (meaning “as accurate as possible”) is legal, otherwise positive values will be taken as absolute (not relative or fractional) change in x. Negative values signal the default of 0.01 to be used, but again note that TOLX=0.01 does not mean a 1% change, but rather an absolute change in x of 0.01 units. If x is typically a large number (say an RPM of 10000.0), then a large TOLX should be supplied (say 10.0 or 100.0 for this RPM example).

`tolf` ..... input (real) tolerance on change in F(x) for achieving convergence. Zero values (meaning “as much accuracy as possible”) is legal, otherwise positive values will be taken as absolute (not relative or fractional) change in F(x). Negative values signal the default of 0.01 to be used, but again note that TOLF=0.01 does not mean a 1% change, but rather an absolute change in F(x) of 0.01 units. If F(x) is typically a large number, as large value of TOLF should be supplied. Convergence of ROOT_FIND will occur if *either* TOLX *or* TOLF is achieved.

Calling Sequence:

```
CALL ROOT_FIND(xin, fatx, xout)
```

where:

`xin`.....input (real) value of x at which function F has been evaluated (e.g., the value corresponding to FATX meaning "F(x)"). XIN should be equal to either the initial value of XOUT returned from ROOT_FIND_I, or the XOUT value returned from a prior call to ROOT_FIND (in a previous iteration). User adjustments to XIN should not be made.

`fatx`.....input (real) value of the function F evaluated at XIN: F(XIN).

`xout`.....output (real) value of x to use for the next evaluation of F(x). When ROOT_FIND is converged, XOUT will equal XIN within the tolerance TOLX (presuming TOLF was not first satisfied). XIN and XOUT can be the same variable (e.g., register) if the user does not wish to check for convergence or monitor changes.

Table 7-6 Illustration of ROOT_FIND_I, ROOT_FIND Usage with LAGI=4, LAGN=3, XLO ≠ XHI

LOOPCT	XIN to ROOT_FIND	FATX to ROOT_FIND	XOUT from ROOT_FIND	Comment
0	N/A	N/A	xlo	XOUT from ROOT_FIND_I in OPERATIONS
1	xlo	N/A	xlo	First call: SINDA/FLUINT not yet solving ...
2	xlo	ignored	xlo	First time F(xlo) was evaluated before ROOT_FIND
3	xlo	ignored	xlo	
4	xlo	ignored	xlo	
5	xlo	F(xlo)	xhi	LAGI F(XLO) solutions completed: go to XHI
6	xhi	ignored	xhi	
7	xhi	ignored	xhi	
8	xhi	ignored	xhi	
9	xhi	F(xhi)	x1	LAGI F(XHI) solutions completed: go to next x
10	x1	ignored	x1	
11	x1	ignored	x1	
12	x1	F(x1)	x2	LAGN F(x1) solutions completed: go to next x
13	x2	ignored	x2	
14	x2	ignored	x2	
15	x2	F(x2)	x3	LAGN F(x2) solutions completed: go to next x
16	x3	ignored	x3	
17	x3	ignored	x3	
18	x3	F(x3)	x4	LAGN F(x3) solutions completed: go to next x
19	x4	ignored	x4	and so on until convergence (XIN=XOUT)

Guidance: If an inaccurate value of F(x) is found, then consider using larger values of LAGI and LAGN and trying again. Print out or plot the values of x and F(x) being passed to ROOT_FIND. If the SINDA/FLUINT solution is still adjusting F(x) significantly when x is next adjusted, then ROOT_FIND will be making decisions based on an inaccurate

value of F(x), and the delays should be increased. On the other hand, if run times are excessive and if messages such as “CONVERGENCE DISALLOWED BECAUSE ROOT_FIND HAS NOT YET CONVERGED” are frequent, then the delays can be decreased.

Guidance: If x varies by many orders of magnitude (i.e., if XLO << XHI), and if XLO and XHI are positive, then consider substituting x’ = log(x) and evaluating F(exp(x’)) to diminish the apparent range of variation. See example below.

Guidance: ROOT_FIND is intended for co-solutions involving the STEADY (or perhaps STDSTL) routines since it accesses the control constant LOOPCT. If called during a transient, no lags will be used: x will be adjusted every time step. To call ROOT_FIND in OPERATIONS (outside of a SINDA/FLUINT solution), set NSOL=2 and use LOOPCT as an iteration variable (perhaps in a DO WHILE loop). Note that NSOL=0 for STEADY, and NSOL=1 for STDSTL, and NSOL=2 for TRANSIENT (FWDBCK).

Caution: If XLO and XHI were not equal in the initial ROOT_FIND_I call, but if the signs of F are the same at those two points, then ROOT_FIND will ignore limits and will instead explore the full range of x in search of a root. If this is not desired, consider larger LAGI values such that the values calculated at F(XLO) and F(XHI) are more accurate (and hence of opposite sign).

Examples:

```
c Find the shaft speed that balances torque in a turbocharger, with
c the compressor being path 301 and the turbine path 401. The
c independent variable X is the shaft speed for both machines, set
c via register shaftRPM, and the torque values are
c calculated by SINDA/FLUINT from the previous iteration. SINDA/FLUINT
c solves up to LAGI=10 times using the initial value of shaftRPM,
c then it is adjusted up to every LAGN=6 iterations thereafter.
c The speed is limited to be from 3000 to 10,000 rpm.
c The tolerance on the shaftRPM is set to 10 rpm (TOLX=10.0),
c and the tolerance on the torque is left as the default 0.01 (ft-lbf or N-m)
c
HEADER OPERATIONS
...
      CALL ROOT_FIND_I (3000., 10000., shaftRPM, 10, 6, 10.0, -1.0)
      CALL STEADY
...
HEADER FLOGIC 0, TBALANCE
...
      NetTorq = TORQ301+TORQ401
      CALL ROOT_FIND( shaftRPM, NetTorq, shaftRPM)
```

```
c Example 2 a similar example to the above, using a parameter whose
c large variation requires a different treatment.
c Find the turbine by-pass flow that balances torque in a turbocharger, with
c the compressor being path 301 and the turbine path 401. The
c independent variable X is the K-factor (FK) for the by-pass LOSS #501.
c The torque values are calculated by SINDA/FLUINT from the
c previous iteration.
c SINDA/FLUINT solves up to LAGI=15 times using the initial value of FK501,
c then it is adjusted up to every LAGN=8 iterations thereafter.
c The K-factor is limited to be from 0.1 to 1000.
c Because of this huge range, the logarithm of the K-factor is used
c as X rather than the actual K-factor itself.
c The tolerance on the log of the K-factor is set to zero (TOLX=0.0),
c and the tolerance on the torque is left as the default 0.01 (ft-lbf or N-m)
c In case other solution routines are also called, the adjustments
c are limited to STEADY (FASTIC), where NSOL = 0
c
HEADER OPERATIONS
...
      CALL ROOT_FIND_I (ALOG(0.1), ALOG(1000.), FTEST, 15, 8, 0.0, -1.0)
      TBALANCE.FK501 = EXP(FTEST)
      CALL STEADY
...
HEADER FLOGIC 0, TBALANCE
...
      if(nsol .eq. 0) then      $ only during STEADY (FASTIC)!
       FTEST= ALOG(FK501)
       NetTorq = TORQ301+TORQ401
       CALL ROOT_FIND( FTEST, NetTorq, GTEST)
       IF(GTEST .EQ. FTEST) THEN
            WRITE (NUSER1,*) 'ROOT_FIND CONVERGED AT LOOPCT = ', LOOPCT
       ENDIF
       FK501 = EXP(GTEST)
      endif
```

Subroutine Names: DROOT_FIND_I, DROOT_FIND

Description: These are double-precision variations of ROOT_FIND_I and ROOT_FIND. They are obsolete: ROOT_FIND_I and ROOT_FIND should be used instead.

7.11 FLUINT Subroutines

This section describes the utility, property, correlation, simulation, and output routines that are available to the FLUINT user.

7.11.1 FLUINT Utility Subroutines

This section describes utility routines for changing certain special parameters, performing dynamic translations, modifying the operation of the network, and fetching additional information.

7.11.1.1 Change Utilities

Certain FLUINT parameters, such as volumes and thermodynamic states, must be changed using a utility routine:

CHGLMP . . . . . .Change lump thermodynamic state
CHGLMP_MIX . . .Simultaneously change multiple species' mass fractions in a lump (see Section 7.11.1.2)
CHALLP. . . . . .Change all lump pressures in a submodel
CHGVOL . . . . . .Change tank volume
TANKTRHO . . . . .Change tank temperature, holding density/mass constant
CHGLSTAT . . . . . .Change LSTAT for a lump
CHGSUC . . . . . .Change path suction status
CHG_HTPPORT . .Change the port path (*iport, fport*) for an HTP tie
CHGFLD . . . . . .Change submodel working fluid

Subroutine Name: CHGLMP

This routine can be used to make instantaneous changes in lump thermodynamic states. A state is described by lump variables TL, PL, XL, AL, XGx, and XFx (where "x" is the letter identifier of the constituent). With this routine, any of the first four quantities may be changed while either TL, PL, or XL is held constant. For example, PL can be changed while holding either TL or XL constant. Only the values being changed or held can be guaranteed, the third value may or may not change. For example, if the current state is subcooled and PL is changed with XL held constant at 0.0, then TL will be the smaller of the current value and the saturation pressure corresponding to the new value of PL.

Changes to XGx (gas constituent mass fraction) cannot be made unless the lump contains some gas (perhaps by changing XL in a previous CHGLMP call), and changes to XFx cannot be made unless liquid exists in the lump. If one of the constituents in a mixture is condensible/volatile, it may take extra care in inputs, and perhaps two or three consecutive calls to CHGLMP to arrive at state

where PL, TL, and XGx are as desired. In such cases, resulting states may vary slightly from the values of input or held parameters (see Guidance below). For simultaneous changes to several or all species' mass fractions, use CHGLMP_MIX (Section 7.11.1.2) instead of CHGLMP.

This routine is normally used to make step changes in pressure, temperature, or constituent concentration in plena, but may be used to change quality and may be used on any lump. Note that changes to a tank's state will affect its mass and energy, but not its volume (even if pressure changes and the tank is compliant).

Restrictions: Do not use within FLOGIC 1 blocks. The desired thermodynamic state must be within the range of fluid properties. PL and TL are assumed to be in the user's unit system as selected using ABSZRO and PATMOS. Input errors result in no changes made.

Calling Sequence:

```
CALL CHGLMP (MODNAM, LUMPNO, VARY, VALUE, HOLD)
```

where:

MODNAM . . . . . fluid submodel name containing lump to be changed, input in single quotes such as 'LOOP'
LUMPNO . . . . . identifier of lump to be changed, such as 100 or 352, etc.
VARY. . . . . property to vary: 'PL', 'TL', 'XL', 'AL', 'XGx', or 'XFx', where "x" is the letter designator of the constituent
VALUE . . . . . new value for property named in VARY
HOLD. . . . . property to hold constant: 'PL', 'TL', or 'XL'. Cannot be same as VARY

Examples:

```
C CHANGE THE TEMP OF LUMP 123 TO 300 DEGREES,
C HOLDING PRESSURE CONSTANT (QUALITY MIGHT THEREFORE CHANGE)
      CALL CHGLMP ('COOLER', 123, 'TL', 300.0, 'PL')
C CHANGE THE FRACTION OF GAS V IN LUMP 321 TO 10% (of gas mass)
      CALL CHGLMP ('MIXDUP', 321, 'XGV', 0.1, 'PL')
```

Guidance: The XG of a condensible species has additional constraints that must be met: its value cannot exceed saturation (100% relative humidity), and the value must correspond to exactly saturation if any volatile liquid of that species is also present in the same lump. Differences between specified and held values may result if XG for the condensible species is the property to vary (VARY). A precursor call to HUMR2X is recommended to avoid poorly chosen values of XG for the condensible species since the code will take extra efforts to try to achieve the input XG value, potentially hitting upper or lower temperature/pressure limits in the process. Consider also specifying the XG of noncondensible species instead of the condensible species, perhaps using CHGLMP_MIX (Section 7.11.1.2) instead of CHGLMP if many such species exist. Specifying other (noncondensible) species signals to the code that more freedom is available in adjusting the XG of the condensible species if necessary.

Subroutine Name: CHALLP

This routine can be used to make instantaneous changes in lump pressures throughout an entire fluid submodel. It is useful for modeling constant-volume systems such as vapor compression cycles, heat pipes, loop heat pipes, and fixed conductance capillary pumped loops.

This routine internally calls CHGLMP (described above) holding quality constant. Refer to the CHGLMP description for more information and restrictions.

Restrictions: Do not use within FLOGIC 1 blocks. The desired thermodynamic state must be within the range of fluid properties. The delta pressure is assumed to be in the user's unit system as selected using UID, ABSZRO and PATMOS. Input errors result in no changes made.

Calling Sequence:

```
CALL CHALLP (MODNAM, DELP)
```

where:

MODNAM .... fluid submodel name to be changed, input in single quotes such as 'LOOP'
DELP ..... delta pressure to impose on all lumps in MODNAM

Example:

```
C RAISE THE PRESSURE IN THE LOOP 2 PSID
      CALL CHALLP ('THELOOP', 2.0)
```

Subroutine Name: CHGVOL

Description: This routine can be used to make changes in tank volumes. Intensive thermodynamic properties (temperature, pressure, etc.) are conserved, so overall mass and energy will be increased or decreased according to the ratio of the volumes. This routine can be used before or after CHGLMP to adjust tank states as desired.

Restrictions: Do not use within FLOGIC 1 blocks. The new volume must be a positive number. Input errors result in no changes made.

Calling Sequence:

```
CALL CHGVOL (MODNAM, NTANK, VNEW)
```

where:

MODNAM . . . . . fluid submodel name containing tank to be changed, input in single quotes such as 'LOOP'

NTANK . . . . . identifier of tank to be changed, such as 100 or 352, etc.

VNEW. . . . . new volume for tank (ft^2 or m^2)

Example:

```
CALL CHGVOL ('FLU', 10300, 1.0E-4)
```

Subroutine Name: TANKTRHO

Description: This routine allows the user to change the temperature of a tank (and only a tank) instantly, holding density constant. It may be used to specify density (and therefore mass) perhaps as an initial condition. TANKTRHO can be used to avoid the need for a steady-state solution that would otherwise be required to bring a tank of known mass to a final temperature, with the final pressure unknown.

TANKTRHO should *not* be called from within FLOGIC 1, and will normally be called in either OPERATIONS or PROCEDURE.

If the tank is compliant (COMP>0), then TANKTRHO will attempt to maintain constant *mass* rather than constant density: the volume may be adjusted. If this treatment is not appropriate, then temporarily set COMP to zero to disable this behavior before a TANKTRHO call, then reset the COMP to the original value following the TANKTRHO call.

TANKTRHO is usually not applicable when starting with an all liquid (XL=0) state, or at least it should be used with caution when traversing the boundary between XL=0 and XL>0. Consider perhaps starting from a very small quality (say, XL=1.e-6) instead, and be careful not to set a new density higher than that which corresponds to all liquid.

Calling Sequence:

```
CALL TANKTRHO(smn, IDtank, Tnew, Dnew)
```

where:

`smn` ..... fluid submodel name, in single quotes
`IDtank` .... user ID for the tank to be changed
`Tnew` ..... new temperature (in user units) to set. If Tnew is less than or equal to *absolute zero* (e.g., -1.0e30), then the current value will be preserved: no change will be made to the tank's temperature. Note that setting Tnew to zero will cause the new temperature to be zero when using degrees F or C.
`Dnew` ..... new density (in user units) to set. If Dnew is zero or negative, the current density will be preserved.

Example (bring tank glide.331 to 200 degrees, holding current mass constant):

```
CALL TANKTRHO('glide', 331, 200.0, 0.0)
```

Caution: TANKTRHO uses an internal iteration that may occasionally stall short of the final temperature or density. If this occurs, consider repeated (redundant) calls to continue moving the tank state toward the desired value as a work-around measure.

Subroutine Name: CHGLSTAT

Description: This routine can be used to make changes in the lump stagnation status. If LSTAT=NORM, then the lump pressure (PL) is treated as static. If LSTAT=STAG, then not only is the pressure treated as stagnation (total), but acceleration losses are applied to appropriate paths flowing out of this lump, kinetic energy is neglected at the upstream end of those paths (since the upstream velocity is zero), and any choking in those paths includes an expansion processes even if AFTH is equal to AF.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no changes made.

Calling Sequence:

```
CALL CHGLSTAT (MODNAM, NLUMP, LSTAT)
```

where:

`MODNAM` .... fluid submodel name containing lump to be changed, input in single quotes such as 'LOOP'
`NLUMP` ..... identifier of lump to be changed, such as 100 or 352, etc.
`STAT` ..... new value of status flag:
'NORM' Normal, static pressure, no entrance accelerations applied
'STAG' Stagnation pressure, entrance accelerations applied

Example:

```
CALL CHGLSTAT ('LAUNCH', 321, 'NORM')
```

Subroutine Name: CHGSUC

Description: This routine can be used to make changes in the path phase suction status.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no changes made.

Calling Sequence:

```
CALL CHGSUC (MODNAM, NPATH, STAT)
```

where:

MODNAM . . . . . fluid submodel name containing path to be changed, input in single quotes such as 'LOOP'
NPATH . . . . . identifier of path to be changed, such as 100 or 352, etc.
STAT. . . . . new value of status flag:
'NORM' . .Normal, extract both phases
'LS' . . . . . Extract only liquid from defined upstream lump
'VS' . . . . . Extract only vapor (or gas) from defined upstream lump
'RLS' . . . . . Extract only liquid from defined downstream lump
'RVS' . . . . . Extract only vapor from defined downstream lump
'DLS' . . . . . Extract only liquid from either end
'DVS' . . . . Extract only vapor from either end

Example:

```
CALL CHGSUC ('COOLER', 123, 'NORM')
```

Subroutine Name: CHG_HTPPORT

Description: This routine allows users to dynamically change the port path (*iport*) on an HTP tie to a new path, or to change the port path multiplier (*fport*), or both.

Restrictions: Do not use in FLOGIC 1. No action will be taken if the specified tie is not an HTP tie, or if the submodel or path is not found, or if a secondary member of a twinned tie is specified.

Calling Sequence:

```
CALL CHG_HTPPORT(MODNAM, NTIE, NPORT, FPORT)
```

where:

MODNAM	fluid submodel name containing tie, input in single quotes
NTIE	identifier of HTP tie whose port is to be changed. Should not be a secondary tie: if twinned, NTIE should be the primary tie ID.
NPORT.....	identifier of new path within MODNAM to use as a port path. If zero, the existing path will be preserved as the port path.
FPORT.....	optional port path multiplying factor (real). If this argument is omitted, the current factor will be used.

Examples:

```
c Change HTP tie 334's port path to be 400, using the same FPORT factor:
      CALL CHG_HTPPORT('NEWPORT', 334, 400)
C This example changes both the port and the factor:
      CALL CHG_HTPPORT('BEACH', 45, 1000, -1.0)
C This example changes just the factor (easier to use registers):
      CALL CHG_HTPPORT('BEACH', 45, 0, -1.0)
```

Subroutine Name: CHGFLD

This routine can be used to change the working fluid within a fluid submodel. The new working fluid is designated by its ASHRAE identifier, or by the identifier provided by the user in an FPROP DATA block. Only those fluids in the prestored library or currently defined in an FPROP block can be used as working fluids. In other words, if a user-defined fluid is to be used to replace the working fluid in a submodel during processor execution, it must have been defined in the input stream within an FPROP DATA block. (Fluids may be defined without necessarily being used in a fluid submodel.)

Input errors result in no changes made.

Use only in OPERATIONS or PROCEDURE.

When a call to CHGFLD is made, all lumps in the submodel will be reinitialized. Most lump thermodynamic properties (PL, TL, DL, HL, etc.) will be reset during this process as needed to be consistent with the new working fluid. For example, if the previous fluid were a two-phase description and the new fluid were intrinsically single-phase (i.e., a 8000 or 9000 series fluid), then the qualities of some or all lumps may need to be changed. The reinitialization is the same as the initialization that is applied when the processor is first started: it may result in error messages being produced if current lump properties (TL, PL) are outside of the valid range of the new working fluid.

The user has the choice of either keeping temperatures or pressures constant during the reinitialization of two-phase states. Within FLOW DATA, the user may choose either to preserve temperatures (the default method) or pressures (using the "PL!" option) of each lump in case of conflicts. With CHGFLD, such a decision must be made for all lumps in the submodel. Therefore, subsequent calls to CHGLMP might be made as needed to customize any initial conditions.

Caution: Avoid the use of RESTAR or RESPAR after a call to CHGFLD unless the working fluid used at the time the corresponding RESAVE or SAVPAR calls were made matches the current fluid. In other words, mismatches between current and previous working fluids will cause property routine error messages and likely result in an aborted run.

Caution: Indirect translations of the fluid property identifier (e.g., “SUB.FI” versus “FI717” - see Section 7.11.2) within logic blocks are not affected by CHGFLD. As an example, assume that “SUB.FI” is used within a logic block to refer to the working fluid of submodel “SUB,” and that the original working fluid for “SUB” (as specified in its FLOW DATA header block) is 9718. The occurrence of “SUB.FI” in logic blocks therefore refers to fluid 9718. Such occurrences are translated once during preprocessing, and are not be affected by later operations performed in the processor. In other words, despite a call to CHGFLD in which the fluid is changed to 9304, “SUB.FI” will still refer to 9718. A dynamic translation routine (INTFLD, Section 7.11.1) can be used if needed to overcome this difficulty.

Restrictions: CHGFLD cannot be applied to multiple constituent submodels.

Calling Sequence:

```
CALL CHGFLD (MODNAM, NEWFLU, HOLD)
```

where:

MODNAM . . . . . fluid submodel name containing lump to be changed, input in single quotes such as ’LOOP’

NEWFLU . . . . . identifier of new working fluid (e.g., 717, 505, 7732, etc.)

HOLD. . . . . property to hold constant during reinitialization: ’PL’ or ’TL’. Using ‘PL’ is analogous to using the “PL!” option (in FLOW DATA) for all lumps in the submodel.

Example:

```
C CHANGE THE FLUID TO 8732 (PERFECT GAS)
C HOLDING TEMP. CONSTANT (PRESSURE MIGHT THEREFORE CHANGE)
      CALL CHGFLD (’HATRAK’, 8732, ’TL’)
```

7.11.1.2 Changing Lump Constituent Mass Fractions

The concentration of any *one* gas or liquid species can be changed via a call to CHGLMP (Section 7.11.1.1). If there are only two species present in a mixture, the concentration of the other (unspecified) species is predictable: it will be one minus the specified fraction. For example, if XFW=0.5 and XFT=0.5, then if XFW is changed to XFW=0.6, it follows that XFT=0.4.

If there are three or more species in the mixture, CHGLMP will adjust the concentrations of the unspecified species *in proportion to their current concentrations*. In other words, if XGA=0.2, XGB=0.2, and XGC=0.6, then changing XGA to 0.1 will result in XGB=0.225 and XGC=0.675

such that the sum is still unity. Since there was initially 3 times as much C as there was B, species C absorbed three times as much of the discrepancy, and there remains three times as much C as there is B.

If the user needed to specify more than once species at a time, or perhaps all of them, it should be apparent that repeated calls to CHGLMP will not achieve the desired result. CHGLMP_MIX provides this functionality. Because several species concentrations must be specified at the same time when using CHGLMP_MIX, its inputs are more complex than the inputs for CHGLMP.

CHGLMP_MIX also provides the option of working with mole fractions* instead of mass fractions as needed to better support chemical reactions. Used in this way, CHGLMP_MIX can be seen as a way to impose or update an equilibrium concentration: an instantaneous variation of EQRATE (Section 7.11.9.9).

Subroutine Name: CHGLMP_MIX

This routine can be used to make instantaneous changes to the concentrations of gases or liquids defining lump thermodynamic states. A state is described by lump variables TL, PL, XL, XGx, and XFx (where "x" is the letter identifier of the constituent). With this routine, one or more of the constituent mass (or mole) fractions in either the gas or liquid phase can be changed while either TL or PL is held constant. For example, the gas mass fractions for species D, R, and Y can be changed while holding TL constant. Or, the liquid mole fractions of species W, E, and T can be changed while holding PL constant.

Only the values being changed or held can be guaranteed: other values may or may not change. For example, if PL is held, it might be necessary to change TL.

Sometimes, even the held variable (PL or TL) might change, and the requested mass fractions might not even be possible. As an example of this situation, consider a current state that is two-phase with a volatile liquid present. If the concentration of the vapor is changed, it may be necessary to also change *both* PL and TL as needed to achieve a saturated (100% relative humidity) state. If either PL or TL hits a range limit, the specified vapor mass or mole fraction might change (perhaps even to unity).

Changes to XGx (gas constituent mass fraction) cannot be made unless the lump contains some gas (perhaps by changing XL in a previous CHGLMP call), and changes to XFx cannot be made unless liquid exists in the lump. If one of the constituents in a mixture is condensible/volatile, it may take extra care in inputs, and perhaps two or three consecutive calls to CHGLMP_MIX to arrive at state where PL, TL, and XGx are as desired. In such cases, resulting states may vary slightly from the values of input or held parameters (see Guidance below).

* Note that, for perfect gas mixtures, the mole fraction is the same as the partial pressure fraction (PPG). For liquid mixtures, the mole fraction is reported in the output variable MF.

This routine is normally used to make step changes in constituent concentrations in plena, but may be used on any lump. Note that changes to a tank's state will affect its mass and energy, but not its volume (even if pressure changes and the tank is compliant).

Similarities between CHGLMP_MIX and EQRATE (Section 7.11.9.9) are intentional. EQRATE can be used to drive one or more reactions in a tank toward equilibrium. CHGLMP_MIX can be used to instantaneously impose an equilibrium condition on any lump (usually a plenum acting as a boundary condition). However, note that each routine defines XS (the specified fractions) slightly differently. XS usually sums to unity in EQRATE for each reaction, but can sum to less than unity in CHGLMP_MIX. Also, the meaning of XS can be mass or mole fractions in CHGLMP_MIX, but is always mass fractions in EQRATE.

Restrictions: Do not use within FLOGIC 1 blocks. The desired thermodynamic state must be within the range of fluid properties. Input errors result in no changes made.

Calling Sequence:

```
      CALL CHGLMP_MIX (MODNAM, LUMPNO, VARY, XS(IC), NS,
     +          IDS(IC), AS, HOLD)
```

where:

MODNAM . . . . . fluid submodel name containing the lump to be changed, input in single quotes such as 'LOOP'

LUMPNO . . . . . identifier of lump to be changed, such as 100 or 352, etc.

VARY. . . . . character string (in single quotes) defining the parameter to be changed:
'XG' or 'XGMASS' ..... change the gas mass fractions
'XF' or 'XFMASS' ..... change the liquid mass fractions
'XGMOLE' ..... change the gas mole fractions
'XFMOLE' ..... change the liquid mole fractions

`XS(ic)` .... SINDA (integer count) style real array reference: the array containing desired species mass or mole fractions.
Normally, this is of the form "smn.An" but if NS=1 (only one species), then as a shortcut this argument can be a single floating point number or variable (between 0.0 and 1.0) instead of an array.

`NS`..... number of reacting species (the number of species listed in XS, XN, and IDS). May be a subset of all available species within this submodel. If NS=1, then shortened forms of the arguments XS(ic) and IDS(ic) are allowed.

`IDS(ic)` ... list of fluid IDs (in the form of a SINDA integer array) corresponding to the species in XS, or zero if AS is used instead.
Normally, this is of the form "smn.NAm" but if NS=1 (only one species), then as a shortcut this argument can be a single integer number or variable (between 11 and 9999) instead of an array.

`AS`..... alphabetic string (in single quotes, or character string) of letter IDs of species corresponding to the species in XS, or an empty string (' ') if IDS is used instead.

`HOLD` ..... property to hold constant: 'PL' or 'TL'.

Examples:

```
C Only one of ? species changed to 80%: species W which is fluid 8718.
C In the following equivalent examples, array 100 contains the value 0.8,
C and integer array 200 contains the value 8718 in the first cell. Also
C real register WFRACT contains the value 0.8, and integer register IWAT
C contains the value 8718.
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XG', A100, 1, NA200, '', 'PL')
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XG', 0.8, 1, 8718, '', 'PL')
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XG', 0.8, 1, 0, 'W', 'PL')
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XG', wfract, 1, iwat, '', 'PL')
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XGMASS', 0.8, 1, iwat, '', 'PL')
C Since only one species is being changed, the following is also equiv.:
      CALL CHGLMP ('GASEOUS', 123, 'XGW', 0.8, 'PL')
C In a variation of the above, 80% is the mole fraction, not the mass
C fraction (a capability unavailable in CHGLMP):
      CALL CHGLMP_MIX ('GASEOUS', 123, 'XGMOLE', 0.8, 1, 0, 'W', 'PL')
C
C For examples of changing multiple species simultaneously (NS > 1),
C see below.
```

Guidance: The XG of a condensible (vapor) species has additional constraints that must be met: its value cannot exceed saturation (100% relative humidity), and the value must correspond to exactly saturation if any volatile liquid of that species is also present in the same lump. Differences between specified and held values may result if XG for the condensible species is listed in XS. A precursor call to HUMR2X is recommended to avoid poorly chosen values of XG for the condensible species since the code will take extra efforts to try to achieve the input XG value, potentially hitting upper or lower temperature/pressure limits in the process. Consider also specifying the XG of non-

condensible species instead of the condensible species. Specifying other (noncondensible) species signals to the code that more freedom is available in adjusting the XG of the condensible species if necessary to preserve a constant PL or TL. See example below.

Equilibrium Concentration Example—Consider a gaseous mixture of O_2, H_2, and H_2O (as species 'O', 'H', and 'W,' respectively). If all of these fluids are reacting to equilibrium, and therefore their concentration needs to be changed simultaneously, then NS=3. In this case, if AS = 'OHW' and XS is specified in ARRAY data as real SINDA array #100:

```
100 = 0.02, 0.01, 0.97
```

this means that, at equilibrium and with no condensate forming, the mixture will contain XGO=0.02, XGH=0.01, and XGW=0.97 if XS represents mass fractions. If instead XS represents mole fractions, then the resulting partial pressure fractions will occur (*approximately* unless all species are perfect): PPGO=0.02, PPGH=0.01, and PGW=0.97.

Assuming mass fractions are specified by XS, the CHGLMP_MIX call for plenum 9 in submodel *stanjan* might appear as:

```
CALL CHGLMP_MIX('stanjan', 9, 'XG', A100, 3, 0, 'OHW', 'PL')
```

Otherwise, if XS contained mole fractions, the 'XG' argument would have been replaced by 'XGMOLE' in the above call.

A more complete listing of the input file for the above example is provided below, including the alternative use (commented out) of the IDS() array instead of the AS string:

```
HEADER FLOW DATA, STANJAN, FIDO = 8732, FIDH = 8704, FIDW = 6070
 ...
HEADER ARRAY DATA, STANJAN
C these are often calculated by an equilibrium solver like STANJAN
      100 = 0.02, 0.01, 0.97
c list of fluid identifiers for IDS() option:
      200 = 8732, 8704, 6070
 ...
HEADER FLOGIC 0, STANJAN
 ...
c using a list of species letters (AS method):
      call CHGLMP_MIX('STANJAN', 2009, 'XG', A100, 3, 0, 'OHW', 'PL')
c OR using a list of FPROP DATA identifiers (IDS method):
c     call CHGLMP_MIX('STANJAN', 2009, 'XG', A100, 3, NA200, ' ', 'PL')
c
c the species can be listed in any order, as long as XS is consistent
c with either AS or IDS.
```

As long as the quality (XL) is 1.0 (the gas is superheated or at least hotter than the dew point) such that no condensate forms, then specifying NS=2 in the above calls (perhaps shortening the NA200 array or the AS string to exclude species W) would have achieved the same result: if XGH=0.01 and XGO=0.02, then specifying XGW=0.97 overspecifies the above 3-species mixture. Overspecifying is not a problem as long as the sum of all specified species fractions does not exceed unity. Similarly, the sum of all specified fractions can be less than unity as long as some unspecified species are available in the submodel to make up the difference.

A variation of the above example that *implicitly* specifies the vapor concentration is as follows:

```
call CHGLMP_MIX('STANJAN', 9, 'XG', A100, 2, 0, 'OH', 'PL')
```

If liquid water were present or if it *could* be present (e.g., if the specified mixture exceeds 100% relative humidity at the current temperature and pressure), then whether or not the fraction for the condensible/volatile species W is specified *does* make a difference. If the species W is included in the XS list (first example above), then the program will make an extra effort to try to meet that request, including perhaps changing PL or TL (despite the value of HOLD) if necessary. If instead W is excluded from the list, then the program will be provided more freedom to reassign its concentration in order to better obey the HOLD criteria.

Equilibrium Concentration, with Nonreacting Species—To continue the above example, assume that N_2 is introduced as nonreacting species 'N' such that XGN=0.5 is preserved (perhaps as an initial condition in a closed tank, or via steady ingress of nitrogen). In this case, NS=3 and XS() array values should sum to 0.5, not 1.0 (unless species N were included in that list).*

Given the following:

```
HEADER ARRAY DATA, STANJAN
C these are often calculated by an equilibrium solver like STANJAN
      100 = 0.01, 0.005, 0.485
 ...
HEADER FLOGIC 0, STANJAN
 ...
c using a list of species letters (AS method):
      call CHGLMP_MIX('STANJAN', 9, 'XG', A100, 3, 0, 'OHW', 'PL')
```

the gas fractions would be XGH=0.01, XGO=0.005, XGW=0.485 assuming nitrogen (as the uninvolved species) took up the remainder at XGN=0.5.

* Similarities between CHGLMP_MIX and EQRATE should be apparent, but note that their use of "XS" is not the same. In EQRATE, the "XS" array refers to one chemical equation (there might be many operating in a single lump). In CHGLMP_MIX, the "XS" array refers to the whole lump's state.

If species W were condensible, to avoid changing PL or TL, the following alternative should be considered (specifying species N instead of W):

```
HEADER ARRAY DATA, STANJAN
C these are often calculated by an equilibrium solver like STANJAN
      100 = 0.01, 0.005, 0.5
 ...
HEADER FLOGIC 0, STANJAN
 ...
c using a list of species letters (AS method):
      call CHGLMP_MIX('STANJAN', 9, 'XG', A100, 3, 0, 'OHN', 'PL')
```

7.11.1.3 Controlling Slip and Nonequilibrium Modes

The routines documented in this subsection are:

NOSLIP . . . . . Force twinned paths to stay in homogenous mode
GOSLIP . . . . . Undoes NOSLIP actions: enables slip mode
NOTWIN . . . . Force twinned tanks to stay in homogenous mode
GOTWIN . . . . Undoes NOTWIN actions: enables nonequilibrium mode
SPLIT_TWIN. Forces equilibrium mode immediately

The routines NOSLIP and GOSLIP can be used to disable and re-enable slip flow modes in twinned paths. Parallel routines NOTWIN and GOTWIN disable and re-enable slip flow modes in twinned tanks (see also Section 3.25.6). However, twinned paths cannot remain combined when connected to active (split) twinned tanks. To help summarize the interactions between these routines, consider the following as guidance when applied to a network containing both twinned tanks and twinned paths:

1. It is not possible to disable slip flow modeling while enabling nonequilibrium modeling.
2. To disable nonequilibrium modeling *but not* slip flow modeling, call NOTWIN. Call GOTWIN to enable a future return to nonequilibrium modeling.
3. To disable nonequilibrium modeling *and* slip flow modeling, call NOSLIP, which will homogenize tanks at the same time. (Tanks pairs not attached to twinned paths may need to be separately homogenized using a call to NOTWIN.) To then re-enable just slip flow modeling, call GOSLIP. To re-enable both slip flow and nonequilibrium modeling, also (*subsequently*) call GOTWIN. Calling GOTWIN may be enough to re-enable slip flow as well, but some paths (not attached to twinned tanks) may require separate calls to GOSLIP, so a prior call to GOSLIP followed by a call to GOTWIN is recommended.

When OPERATIONS begins, any twinned tanks whose initial quality is neither 0.0 nor 1.0 *may* be in a split state, with the secondary twin active. If the user needs to perform initializations before calling STEADY, or for some other reason needs to have the model start with combined (homogenized) tanks, the NOTWIN routine should be used to combine the twins. Since NOTWIN will also prevent such twins from later splitting, a subsequent call to GOTWIN should be considered. GOTWIN will not split the tanks, rather it will *re-enable* them to be split later during network solutions. SPLIT_TWIN, on the other hand, will immediately split the tanks, including adjusting void fraction and volume if needed (if the tank is initially single-phase, for example).. See also Section 3.25.6.

Subroutine Name: NOSLIP*

Description: This routine is used to force twinned tubes and STUBE connectors to behave homogeneously, or at least restrains them from slipping. If the twins are currently in the slip flow mode, their flow rates will be summed into the primary path, the flow rate of the (now ignored) secondary path will be set to zero, and the phase suction status will be reset to NORM. If the paths are currently in the homogeneous mode, they will remain in that mode until GOSLIP is called to enable the slip mode.

NOSLIP can be used to restrain a single path pair, or to restrain all twinned paths in the named submodel.

Restrictions: If the ID of the input primary path is nonzero, it must be a twinned path or the program will abort. (It does not have to be currently in the slip flow mode; NOSLIP can be repeated on the same path without error.) Do not use in FLOGIC 1.

Restrictions: Twinned paths cannot be homogenized if they are connected to twinned lumps that are not themselves homogenized. Therefore, a call to NOSLIP implies that any twinned lumps to which these paths are attached are to be restrained or homogenized using an internal call to NOTWIN. *A subsequent call to GOSLIP does not by itself reverse this action*: the user must also call GOTWIN to permit restrained lumps to also split into twins.

Calling Sequence:

```
CALL NOSLIP (MOD, K)
```

where:

MOD . . . . . fluid submodel name, in single quotes
K. . . . . ID of primary path in MOD to be restrained to homogeneous flow. If zero (0), all twinned paths in the submodel will be restrained

Examples:

```
C
C     RESTRAINS (MAKES STAY HOMOG.) PATH 101 IN FRED:
C     (ANY TWINNED TANKS ON EITHER END OF THIS PATH WILL BE
C     HOMOGENIZED PER AN INTERNAL CALL TO NOTWIN)
C
       CALL NOSLIP ('FRED', 101)
C
C     RESTRAINS ALL TWINNED PATHS IN WILMA:
C     (ANY TWINNED TANKS ON EITHER END OF THESE PATHs WILL BE
C     HOMOGENIZED PER AN INTERNAL CALL TO NOTWIN)
C
       CALL NOSLIP ('WILMA', 0)
```

* Formerly "GOHOMO," which is still supported for backward compatibility.

Subroutine Name: GOSLIP

Description: This routine is used to reverse the effects of a prior call to NOSLIP, enabling twinned paths to resume slip flow simulation *if and when they are able.* (Slip flow does not exist in the limits of very high and very low void fractions.)

GOSLIP can be used to release a single path pair, or to release all twinned paths in the named submodel.

Restrictions: If the ID of the input primary path is nonzero, it must be a twinned path or the program will abort. (It does not have to be currently in the homogeneous mode; GOSLIP can be repeated on the same path without error.) Do not use in FLOGIC 1.

Caution: A call to GOSLIP does not reverse the effects of a prior call to NOSLIP with respect to twinned tanks, which may have been restrained or homogenized and which will require a *subsequent* call to GOTWIN to reverse that action.

Calling Sequence:

```
CALL SLIP (MOD, K)
```

where:

MOD ..... fluid submodel name, in single quotes
K ..... ID of primary path in MOD to be released to resume slip flow. If zero (0), all twinned paths in the submodel will be released

Examples:

```
C
C     PATH 101 IN FRED WILL RESUME SLIP FLOW WHEN ABLE:
C
       CALL GOSLIP ('FRED', 101)
C
C     RELEASES ALL TWINNED PATHS IN WILMA:
C
       CALL GOSLIP ('WILMA', 0)
```

Subroutine Name: NOTWIN

Description: This routine is used to force twinned tanks to behave homogeneously, or at least restrains them from splitting. If the twins are currently in the split mode, their masses and energies will be summed into the primary tank, and the superpaths, fties, and ifaces between them will become inactive. If the twinned tanks are currently in the homogeneous mode, they will remain in that mode until GOTWIN is called to enable the nonequilibrium mode.

NOTWIN can be used to restrain a single tank pair, or to restrain all twinned tanks in the named submodel.

Restrictions: If the ID of the input primary tank is nonzero, it must be a twinned tank or the program will abort. (It does not have to be currently in the nonequilibrium mode; NOTWIN can be repeated on the same tank without error.) Do not use in FLOGIC 1.

Guidance: This routine does not prevent attached twinned paths from slipping. When a tank is homogenized, any secondary paths will be reattached to the primary tank, and the phase suction options will be adjusted accordingly.

Calling Sequence:

```
CALL NOTWIN (MOD, L)
```

where:

MOD . . . . . fluid submodel name, in single quotes
L. . . . . ID of primary tank in MOD to be restrained to homogeneous. If zero (0), all twinned tanks in the submodel will be restrained

Examples:

```
C
C     RESTRAINS (MAKES STAY HOMOG.) TANK 101 IN FRED:
C
       CALL NOTWIN ('FRED', 101)
C
C     RESTRAINS ALL TWINNED TANKS IN WILMA:
C
       CALL NOTWIN ('WILMA', 0)
```

Subroutine Name: GOTWIN

Description: This routine is used to reverse the effects of a prior call to NOTWIN, enabling twinned tanks to resume nonequilibrium simulation *if and when they are able.*

GOTWIN can be used to release a single tank pair, or to release all twinned tanks in the named submodel.

Restrictions: If the ID of the input primary tank is nonzero, it must be a twinned tank or the program will abort. (It does not have to be currently in the homogeneous mode; GOTWIN can be repeated on the same tank without error.) Do not use in FLOGIC 1.

Caution: Enables slip flow (internal GOSLIP calls) as needed, since a tank cannot be split unless any twinned paths flowing into or out of that tank are also split. However, twinned paths carrying single-phase flow INTO a pair of twinned tanks may still be remain homogeneous and will flow just into the primary or secondary.

Calling Sequence:

```
CALL GOTWIN (MOD, L)
```

where:

MOD ..... fluid submodel name, in single quotes
L ..... ID of primary tank in MOD to be released to resume nonequilibrium modeling. If zero (0), all twinned tanks in the submodel will be released

Examples:

```
C
C     TANK 101 IN FRED WILL RESUME NONEQUILIBRIUM WHEN ABLE:
C
       CALL GOTWIN ('FRED', 101)
C
C     RELEASES ALL TWINNED TANKS IN WILMA:
C
       CALL GOTWIN ('WILMA', 0)
```

Subroutine Name: SPLIT_TWIN

Description: This routine is used to immediately split a homogeneous (combined mode) twinned tank into split (nonequilibrium mode): a forceful version of GOTWIN that may change the tank's thermodynamic state if needed to enable the split. This routine reverse the effects of a prior call to NOTWIN.

SPLIT_TWIN can be used to split a single tank pair, or to split all twinned tanks in the named submodel. It is used primarily in OPERATIONS to enable initializations (subsequent calls to CHGLMP, CHGVOL, etc.).

Restrictions: If the ID of the input primary tank is nonzero, it must be a twinned tank or the program will abort. (It does not have to be currently in the homogeneous mode; SPILT_TANK can be repeated on the same tank without error.) Do not use in FLOGIC 1.

Caution: Enables slip flow as needed, since a tank cannot be split unless any twinned paths flowing into or out of that tank are also split. However, twinned paths carrying single-phase flow INTO a pair of twinned tanks may still be remain homogeneous and will flow just into the primary or secondary tank.

Caution: If the tank to be split is single-phase (or if it is too dry or too wet to be considered two-phase), SPLIT_TWIN will change the void fraction and quality *and perhaps the temperature or pressure* of the tank as needed to establish the missing (or insufficient) phase.

Calling Sequence:

```
CALL SPLIT_TWIN (MOD, L)
```

where:

MOD . . . . . fluid submodel name, in single quotes

L. . . . . ID of primary tank in MOD to be split into nonequilibrium modeling. If zero (0), all twinned tanks in the submodel will be split.

Examples:

```
C
C     SPLITS TANK 101 IN FRED. IF 102 is the vapor (secondary) twin,
C     it will be present immediately after the call (VOL102>0),
C     such that CHGLMP calls are valid as shown.
C
       CALL SPLIT_TWIN ('FRED', 101)
       CALL CHGLMP('FRED', 102, 'TL', FRED.TL101+10.0, 'XL')
C
C     SPLITS ALL TWINNED TANKS IN WILMA:
C
       CALL SPLIT_TWIN ('WILMA', 0)
```

7.11.1.4 Flat Front (Fill/Purge) Duct Modeling Modes

Flat-front modeling (Section 3.27) is not an appropriate assumption to make if significant phase change occurs: if the liquid boils and/or the vapor condenses. Furthermore, flat-front modeling implicitly assumes homogeneous (nonslip) flow, so the rising of bubbles through the liquid column and the falling of droplets through the vapor column cannot be modeled with a flat-front approach since those effects require twinned paths (slip flow).

This limitation can be overcome using the DUCT_MODE routine, which allows the use of "normal" (non-flat-front or "dispersed front") two-phase methods away from the main liquid/vapor front. This alternative approach is applicable for cases where low levels of boiling or vapor/gas injection occur in the liquid column, or where low levels of condensation or liquid injection occur in the vapor/gas column, but there should be a single distinct phase change front within the duct.

By default, a flat-front duct uses IPDC=-5 (axially stratified flow) in all parts of the duct. If a void appears within the liquid column, or if a droplet appears within the vapor column, this is treated as a second front, which can lead to undesirable predictions and even crashes. Instead, the user may wish to leave bubbles in the liquid column, or perhaps let them rise to the surface. Similarly, droplets may be left in the vapor column, or perhaps allowed to fall to the surface. These behaviors are enabled by choosing FF_ON1: flat front is "on" but is only active at the front itself. Away from the liquid/vapor front, "normal" or "dispersed front" two-phase options are permitted. Variations of this mode include suppression of twinned tanks and/or slip flow, in the event that the flat-front duct has been modeled using twinned tanks and therefore twinned paths (as recommended), but these choices are not needed away from the front (which is usually the case).

Subroutine Name: DUCT_MODE

Description: This routine changes one or more flat-front (FF=FILL/PURGE) duct macros from the default mode (corresponding to FF_ON) to an alternate mode that restrict the flat-front methods to a single front (FF_ON1, including variations FF_ON1_S and FF_ON1_H which restrict twinned tanks and twinned paths away from the front).

Flat-front modeling can also be disabled entirely, returning the duct to a "normal" UP or DOWN discretized model using FF_OFF. However, note the centered discretization is preferred for two-phase models, so this option should only be used sparingly.

DUCT_MODE can be called numerous times as needed with different mode options.

Restrictions: Do not use within FLOGIC 1 blocks. A single front location will be tracked: multiple front locations are not permitted except in the FF_ON (default) mode. If the tanks are twinned and active, this front is assumed to be the first pair of active (split) twinned tanks that is encountered from the dry end, so use FF_ON1_S or FF_ON1_H if tanks are twinned and condensation is extensive.

Restriction: Do not use expressions for IPDC and UPF, or at least do not change any underlying registers in such expressions during execution. The IPDC and UPF values will be changed by DUCT_MODE. When flat-front is suppressed, IPDC will be changed to 6, and UPF to 0.5. When flat-front is restored, IPDC will be changed to -5 and UPF will be changed to either 0.0 (FF=FILL) or 1.0 (FF=PURGE).

Restriction: This routine overrides GOSLIP/NOSLIP/GOTWIN/NOTWIN calls that have been made on lumps or paths within flat-front duct macros, and such calls should not be made independently on such elements while using DUCT_MODE.

Calling Sequence:

```
CALL DUCT_MODE (MODNAM, MAC_ID, MODE)
```

where:

`MODNAM` . . . . . fluid submodel name containing duct to be modified, input in single quotes. Use ‘ALL’ to affect all fluid submodels.

`MAC_ID` . . . . . identifier of duct macro to be changed (integer). Specify zero to affect all suitable (fill/purge) ducts within this (or all) submodels. Ducts that have not been designated as FF=FILL or FF=PURGE will be ignored.

`MODE` . . . . . duct mode to set, in single quotes:

- ‘FF_ON’ . . . . . All flat-front options active (original/default)
- ‘FF_ON1’ . . . . Flat-front options active only at front, rest “normal”
- ‘FF_ON1_S’ . . Same as above, except twinned tanks suppressed (slip only) away from the front, as if NOTWIN had been called
- ‘FF_ON1_H’ . . Same as above, except slip flow also suppressed (all homogeneous) away from the front, as if NOSLIP had been called
- ‘FF_OFF’. . . . . All flat-front options inactive (all “normal” two-phase)

Example:

```
C Change all flat-front ducts to use homogeneous methods away from front
      CALL DUCT_MODE ('ALL', 0, 'FF_ON1_H')
C Turn off all flat-front modeling in duct #3 in submodel EVFLO
      CALL DUCT_MODE ('EVFLO', 3, 'FF_OFF')
```

7.11.1.5 Lump Solution Mode Options

This section describes utilities that can change the way in which tanks and junctions operate:

HLDLMP . . . . Hold lump thermodynamic state
HLDTANKS. . Holds all tank thermodynamic states
RELTANKS . . Releases all tank thermodynamic states
HTRLMP . . . . Hold lump enthalpy (temperature or quality)
RELLMP . . . . Release HLDLMP and HTRLMP actions

Subroutine Name: HLDLMP

Description: This routine is used to make a tank or junction act temporarily as a plenum or boundary. In other words, it "holds" the lump in a reference state. This action will speed the solution process and may be used as part of control logic. (See also RELLMP and HTRLMP.)

HLDLMP *must* be called for at least one lump within a fluid submodel prior to a STEADY ("FASTIC") run if no plena are present. Otherwise, it may be called at any time in any logic block except FLOGIC 1. HLDLMP has no effect on plena. The output routine LMXTAB describes the status of any holds placed on lumps.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no holds placed. HTRLMP overrides previous HLDLMP calls and vice versa. RELLMP releases all previous holds. CHGLMP (Section 7.11.1.1) may be used concurrently. *Other restrictions apply when used in combination with ifaces, as described in Section 3.8.6.3.*

If plenum thermodynamic states (PL, TL, XL), which are program *inputs*, are defined using expressions, then those plenum states will be updated automatically as needed during a run. Such thermodynamic state expressions represent initial conditions for a tank or junction since the states of those lumps are normally program *outputs*: changes to the underlying expressions do not affect tank and junction states, whether or not HLDLMP was invoked. Use CHGLMP (Section 7.11.1.1) in FLOGIC 0 as needed to adjust the states of held tanks and junctions.

Calling Sequence:

```
CALL HLDLMP (MODNAM, LUMPNO)
```

where:

MODNAM . . . . fluid submodel name containing lump to be held, input in single quotes
LUMPNO . . . . identifier of lump to be held

Example:

```
CALL HLDLMP ('TMS', 100)
```

Subroutine Name: HLDTANKS

Description: This routine is used to make all tanks in a fluid submodel act temporarily as a plenum or boundary. In other words, it "holds" the tanks in a reference state. This action, *equivalent to calling HLDLMP for all tanks in a submodel*, is useful for calling steady state solutions to update junctions and paths without disrupting specially selected initial conditions in tanks.

For example, in a model of a gas tank being blown down (depressurized) through an exhaust piping network, the supply tanks should be held full during steady states, but a piping network comprised of junctions and connectors should be initialized before starting a transient.

RELTANKS (described below) reverses the actions of HLDTANKS. The output routine LMX-TAB describes the status of any holds placed on lumps.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no holds placed. HTRLMP and HLDLMP overrides previous HLDTANKS calls and vice versa. CHGLMP may be used concurrently. *Other restrictions apply when used in combination with ifaces, as described in Section 3.8.6.3.*

If plenum thermodynamic states (PL, TL, XL), which are program *inputs*, are defined using expressions, then those plenum states will be updated automatically as needed during a run. Such thermodynamic state expressions represent initial conditions for a tank since the states of tanks are normally program *outputs*: changes to the underlying expressions do not affect tank states, whether or not HLDLMP was invoked. Use CHGLMP (Section 7.11.1.1) in FLOGIC 0 as needed to adjust the states of held tanks.

Calling Sequence:

```
CALL HLDTANKS (MODNAM)
```

where:

MODNAM . . . . . fluid submodel name containing tanks to be held, input in single quotes

Example:

```
CALL HLDTANKS ('BLOWDOWN')
CALL STEADY
CALL RELTANKS('BLOWDOWN') $ see documentation below
CALL TRANSIENT
```

Subroutine Name: HTRLMP

Description: This routine is used to hold the enthalpy of a junction, making it a "heater junction." Constant enthalpy means nearly constant temperature or quality (for single-constituent two-phase working fluids) as long as pressure does not change significantly. CHGLMP can be used

within FLOGIC 0 to maintain constant temperatures or qualities in case pressure *does* change significantly; CHGLMP can be used to make indirect changes to the lump enthalpy. (See also RELLMP.) The output routine LMXTAB describes the status of any holds placed on lumps.

When a junction (or STEADY tank) is in the heater mode via a HTRLMP call, then the power required to maintain the current enthalpy (HL) is calculated and is available as the junction QDOT parameter. In other words, instead of calculating the enthalpy given the currently imposed heating rate (which is the normal solution mode for a junction), a HTRLMP call reverses the solution mode such that the imposed heating rate is calculated given the current enthalpy. Mathematically, in the normal solution mode HL = f(QL,FR ...) whereas in the HTRLMP mode QDOT = f(HL,FR ...), with QL being an input parameter and QDOT being an output parameter.

If a heater junction is tied to a node or has an FTIE to a lump, then the QDOT required to maintain the junction at the current enthalpy is assumed to come from that node and/or those lumps. The QTIEs and FQs for such ties and FTIEs do not represent any real heat transfer process, but their value will be at least consistent with the operation of the rest of the program. If the junction is connected to more than one node and/or lump, then the required QDOT is distributed amongst the nodes and lumps according to the relative “strengths” of each tie and FTIE based on the QTIEs and QFs that *would* have existed without the HTRLMP action.

As the name implies, heater junctions may be used like heater nodes for sizing or simulating idealized heaters. However, because they behave differently from any other network element, they can be used for many modeling purposes such as temperature or quality control systems, oversized or perfect heat exchangers, capillary pumps (by holding at saturated vapor and adding an MFRSET connector), etc. Refer to Sample Problems E and F for examples of HTRLMP usage.

The user will recall that tanks are treated like junctions in every way within STEADY. Thus, the enthalpy of tanks may also be held in STEADY, *but such holds are removed at the start of other solution routines.* The HTRLMP call must be repeated prior to any STEADY calls that follow STDSTL or transient calls, including those inside of repetitive procedures (PROCEDURE, REL-PROCEDURE, etc.).

Caution: After releasing an untied heater lump, the last applicable QDOT value will remain in effect as a constant unless the user overrides this value. This is especially important for constant enthalpy tanks in STEADY since they are considered “released” in subsequent calls to other solution routines.

Caution: If either the temperature or quality of a plenum (TL or XL, both of which are program *inputs)* is defined using an expression containing a register or processor variable, then that plenum value will be updated automatically as needed during a run. Such thermodynamic state expressions represent initial conditions for a tank or junction since the states of those lumps are normally program *outputs*: changes to the underlying expressions do not affect tank and junction states, whether or not HTRLMP was invoked. Therefore, use CHGLMP (Section 7.11.1.1) in FLOGIC 0 as needed to adjust the temperature or quality of heater junctions.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no holds placed. HLDLMP overrides previous HTRLMP calls and vice versa. RELLMP releases all holds.

Calling Sequence:

```
CALL HTRLMP (MODNAM, LUMPNO)
```

where:

MODNAM . . . . . fluid submodel name containing lump to be held, input in single quotes
LUMPNO . . . . . identifier of lump to whose enthalpy is to be held

Example:

```
CALL HTRLMP ('TMS', 100)
```

Subroutine Name: RELLMP

Description: This routine releases a tank or junction from holds placed by HLDLMP or HTRLMP, returning it to normal behavior. The output routine LMXTAB describes the status of any holds placed on lumps.

Note that returning a tank or junction to normal status from a HLDLMP hold immediately forces a mass and energy balance where none was required before. Small time steps and/or many iterations may result until the changes are absorbed.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no releases made.

Calling Sequence:

```
CALL RELLMP (MODNAM, LUMPNO)
```

where:

MODNAM . . . . . fluid submodel name containing lump to be released, input in single quotes
LUMPNO . . . . . identifier of lump to be released

Example:

```
CALL RELLMP ('TMS', 100)
```

Subroutine Name: RELTANKS

Description: This routine is used to release all tanks in a fluid submodel. This action, *equivalent to calling RELLMP for all tanks in a submodel*, is useful for releasing prior holds on tanks in anticipation of a subsequent transient analysis.

RELTANKS reverses the actions of HLDTANKS (described above). The output routine LMX-TAB describes the status of any holds placed on lumps.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no holds placed.

Calling Sequence:

```
CALL RELTANKS (MODNAM)
```

where:

MODNAM .... fluid submodel name containing tanks to be released, input in single quotes. If omitted, or if 'ALL' is input, all held tanks are released in all fluid submodels.

Example:

```
CALL RELTANKS('BLOWDOWN')
```

7.11.1.6 Dynamic Translation Utilities

This section documents utilities used to perform "dynamic translations:" to look up the internal sequence number for an element, fluid, or submodel:

INTLMP . . . . . Return internal lump sequence number
INTPAT . . . . . Return internal path sequence number
INTTIE . . . . . Return internal tie sequence number
INTFTI. . . . . Return internal ftie sequence number
INTIFC . . . . . Return internal iface sequence number
INTPTS . . . . . Return internal subpath sequence number
INTFLD. . . . . Return internal working fluid pointer
INTSPE . . . . . Return internal species pointer
MFLTRN . . . . . Return fluid submodel sequence number

Function Name: INTLMP

Description: This routine provides the user with the internal location pointer for a given lump. It is analogous to routine NODTRN. This sequence number can be used to directly address the arrays TL, PL, HL, DL, XL, AL, DF, DG, CX, CY, CZ, and QDOT using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix). The arrays VOL, VDOT, and COMP can similarly be addressed for tanks only.

Guidance: Note that lumps are stored internally in the order: tanks, junctions, and then plena. Within each category the lumps are stored by submodel, and within each submodel they are stored according to the order in which they were input.

Input errors will result in a program abort.

Calling Sequence:

```
LSEQ = INTLMP (MODNAM, LUMPNO)
```

where:

LSEQ. . . . . internal sequence number of lump (program designation)
MODNAM . . . . . fluid submodel name containing lump, input in single quotes
LUMPNO . . . . . identifier of lump (user designation). Negative LUMPNO signals to return -1 instead of abort if lump is not found.

Examples:

```
      LTEST = INTLMP ('DERF', 332)
      QTEST = WALL.G10*GLOBAL.TL(LTEST)
C OR EQUIVALENTLY:
      QTEST = WALL.G10 *
F    +  TL(INTLMP('DERF',332))
C OR EQUIVALENTLY:
      CTEST = WALL.G10*GLOBAL.TL(INTLMP('DERF',332))
```

Function Name: INTPAT

Description: This routine provides the user with the internal cell number for a given path. (See INTPTS for addressing subpaths.) This sequence number can be used to directly address the arrays FR, TLEN, DH, AF, FC, FPOW, AC, HC, UPF, DUPI, DUPJ, and IPDC using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix).

Guidance: Note that paths are stored internally in the order tubes and then connectors. Within each category the paths are stored by submodel, and within each submodel they are stored according to the order in which they were input.

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTPAT (MODNAM, NPATH)
```

where:

KSEQ.....	internal sequence number of path (program designation)
MODNAM	fluid submodel name containing path, input in single quotes
NPATH.....	identifier of path (user designation). Negative NPATH signals to return -1 instead of abort if path is not found.

Examples:

```
      K = INTPAT ('TMS', 299)
      DO 10 K = 1,10
F10         WRITE(NOUT,*) 'FLOW RATE = ', FR(K)
```

Function Name: INTTIE

Description: This routine provides the user with the internal cell number for a given tie. This sequence number can be used to directly address the UA, QTIE, DUPN, and DUPL arrays using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix).

Guidance: Note that ties are stored internally in the input order.

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTTIE (MODNAM, NTIE)
```

where:

KSEQ. . . . . internal sequence number of tie (program designation)
MODNAM . . . . . fluid submodel name containing tie, input in single quotes
NTIE. . . . . identifier of tie (user designation). Negative NTIE signals to return -1 instead of abort if tie is not found.

Examples:

```
K = INTTIE ('TMS', 544)
GLOBAL.UA(INTTIE('FRD',ITEST)) = UTEST
```

Function Name: INTFTI

Description: This routine provides the user with the internal cell number for a given ftie. This sequence number can be used to directly address the GF, QF, DUPC, and DUPD arrays using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix).

Guidance: Note that fties are stored internally in the input order.

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTFTI (MODNAM, NFTIE)
```

where:

KSEQ. . . . . internal sequence number of ftie (program designation)
MODNAM . . . . . fluid submodel name containing ftie, input in single quotes
NFTIE . . . . . identifier of ftie (user designation). Negative NFTIE signals to return -1 instead of abort if ftie is not found.

Examples:

```
K = INTFTI ('TMS', 1101)
GLOBAL.GF(INTFTI('GARG', ITEST)) = GTEST
```

Function Name: INTIFC

Description: This routine provides the user with the internal cell number for a given iface. This sequence number can be used to directly address the iface arrays (e.g., VHI, VLO, VB, PA, DUPA, DUPB, etc.) using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix).

Guidance: Note that ifaces are stored internally in the input order.

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTIFC (MODNAM, NFACE)
```

where:

KSEQ	internal sequence number of iface (program designation)
MODNAM	fluid submodel name containing iface, input in single quotes
NFACE.....	identifier of iface (user designation). Negative NFACE signals to return -1 instead of abort if iface is not found.

Examples:

```
K = INTIFC ('TMS', 544)
GLOBAL.VHI(INTIFC('SLAM',ITEST)) = VTEST
```

Function Name: INTPTS

Description: This routine provides the user with the internal cell number for a given subpath. This sequence number can be used to directly address the path arrays (e.g., GK, HK, etc.) using F-type statements (or by turning translation off locally using the GLOBAL submodel prefix).

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTPTS (MODNAM, UPATH, ARG)
```

where:

KSEQ. . . . . internal sequence number of subpath (program designation)
MODNAM . . . . . fluid submodel name containing subpath, input in single quotes
UPATH. . . . . identifier of the superpath (user designation). Negative UPATH signals to return -1 instead of abort if super path is not found.
ARG . . . . . ‘LIQ’, ‘VAP’, or ‘x’ where x is the letter designator of the desired species subpath TSPECx, such as ‘W’ to address the TSPECW subpath.

Examples:

```
K = INTPTS ('TMS', 1101, 'LIQ')
GLOBAL.GK(INTPTS('GLURP', ITEST, 'G')) = 0.0
```

Function Name: INTFLD

Description: This routine provides the user with the internal pointer into the fluid property array FI. FI is used as an argument in various property calculation routines (Section 7.11.2).

INTFLD can be used to locate the pointer into the FI array either as a function of fluid submodel or as a function of the working fluid ID number.

Although it can be used for other purposes, the primary purpose of this routine is to enable the user to make logical manipulations and calculations using property routine calls even if the working fluid has changed via a call to CHGFLD. Applications of INTFLD are few if CHGFLD is never used.

Input errors will result in a program abort.

Calling Sequence:

```
KFLD = INTFLD (MODNAM, NFLD)
```

where:

KFLD. . . . . internal sequence number of the fluid (pointer into FI array)
MODNAM . . . . . fluid submodel name containing fluid, input in single quotes
NFLD. . . . . working fluid identifier. Negative NFLD signals to return -1 instead of abort if fluid ID is not found.

If MODNAM is a valid fluid submodel, then NFLD is ignored. Otherwise, if MODNAM is invalid, the value of NFLD is applied. If both are incorrect or invalid, the program will abort.

On some systems, empty quotes (‘’) cannot be input as a purposely invalid model name without generating a compiler error. For such systems, simply use a space (‘ ’) as an invalid model name.

Example:

```
C PRPMAP OF THE FLUID CURRENTLY BEING USED BY SUBMODEL FRED
F     CALL PRPMAP(300.0, 400.0, FI(INTFLD('FRED',0)))
C NOTE THAT UNLESS CHGFLD IS CALLED, THIS IS NORMALLY EASIER:
      CALL PRPMAP(300.0, 400.0, FRED.FI)
C
C PRPMAP OF THE FLUID 6070
F     CALL PRPMAP(300.0, 400.0, FI(INTFLD(' ',6070)))
C NOTE THAT THIS IS NORMALLY EASIER FOR THE ABOVE:
      CALL PRPMAP(300.0, 400.0, FI6070)
```

Note that FSTART or an F in column 1 must be used to prevent preprocessor translation errors. Unlike other parameters, "GLOBAL." cannot be used with FI to turn off translation within a line.

Function Name: INTSPE

Description: This routine provides the user with the internal cell number for a given fluid species. This integer address, combined with the result from INTLMP (see above, in this subsection), can be used to directly address cells of the arrays XG, XF, PPG, and MF. For example:

```
F     PTEST = PPG( INTSPE('FMOD','H'), INTLMP('FMOD',504) )
F     XTEST = XF( INTSPE('OTHER','A'), INTLMP('OTHER',77) )
```

is equivalent to:

```
      PTEST = FMOD.PPGH504
      XTEST = OTHER.XFA77
```

As can be generalized from the above examples, the XG* data for species A through Z is stored in a two-dimensional array named XG, and the same is true for XF*, PPG*, and MF*. However, it is not necessarily true that A=1, B=2, etc. since rarely will all 26 possible species be used. Thus, if junction #40 is stored internally in the 503rd cell of TL, PL, etc. per the results of INTLMP, and if species R is the 11th species per INTSPE, then both "XGR40" and "XGR(40)" will be translated by the preprocessor as "XG(11,503)."

Guidance: Note that species are stored internally in the order in which they appear in the alphabet.

Input errors will result in a program abort.

Calling Sequence:

```
KSEQ = INTSPE(MODNAM, SPEC)
```

where:

KSEQ. . . . . internal sequence number of species (program designation)
MODNAM . . . . . fluid submodel name containing species, input in single quotes
SPEC. . . . . single character identifier of species (user designation), input in single quotes. A negative sign in front of SPEC signals to return -1 instead of abort if the species is not found.

Example:

```
K = INTSPE ('GOFLO','M')
```

Function Name: MFLTRN

Description: This function returns the relative location of a fluid submodel name in the array of submodel names (as defined by the BUILDF records).

Restrictions and Guidance: This also allows calls to many internal routines. Invalid fluid submodel names will *not* result in an abort; rather, a zero is returned. The user should include checks against returned zero values, since use of a zero array pointer will cause unexpected results.

Calling Sequence:

```
ISUB   = MFLTRN(MODNAM)
```

where:

MODNAM . . . . . a fluid submodel name enclosed in single quotes
ISUB. . . . . the returned submodel sequence number

Example:

```
F     XTEST = OUTPTF(MFLTRN('PATF'))
```

7.11.1.7 Capillary Modeling Utilities

This subsection documents the following utility:

SPRIME . . . . . Force a capillary device to stay primed

Subroutine Name: SPRIME

Description: This routine, *when called from FLOGIC 0*, can be used to force a capillary device or component model (CAPIL connectors, CAPPMP macros) to maintain a primed state. This is particularly useful during steady-state runs when capillary devices can easily deprime due to artificial pressure or quality fluctuations. Once deprimed, such devices tend to stay deprimed indefinitely. The user *must assume a capillary device stays primed and then verify that a viable solution can be achieved with that assumption.* Attempting to run without this routine will invariably result in a steady-state solution where the device is deprimed. This is a valid answer, but not always desirable. See also Section 3.11.2.

This routine uses internal calls to CHGLMP (Section 7.11.1.1) such that the following conditions are met:

1) XL of liquid lump = 0.0 (set the liquid lump ID negative to avoid this action)

2) XL of vapor lump = 1.0

3) The pressure difference between the vapor and the liquid lumps is nonnegative and no greater than the capillary structure can maintain (two times the surface tension divided by the capillary radius). Pressure drops internal to the CAPPMP macro (which are inversely proportional to the input CFC) are neglected.

4) The vapor lump temperature is subcritical.

Messages can be printed to show the changes needed to keep the device primed. Such messages should be ignored if they occur only at the start of STDSTL or STEADY, but should not be ignored if they persist or if they appear in a transient run. Persistent, repetitive adjustments made by SPRIME at the end of a nonconvergent steady state run mean that the device cannot stay primed under the given circumstances. The adjustments made should give some clues as to why the device won’t stay primed.

If modeling a loop heat pipe (LHP) or some other system in which the liquid side need not be completely cleared of vapor to stay primed, signal this with a negative sign on the liquid lump identifier. Although all vapor will not be cleared, some liquid may be introduced to maintain prime.

Guidance: Use the NSOL flag within FLOGIC 0 blocks to determine the solution routine currently being used. For CAPIL connectors and CAPPMP macros, the values of RC, XVH, and XVL can be adjusted to make the model more resistance to deprime with or without SPRIME calls. See Section 3.11.2.

Restrictions: Do not use within FLOGIC 1 blocks. Input errors result in no changes made. SPRIME cannot account for internal pressure drops within the wick, and these drops are accounted for in CAPPMP macros. In other words, if the resistance in the wick is substantial compared to the capillary pressure gain, SPRIME cannot guarantee reprime. In these cases, the user might consider adding a degradation factor to the value of RC passed to the SPRIME routine (compared to the actual RC) to account for internal resistances.

Calling Sequence:

```
CALL SPRIME (MODNAM, LMPLIQ, LMPVAP, RC, MESS)
```

where:

MODNAM . . . . . fluid submodel name containing device or component to be primed, input in single quotes such as 'LOOP'

LMPLIQ . . . . . identifier of liquid lump (negative to signal LHP option)

LMPVAP . . . . . identifier of vapor lump

RC . . . . . current effective two-dimensional capillary radius (for CAPIL and CAPPMP this variable may be specified as RCn, where n is the identifier of the CAPIL connector or the first path of the CAPPMP model.) May be non-positive, which signals an infinite pressure difference capacity.

MESS. . . . . Message flag. If nonzero prints a messages describing the changes needed to stay primed, otherwise no messages are printed.

Examples:

```
HEADER FLOGIC 0, LOOP
       IF(NSOL .LE. 1) CALL SPRIME ('LOOP', 10, 20, RC20,1)
       CALL SPRIME ('HOTSYS', 30, 31, RC10,1)
       CALL SPRIME ('MODELA', -100, 200, 0.0001,1)
       CALL SPRIME ('CAPILY', 15, 20, -1.0, NSOL)
```

7.11.1.8 Moving a Path or Tie

This subsection documents utilities that can be used to dynamically relocate a path or tie. The user should, however, avoid these manipulations in favor of other methods* when using graphical user interfaces (Sinaps® or FloCAD®) which cannot accommodate such usage.

PUTTIE . . . . . Move a tie from one lump and/or node to another
PUTPAT . . . . . Move a path from one lump to another

Subroutine Name: PUTTIE

Description: This routine allows users to dynamically relocate heat transfer ties during processor execution. HTU, HTN, and HTNC ties may be moved to other lumps in the same submodel, and/or moved to any other node in any valid submodel. Ties always belong in the same submodel in which they were input, and therefore cannot be moved to a lump in other fluid submodels.

When a tie moves, it retains its identity (e.g., HTU), and its current values of UA, QTIE, DUPN, and DUPL. In fact, its internal storage location (i.e., the value returned by INTTIE) does not change. Because the duplication factors are invariant, the user should exercise caution when moving the lump from one duplicated section of the network to another. If the new node is not in the same submodel as the old node, all that is required is that the new submodel name be valid. No other rule changes are involved—the tie behaves as if it had been originally located at that node. If the new submodel is not currently built, the tie becomes adiabatic. If the new submodel is built but is dormant (via a DRPMOD call), the node is treated as if it were a boundary node. PUTTIE may be called repetitively in FLOGIC blocks even if no changes are expected; no action is taken if the designated node and lump correspond to the current location of the tie.

Restrictions: Do not use in FLOGIC 1. Do not attach additional ties to CAPPMP junctions. No action will be taken if the specified tie is not an HTU, HTN, or HTNC tie. When all ties have been removed from a lump, the QDOT will be reset to zero and may be altered by the user. Conversely, moving a tie to a lump that previously had no ties will override its QDOT value. Calls to RESTAR or RESTNC do not affect prior PUTTIE calls in the current run—*the locations of ties in previous runs are ignored as are PUTTIE calls made in those runs.*

PUTTIE internally calls INTTIE, INTLMP, and NODTRN. Hence, with the exception of zeroes used as flags (as defined below), input errors will result in a program abort.

Calling Sequence:

```
CALL PUTTIE(MODNAM, NTIE, LUMP, MODTHM, NODE)
```

* For example, turning on or off extra ties and paths using perhaps duplication factors.

where:

MODNAM . . . . . fluid submodel name containing tie, input in single quotes
NTIE. . . . . identifier of tie to be moved (user designation). HTU, HTN, or HTNC only.
LUMP. . . . . identifier of new lump in same submodel (user designation), or zero if the lump remains the same
MODNAM . . . . . thermal submodel name containing the node, input in single quotes
NODE. . . . . identifier of new node (user designation), or zero if the node remains the same

To move the tie to a new node without changing the lump, set the lump number to zero. To move the tie to a new lump without changing the node, set the node number to zero, in which case the thermal submodel input is ignored and can be anything.

Examples:

```
CALL PUTTIE('OK', 4559, 455, 'THIS', 10)
CALL PUTTIE('IS', 59, 0, 'POLITCLY', 7)
CALL PUTTIE('CORRECT', 1422, 999, ' ', 0)
```

Subroutine Name: PUTPAT

Description: This routine is used to move the end lumps of a path. Primarily intended to move untwinned paths to active lumps of twinned tanks.

Restrictions: Cannot be used to move twinned paths.

Calling Sequence:

```
CALL PUTPAT (MOD, PATHID, LU, LD)
```

where:

MOD . . . . . fluid submodel name, in single quotes
PATHID . . . . . User path ID
LU . . . . . ID of the upstream lump path is to be attached to. Zero entry will result in no change to the upstream lump placement.
LD . . . . . ID of the downstream lump path is to be attached to. Zero entry will result in no change to the downstream lump placement.

Example:

```
C
C     PATH 40 IN LOOP WILL FLOW INTO TANK 101 LEAVING UPSTREAM
C     LUMP UNCHANGED
C
       CALL PUTPAT ('LOOP', 40,0,101)
```

7.11.1.9 Auxiliary Calculations

This subsection documents the following auxiliary calculations:

MACONE . . . . Reinitialize the geometry of a LINE or HX macro
FKCONE . . . . . Calculates K-factors (FK) for conical expanders and reducers

Subroutine Name: MACONE

Description: This routine helps the user initialize or reinitialize circular-cross section duct macros (LINE or HX), whether flow areas vary or not. When flow areas do vary, a conic shape (linearly varying diameter) is assumed, in contrast to the FLOW DATA approach of linearly varying flow area, which results in a horn shape.

When MACONE is called, the DH, AFTH, AF, AFI, AFJ, and TLEN of all macro paths are updated to reflect the requested shape, assuming that the diameter varies linearly with length and that the cross section is circular. The VOL of any tanks inside the macro are similarly updated.

If DI and DJ are equal, the updates are still applied, but AFI and AFJ are set to =-1.0 (signaling a constant cross sectional flow area). This makes MACONE a useful means of updating macros even if they have a constant flow area.

[If the key dimensions are defined via register-containing formulas, the user will find it easier to make adjustments using the dynamic register features (Section 4.9), which render MACONE obsolete.]

Calling Sequence:

```
CALL MACONE(MODNAM, MACNUM, DI, DJ, TLENT)
```

where:

MODNAM . . . . . fluid submodel name containing macro, input in single quotes
MACNUM . . . . . macro identifier number
DI . . . . . diameter at defined inlet of macro
DJ . . . . . diameter at defined outlet of macro
TLENT . . . . . total length of the macro

Input mistakes result in cautions being issued, and in no changes being made.

Examples:

```
CALL MACONE('WET', 12, 0.5/12.0, 1.0/12.0, 5.0)
CALL MACONE('MYMOD',39, 0.01, 0.01, 0.5)
```

Subroutine Name: FKCONE

Description: This routine performs the calculations of the irrecoverable loss factors for conical reducers and expanders, per Section 3.15.2.4. The AC factor should also be applied to account for recoverable losses per that section by using the AFI/AFJ factors, perhaps also placing the path within a duct (LINE or HX) macro.

NOTE: This routine is documented to support interpretations of previous versions. Use MODEC=3 REDUCER connectors (Section 3.5.3.3) or EXPANDER connectors (Section 3.5.4.3) instead of FKCONE.

If the flow rate doesn‘t change direction, this routine may be placed in OPERATIONS. Otherwise, it should be placed in FLOGIC 0.

Calling Sequence:

```
CALL FKCONE(FK, FR, DI, DJ, TLEN)
```

where:

FK.....	K-factor (returned real value)
FR.....	current flow rate (for determining direction only)
DI.....	diameter at defined inlet
DJ.....	diameter at defined outlet
TLEN.....	length of the reducer or expander

Cautions: No input validations are performed. The underlying correlation* is based on high Reynolds number incompressible flows only. *Additional losses should be considered for compressible flows. This correlation tends to underpredict losses compared to other sources.*

Example:

```
 CALL FKCONE(FK12, FR12, SQRT(4.0*AFI12/pie),
+      SQRT(4.0*AFJ12/pie), TLEN12)
```

* Flow of Fluids Through Valves, Fittings, and Pipe. Crane Technical Paper No. 410, Reprinted Dec. 2001.

7.11.1.10 Inquiries

XTRACT . . . . . Return effective path suction quality
XTRACC. . . . . Return effective path species suction fraction
SUMFLO. . . . . Summed or average properties in a fluid submodel
SUMDFLO . . . Same as SUMFLO but takes into account path duplication factors
LUMPRATES . Return lump dM/dt and dU/dt (mass and energy imbalances)
REPATH . . . . . Returns the estimated Reynolds number for a path
CHKFLASH . . Verify that no lump, throat state, or pump will flash or cavitate
TOTALTP . . . . Calculates total (stagnation) temperature and pressure
STATICTPA. . . Calculates static temp. and press. in absolute (non-rotating) reference frame
VELPATH . . . . Calculates the FR-based velocity, the tangential velocity, and the total velocity for a path

Subroutine Name: XTRACT

This routine calculates the effective quality currently seen by a path in the current flow rate direction. This may not equal the quality of the upstream lump if phase suction options are active, *even if they are not active for this path*. Furthermore, for upstream junctions or STEADY tanks, the effective quality might not equal 0.0 or 1.0 even if phase suction *is* active for this path. XTRACT output corresponds to the quality printed in PTHTAB under the heading “XL UPSTRM.” See also Section 3.11.1.

Calling Sequence:

```
CALL XTRACT( XS, MODNAM, PATH)
```

where:

XS . . . . . Returned value of the effective suction quality
MODNAM . . . . . Fluid submodel name in single quotes
PATH. . . . . ID of path whose suction quality is to be evaluated

A value of -1.0 is returned if the model name and/or the path ID are invalid.

Example:

```
CALL XTRACT(xtest,'wetter',304)
```

Subroutine Name: XTRACC

This routine calculates the effective species fraction currently seen by a path in the current flow rate direction. This may not equal the species fraction of the upstream lump if phase- and/or species-specific suction options are active, *even if they are not active for this path*.

Calling Sequence:

```
CALL XTRACC( XS, MODNAM, PATH, SPEC)
```

where:

XS	Returned value of the effective species fraction
MODNAM	Fluid submodel name in single quotes
PATH	ID of path whose suction fraction is to be evaluated
SPEC	Letter identifier of species to be evaluated, in single quotes

A value of -1.0 is returned if the model name and/or the path ID are invalid. If the species is invalid, the program will abort.

Example:

```
CALL XTRACC(xtest,'wetter',304,'c')
```

Subroutine Name: LUMPRATES

Description: This routine returns the current energy and mass storage rates, dM/dt and dU/dt, for a specific lump. These numbers correspond to the last two columns of the LMPTAB routine.

Calling Sequence:

```
CALL LUMPRATES(MODNAM, lump, dmdt, dudt)
```

where:

MODNAM	fluid submodel name, input in single quotes
lump	user ID of a lump
dmdt	returned real mass storage rate, dM/dt
dudt	returned real energy storage rate, dU/dt. For variable volume tanks in a transient, this includes the work term.

Example:

```
CALL LUMPRATES('slam_valve', 450, dtest, etest)
```

Subroutine Name: SUMFLO

Description: This routine is useful for summing submodel-level data such as masses and volumes, for finding the net flow through a loop, and for calculating average properties.

Caution: Path duplication factors are not taken into account in the summations (QDOT, VOL, etc.) performed by this routine: each lump is assumed to exist once from a model-level perspective. Use SUMDFLO as an alternative. This does not affect averages (PL, TL, etc.).

Calling Sequence:

```
CALL SUMFLO(MODNAM, PARAM, RESULT)
```

where:

MODNAM . . . . . fluid submodel name, input in single quotes
PARAM . . . . . Parameter to be summed or averaged, in single quotes:
- ‘PL’ . . . . . Return average of lump pressures
- ‘TL’ . . . . . Return average of lump temperatures
- ‘TLV’ . . . . . Return average of lump vapor temperatures (zero if none)
- ‘XL’ . . . . . Return average of lump qualities
- ‘DL’ . . . . . Return average of lump densities
- ‘DF’ . . . . . Return average of lump liquid densities (zero if none)
- ‘DG’. . . . . Return average of lump vapor densities (zero if none)
- ‘AL’ . . . . . Return average of lump void fractions
- ‘QDOT’ . . . Return sum of lump heat rates
- ‘VOL’ . . . . Return sum of tank *and junction* volumes
- ‘MASS’ . . . Return sum of tank *and junction* fluid masses
- ‘ENERGY’ Return sum of tank *and junction* internal energies
- ‘VOID’ . . . Return sum of tank *and junction* void volumes
- ‘VMASS’. . Return sum of tank *and junction* vapor masses
- ‘MRATE’. . Return sum of tank *and junction* mass storage rates (dM/dt)
- ‘ERATE’ . . Return sum of tank *and junction* energy storage rates (dU/dt)

RESULT . . . . . returned value requested (may be zero)

Input mistakes result in cautions being issued, and in no results being returned.

Caution: Note that junction volumes, otherwise neglected throughout the code, are taken into account in this routine such that it can calculate the volumes, masses, etc. that would be present if the junctions were treated as tanks. Set junction volumes to zero if this effect is not desired.

Example:

```
C PLACE NET LOOP HEAT IN QTEST
      CALL SUMFLO(MODNAM, ’QDOT’, QTEST)
```

Subroutine Name: SUMDFLO

Description: This routine is useful for summing submodel-level data such as masses and volumes, for finding the net flow through a loop, and for calculating average properties. SUMDFLO is the same as SUMFLO except that it includes the effects of path duplication factors (DUPI, DUPJ) and therefore must be provided a starting point (reference lump, which is assumed to exist once relative to summations). If a closed loop exists and the path duplication factors are not balanced, an abort will occur.

Calling Sequence:

```
CALL SUMDFLO(MODNAM, LSTART, PARAM, RESULT)
```

where:

MODNAM .... fluid submodel name, input in single quotes
LSTART .... user ID of a reference lump or starting point in MODNAM
PARAM..... Parameter to be summed or averaged, in single quotes:
- 'PL'..... Return average of lump pressures
- 'TL'..... Return average of lump temperatures
- 'TLV' .... Return average of lump vapor temperatures (zero if none)
- 'XL' ..... Return average of lump qualities
- 'DL' ..... Return average of lump densities
- 'DF' ..... Return average of lump liquid densities (zero if none)
- 'DG' ..... Return average of lump vapor densities (zero if none)
- 'AL' ..... Return average of lump void fractions
- 'QDOT' .. Return sum of lump heat rates
- 'VOL' .... Return sum of tank *and junction* volumes
- 'MASS' .. Return sum of tank *and junction* fluid masses
- 'ENERGY' Return sum of tank *and junction* internal energies
- 'VOID' ... Return sum of tank *and junction* void volumes
- 'VMASS' . Return sum of tank *and junction* vapor masses
- 'MRATE' . Return sum of tank *and junction* mass storage rates (dM/dt)
- 'ERATE'.. Return sum of tank *and junction* energy storage rates (dU/dt)

RESULT .... returned value requested (may be zero)

Input mistakes result in cautions being issued, and in no results being returned.

Caution: Note that junction volumes, otherwise neglected throughout the code, are taken into account in this routine such that it can calculate the volumes, masses, etc. that would be present if the junctions were treated as tanks. Set junction volumes to zero if this effect is not desired.

Guidance: Choosing a reference lump: In general, the user should choose a lump that only occurs once in the real system, and is not duplicated from the perspective of any other lumps. For example, in the Balloon sample problem (Sample Problem Appendix), the supply tank (#1) or the junction (#2) could be a reference but the balloon tank (#3) should not. In the Radiator system model

(SYSFLUX) sample problem, lumps within the condenser (81 through 86) should not be chosen as references. Otherwise, any such nonduplicated lump will suffice as an input. Choosing a duplicated lump will not cause aborts, and will make no difference for average properties (PL, TL, etc.). However, such a selection will magnify or reduce sums (QDOT, VOL, etc.). For example, choosing tank #3 in the balloon sample problem would return a system mass, energy, etc. that are 1/3 of the actual value.

Example:

```
C PLACE NET LOOP MASS IN ETEST
      CALL SUMDFLO(MODNAM, 100, 'MASS', ETEST)
```

Subroutine Name: REPATH

Description: This routine returns an estimated Reynolds number for a specified path. For homogeneous tubes and STUBE connectors, the translatable path output variable, REY, is more convenient to use.

This routine returns about the same values as those used in PTHTAB and TWNTAB, which are useful for reference but do not take into account higher order effects such as gradients across the path (as do the internal solution techniques). The flow area (AF) and upstream thermodynamic state are used as the basis for the calculations.

Calling Sequence:

```
CALL REPATH(MODNAM, path, re)
```

where:

`MODNAM` . . . . . fluid submodel name, input in single quotes
`path` . . . . . user ID of a path
`re` . . . . . returned estimated Reynolds number for the path. If illegal inputs, including specification of a path for which no flow area (AF) has been specified, a value of -1.0 is returned

For twinned paths, each path is treated individually unless a primary twin is input *as a negative path number,* in which case an effective two-phase Reynolds number is returned.

Guidance For paths such as LOSS and valve elements that might have an AF but which lack a diameter (DH), the diameter is estimated assuming a circular cross section.

Example. Set RTEST to be the Reynolds number of path 1020 in submodel “pipes:”

```
CALL REPATH('pipes', 1020, rtest)
```

Subroutine Name: CHKFLASH

Description: In a system in which boiling or flashing is undesirable, a 9000 series fluid should be used instead of a two-phase fluid: failure should be detected and corrected, not simulated. CHKFLASH can be used to see if any 9000 series fluids (at least, those with defined saturation curves defined using "AT, PSAT, ..." as described in Section 3.21.5.2) within one or all fluid submodels are below their saturation pressure or above their saturation temperature. CHKFLASH generates warnings in the output file, and returns an integer flag of zero if there are no problems.

CHKFLASH checks both current lump states as well as path throat states. CHKFLASH will also detect net positive suction head (RNPSH, as will as maximum suction specific speed, RNSS) violations at the inlet of PUMP connectors.

CHKFLASH should normally be placed in FLOGIC 2 to detect any flashing or boiling violation of the 9000 series assumption, but can also be placed in OUTPUT CALLS for less frequent checks. For steady state solutions, FLOGIC 2 is a good choice since it is called at the end of the solution.

Calling Sequence:

```
CALL CHKFLASH(MODNAM, iflag)
```

where:

`MODNAM` .... fluid submodel name, input in single quotes, or 'ALL' for all fluid submodels

`iflag`..... returned integer flag: zero if there are no flashing or cavitation occurrences detected, otherwise it is returned as nonzero and descriptive cautions will have been written to the output file.

Example.

```
CALL CHKFLASH('pipes', itest)
if(itest .ne. 0)then
      write(nuser1,*) 'uh oh'
      timend = timen   $ stop transient and investigate
endif
```

Subroutine Name: TOTALTP

Description: This routine calculates the current total (stagnation) temperature and pressure for a particular lump/path pair.

The stagnation state is not a function of the lump state alone: lumps lack velocities, paths lack thermodynamic states. Both a lump and an out-flowing path must be specified in order to perform such a calculation.* The stagnation pressure is calculated as the current lump pressure plus the

dynamic pressure ($\rho V^2/2$) of the out-flowing path, where the velocity and density take into account flow areas (e.g., AF, AFI, AFJ) as well as phase and species suction (STAT) options. The density used in the dynamic pressure calculation may therefore differ from the density of the lump. *Two different paths flowing out of the same lump might therefore differ in their dynamic pressure.*

The returned total temperature and pressure values may reveal differences from an upstream stagnant (LSTAT=STAG) inlet state, especially in transients (due to effects such as the filling and emptying of control volumes) but even at steady state. Such differences are usually attributable to irrecoverable head losses and heat transfer along the flow path, but can also exist if flow area discontinuities exist since those cause step changes in kinetic energy. (SINDA/FLUINT produces one-time warnings at the start of each run regarding such flow area discontinuities, although these checks do not take into account branching flows.)

The total temperature is calculated based on the thermodynamic state represented by the total pressure and the total enthalpy (again respecting phase and species suction options). In other words, actual fluid properties are employed. Therefore, the total temperature calculated by TOTALTP may differ from the value predicted by the simplified formulas that are often presented in texts (e.g., that assume perfect gases).

Calling Sequence:

```
CALL TOTALTP (MODNAM, LUMPNO, PATHNO, Ttotal, Ptotal)
```

where:

`MODNAM` . . . . . fluid submodel name containing lump and path to be queried, input in single quotes such as 'LOOP'

`LUMPNO` . . . . . integer ID of the lump to be used for pressure and temperature basis. If LSTAT=STAG for this lump, Ttotal and Ptotal will be the TL and PL of this lump independent of the PATHNO input.

`PATHNO` . . . . . integer ID of an adjacent *and currently out-flowing* path to be used for velocity basis. This path must be currently extracting fluid from LUMPNO and must have a defined flow area (AF, perhaps via AFI/AFJ) or the routine will return the TL and PL of LUMPNO.
If PATHNO=0, the program will use the path that is the *current* outlet for LUMPNO, where "current" means that the designation might change if flow rates reverse. If PATHNO=0 and no such path exists, the routine will print an error message and respond with the TL and PL of LUMPNO. If PATHNO=0 and more than one such path exists, the routine will print a caution and choose one of the available paths arbitrarily.

`Ttotal` . . . . . output total temperature (real), in user TL units based on UID and ABSZRO.

`Ptotal` . . . . . output total pressure (real), in user PL units based on UID and PATMOS. The dynamic pressure may be calculated as Ptotal - PL(LUMPNO).

* A program option explained below, PATHNO=0, directs the routine to search for an appropriate path.

Examples:

```
CALL TOTALTP ('flow', 102, 122, ttest, ptest)
CALL TOTALTP ('hots', 332, 0, Ttot, Ptot)  $ Use current outlet
Pdyn = Ptot - PL332           $ Calculate dynamic pressure
```

Note: TOTALTPA, an obsoleted routine, is equivalent to TOTALTP.

Subroutine Name: STATICTPA

Description: STATICTPA is similar in calling sequence to TOTALTP (above). STATICTPA is used to estimate the static temperature and pressure at the inlet of a path with rotating and shearing walls, such as the axial flow within a journal bearing or the radial flow between a pump impeller shroud and housing.

FLUINT thermodynamic states are normally (LSTAT=NORM) static (flowing) states: the thermodynamic state of the fluid. When a path has rotation (nonzero ROTR, see Section 3.26) with relative wall motion (RVR<1), then the flow rate (FR) vector will be less than the actual fluid core velocity. This means the TL and PL of an LSTAT=NORM lump within such a flow will be somewhat higher than the static temperature, and less the total pressure (see TOTALTP above). See also VELPATH (below) for a means of calculating the velocity components.

One of the primary purposes of STATICTPA is to check for flashing or other strong fluid property changes that may be missed by SINDA/FLUINT since it uses the higher TL and PL values of the lump.

Calling Sequence:

```
CALL STATICTPA (MODNAM, LUMPNO, PATHNO, Tstat, Pstat)
```

where:

MODNAM . . . . . fluid submodel name containing lump and path to be queried, input in single quotes such as 'LOOP'

LUMPNO . . . . . integer ID of the lump to be used for pressure and temperature basis. If LSTAT=NORM for this lump, Tstat and Pstat will be the TL and PL of this lump independent of the PATHNO input if ROTR=0.0 or RVR=1.0.

PATHNO . . . . . integer ID of an adjacent *and currently out-flowing* path to be used for velocity basis. This path must be currently extracting fluid from LUMPNO and must have a defined flow area (AF, perhaps via AFI/AFJ) or the routine will return the TL and PL of LUMPNO.
If PATHNO=0, the program will use the path that is the *current* outlet for LUMPNO, where "current" means that the designation might change if flow rates reverse. If PATHNO=0 and no such path exists, the routine will print an error message and respond with the TL and PL of LUMPNO. If PATHNO=0 and more than one such path exists, the routine will print a caution and choose one of the available paths arbitrarily.

Tstat . . . . . output static temperature (real), in user TL units based on UID and ABSZRO. This static temperature is based on the total velocity including rotation.

Pstat . . . . . output static pressure (real0, in user PL units based on UID and PATMOS. This static temperature is based on the total velocity including rotation.

Example:

```
         CALL STATICTPA ('flow', 123, 123, ttest, ptest)
c check to see if the fluid is actually flashing, with a little tolerance
         stest = vts(ptest - PATMOS,flow.fi) + ABSZRO
         if(ttest .gt. stest*1.00001) then
                  write(nuser1,*) ' Liquid in bearing is flashing! '
         end if
```

Subroutine Name: VELPATH

Description: VELPATH is similar in calling sequence to TOTALTP and STATICTPA (above), but it returns three velocity components instead of temperatures and pressures. The path must have a definite flow area, otherwise zeroes are returned. It need not be a rotating path: ROTR=0.0 is legal such that VELPATH can be used with dummy (ignored) arguments in order to calculate the path velocity. The returned velocity components are as follows:

Vfr: The velocity at the path upstream end corresponding to the mass flow rate (FR) vector. This is calculated as FR/(A_{flow}*DL_{eff}) where A_{flow} is the upstream area (AF, AFI, or AFJ) and DL_{eff} is often the same as the upstream lump's DL (density) unless phase- and/or species-specific suction is active. However, if the upstream lump is stagnant (LSTAT=STAG), then Vfr=0 independent of the current FR entering the path.

Vrot: The rotary or tangential component of velocity: f*ROTR*RAD*RVR where RAD is either RADI or RADJ, and either f=π/30 for UID=SI or f=120π for UID=ENG such that the resulting velocity is in user units of either m/s or ft/hr. (Recall that ROTR always has units of revolution per minute, or RPM.)

Vtot: The total velocity in an absolute reference frame: the vector sum of Vfr and Vrot taking into account the velocity angles (VAI or VAJ). If Vrot=0.0, then Vtot = |Vfr|.

Note that the downstream end of the path might have a different set of velocity components due to changes in area, density, and rotation. Because the downstream lump's state may not be representative of the path exit state, VELPATH lists only the upstream velocity components.

Calling Sequence:

```
CALL VELPATH (MODNAM, LUMPNO, PATHNO, Vfr, Vrot, Vtot)
```

where:

`MODNAM` .... fluid submodel name containing lump and path to be queried, input in single quotes such as 'LOOP'

`LUMPNO` .... integer ID of the lump to be used for density basis. If LSTAT=STAG for this lump, the upstream velocity of the path (Vfr) is zero.

`PATHNO` .... integer ID of an adjacent *and currently out-flowing* path whose velocities are to be queried. This path must be currently extracting fluid from LUMPNO and must have a defined flow area (AF, perhaps via AFI/AFJ) or the routine will return zero Vfr and Vtot.
If PATHNO=0, the program will use the path that is the *current* outlet for LUMPNO, where "current" means that the designation might change if flow rates reverse. If PATHNO=0 and no such path exists, the routine will print an error message and respond with zero Vfr and Vtot. If PATHNO=0 and more than one such path exists, the routine will print a caution and choose one of the available paths arbitrarily.

`Vfr` ..... output velocity corresponding to FR (real), in units of either m/s or ft/hr. If LUMPNO is stagnant (LSTAT=STAG), Vfr=0.0.

`Vrot` ..... output rotary (tangential) velocity (real), in units of either m/s or ft/hr.

`Vtot` ..... output total velocity (real), in units of either m/s or ft/hr. This is the vector sum of Vfr and Vrot. If ROTR=0.0, Vtot = |Vfr|.

Example:

```
CALL VELPATH ('flow', 123, 123, ftest, rtest, vtest)
```

7.11.2 Property Subroutines

This section describes the thermodynamic and transport property routines used in FLUINT that are available for direct calls by the user. These are internal routines that have been documented so as to be accessible by the user. This distinction is important because little to no error trapping or handling logic is employed within these routines to save execution costs during normal calls by the program itself. For this reason, ***users are cautioned to read this section carefully before attempting direct calls.***

Twenty fluids commonly used in the room temperature regime (around 150 to 600 K) are available in FLUINT library. They are referred to by their ASHRAE refrigeration number, as listed in Table A-1. In addition, the user may refer to any fluid defined in an FPROP DATA block by the fluid identification number (from 6000 to 9999) selected in that block. Fluids properties are available whether or not they are used in any fluid submodels.

Both transport (viscosities and conductivities) and thermodynamic properties (pressures, temperatures, enthalpies, densities, etc.) are available. These properties are based in two *absolute* units sets, according the values of UID, ABSZRO, and PATMOS selected in the CONTROL DATA, GLOBAL block. These unit sets are listed in Table A-4. These routines agree with tabulated ASHRAE values for standard library fluids. Refer to Appendix C for the ranges of valid properties, which are also available using routines VTMIN, VTGMAX, VTLMAX, and VPGMAX.

The routines available and their arguments are listed in Table 7-7. *For user-defined fluids, some routines will not be applicable, or will not be available unless certain optional inputs have been provided.* If routines are called for which inadequate input has been provided, a warning and/or fatal error will be produced. Table 7-8 lists additional properties applicable for 6000 series COMPLIQ fluids, for which liquids are functions of both temperature and pressure.

Vapor transport properties are functions of saturation temperature only for standard library fluids, even though pressure is an input argument. This argument (pressure) is applicable for 6000 series fluids only: it is required but ignored for other fluids.

Note that *all inputs and outputs must be in absolute units*, and that the last argument is a fluid identifier. The fluid identifier must be translated; the routine call cannot be preceded by an F in column 1. There are two ways of specifying the fluid identifier, as described below:

Identifier (Last argument) . . . . . Fluid identified

`smn.FI` . . . . . Indirect method: identifies the fluid used in submodel smn where smn is the name of a valid fluid submodel. If smn contains more than one constituent, the routine PREPMC must be called beforehand to define concentrations.

`smn.FIx` . . . . . Indirect method: identifies the constituent “x” used in submodel smn where smn is the name of a valid fluid submodel and “x” is the letter identifier of a constituent in that submodel

`FIn` or `FI(n)` . . . . . Direct method: identifies fluid number n, where n is a valid number from Table A-1 or from an FPROP DATA block.

Table 7-7 Property Routines

FUNCTION	OUTPUT (returned value)	INPUT (first arguments)
VPS	Saturation P	T
VPS9	*Virtual Saturation P (9000 only)*	*T*
VTS	Saturation T	P
VSV	Vapor V	P, T
VH	Vapor H	P, T, V
VS	Vapor S	T, V
VDL	Liquid Density	T
VHFG	Heat Of Vaporization	P, T, V (vapor)
VQUALH	X (Quality)	P, H
VVISCF	Liquid Dynamic Viscosity	T
VVISCV	Vapor Dynamic Viscosity	P, T
VCPF	Liquid Heat Capacity	P, T
VHLIQ	Liquid H	P, T
VTLIQ	Liquid T	P, H
VSLIQ	Liquid S	T
VDPDT	Saturation dP/dT	P
VDPDTT	Saturation dP/dT	T
VCPV	Vapor Heat Capacity	P, T
VRGAS	Vapor Gas Constant	P,T
VQUALS	X (Quality)	P, S
VST	Liquid/Vapor Surface Tension	T
VCONDF	Liquid Thermal Conductivity	T
VCONDV	Vapor Thermal Conductivity	P, T
VALPHA	Vapor Volume Fraction	T, X
VSOS	Vapor Speed Of Sound	P, T
VDLC	Compressed Liq Density	P, T
VSOSF	Liquid Speed of Sound	P, T
VSOSFV	Two-phase Speed of Sound	P, T, X, M
VTMIN	Minimum temp., any phase	-
VTGMAX	Maximum vapor/gas temp.	-
VTLMAX	Maximum liquid temp.	-
VPGMAX	Maximum vapor/gas press.	-
VDIFV	Vapor phase diffusion volume	-
VMOLW	Molecular weight	-
VMOLWPT	Molecular weight	P, T
VMISC	Miscibility (fractional)	T

SUBROUTINE	OUTPUT (first arguments)	INPUT (next arguments)
VTAV1	Vapor T, V	P, S
VTAV2	Vapor T, V	P, H

Where:

P	Pressure (***absolute units***)
T	Temperature (***absolute units: K or R***)
X	Quality (Vapor To Total Mass Ratio)
H	Enthalpy, Intensive
S	Entropy, Intensive
V	Specific Volume
M	Two-phase sound speed method (1 or 2)

Table 7-8 Additional Property Routines, Compressible Liquids

FUNCTION	OUTPUT (returned value)	INPUT (first arguments)
VDLC	Liquid Density	P, T
VCONDFC	Liquid Thermal Conductivity	P, T
VVISCFC	Liquid Dynamic Viscosity	P, T
VSLIQC	Liquid Entropy	P, T
VULIQC	Liquid Internal Energy	P, T

Where: P Pressure (***absolute units***)
T Temperature (***absolute units: K or R***)

If the fluid within a submodel is changed during processor execution using CHGFLD (Section 7.11.1.1), then the FI array can be dynamically translated using the INTFLD routine (Section 7.11.1.6). This is the only time when an “F” should appear (and in fact must appear) in column 1 when “FI” is used as an argument on that line.

Examples:

```
 TEMP = VTS(PRES-PATMOS,LOOP.FI)   $ fluid in LOOP
 CALL VTAV2(T,V,PL33-PATMOS,HL33,FI6070) $ 6070 = water
 VDEN = 1.0/VSV(PRES-PATMOS,
+   TEMP-ABSZRO,FI717)             $ 717 = library ammonia
 VISC = VVISCF(MAIN.TL44-ABSZRO,
+    FI9014)                       $ user liquid #9014
 HFG  = VHFG(14.7-PATMOS,
+   YY.TL99-ABSZRO,1.0/VDEN,XX.FI) $ fluid in XX
```

The PRPMAP output routine, described in Section 7.11.8, may be used to print out working fluid properties. Because it is considered an output routine, input temperatures and pressures are expected in relative units.

Refer also to Section 3.21 for information regarding alternative fluid descriptions.

7.11.2.1 VSOSFV Description

The routine VSOSFV calculates speed of sound in a two-phase mixture using one of two methods: Wallis’ adiabatic method or Bursik’s metastable method. The method to be used is signaled by the third argument, M, which is an integer: 1 for Wallis’, 2 for Bursik’s method, as described in Section 3.18.1.2.

This routine is invalid for 6000 fluids unless the VSOSFV function has been input or if the VDLC *and/or* the VSOSF functions have been input. Quality may range from 0.0 to 1.0, inclusive.

When dissolved species are present, they are assumed to have no effect on the liquid speed of sound, and are assumed to stay in solution for the purposes of calculating sound speeds.

7.11.2.2 Multiple-constituent (Mixture) Properties

In single-constituent submodels, the last argument in the V* property routines (e.g., VDL for liquid density) is the fluid designator: either FInnnn where nnnn is the fluid ID, or smn.FI where smn is the submodel name. The former (direct) method is still valid for referring to fluids used as constituents in multiple-component models. However, the indirect method "smn.FI" refers to all constituents in a multiple constituent model, as will be elaborated below. To refer to constituent "A" in a submodel, use the format: smn.FIA (for example), etc. as defined above.

When referring to all constituents in a submodel using "smn.FI," the program will produce an error if the amount of each constituent is unknown. Such concentration data cannot be supplied as arguments. Therefore, before calling such a routine, the user must first indirectly specify the concentrations by specifying a lump whose current concentrations are to be used in subsequent and immediate property routine and correlation calls.

To name the lump to be used in subsequent routines, the PREPMC (prepare multiple constituent) routine is used, as described next. If no appropriate lump exists that contains the desired thermodynamic state, a plenum can be created for this purpose, with its state manipulated as needed by calls to CHGLMP or CHGLMP_MIX (Section 7.11.1.1) before any calls to PREPMC.

Subroutine Name: PREPMC

Description: This routine defines the lump to use for "subsequent and immediate" property routine calls for mixture properties in multiple-constituent fluid submodels.

"Subsequent and immediate" means that the chosen lump information will be used for subsequent calls within the same logic block and before any solution, output, restart, CHGLMP, CHGVOL, or parametric operations are performed. If the user fails to call PREPMC after such operations or break-points, but before the desired property routine call(s), the program will "forget" which lump to use and will produce an error. *PREPMC can be called as often as needed to make sure the correct and recent information is used.*

Calling Sequence:

```
CALL PREPMC(MODNAM,LUMP)
```

where:

MODNAM .... fluid submodel name containing lump, input in single quotes
LUMP ..... identifier of lump to use for concentration data

Example:

```
C     CALCULATE THE SURFACE TENSION IN MULTIPLE-CONSTITUENT LUMP 314
C     (OR AT LEAST AT THAT SAME THERMODYNAMIC STATE)
      CALL PREPMC('FRED',314)
      STEST = VST(TL314-ABSZRO,FRED.FI)
```

7.11.2.3 Henry's Coefficient

Subroutine Name: HENRY

Description: This routine returns the Henry's constant in units of the user's pressure units (Pa or psia) over unitless *mass* fraction for any binary solvent/solute system. The effects of other solvents are neglected -- use the HEN constant (Section 3.23.3) to find the actual effective constant including the effects of other solvents.

Calling Sequence:

```
CALL HENRY(P, T, idg, ids, HENOUT)
```

where:

`P` ..... *Partial* pressure, *absolute user units.*
`T` ..... Temperature, *absolute user units.*
`idg` ..... the 4 digit identifier of the solute gas: 8nnn.
`ids` ..... the identifier of the solvent (a 6000, 7000, 9000 series or library fluid).
`HENOUT` .... the returned value of the Henry's constant (Pa or psia per *mass* fraction).

Restrictions and Guidance: P and T are assumed to be in absolute units: psia and R, or Pa and K. P is partial pressure of the species designated with "idg." The output is in the unit system convention chosen by the user-input values of UID in CONTROL DATA, GLOBAL: either Pa or psia (partial pressure over *mass* fraction of solute). These units may differ from the units used in MIXTURE DATA, which are often *atmospheres* per *mole*-fraction of solute.

PREPMC must be called first in order to indicate the mixture properties.

Example:

```
 CALL PREPMC('DIS_MOD',559)
CALL HENRY( PPGA559*(PL559-PATMOS), TL559-ABSZRO,
+ 8029, 9001, HTEST)
```

7.11.2.4 Psychrometric Utilities

This section contains utilities for calculating dew points and relative humidities in lumps, or for finding the mass fraction XGx required to match a certain desired relative humidity or dew point in a lump.

These routines are not restricted to mixtures of air and water. However, they will produce warnings and no results will be returned if they are applied to mixture other than the following:

1. contains a condensible/volatile species (library, 7000, 6000 but not NEVERLIQ)
2. also contains at least one noncondensible species (8000 series gas)

Additional gases and nonvolatile liquids (9000 series liquids) may be present as well.

Subroutine Name: DEWPT

Description: This routine returns the dew point for a mixture consisting of a condensible/volatile substance plus one or more noncondensible gases. The dew point is the temperature at which liquid droplets would form in the vapor/gas phase of the current mixture (defined by identifying a lump) if it were cooled while maintaining the current pressure.

Calling Sequence:

```
CALL DEWPT('fmsn', lump, Tdew)
```

where:

`fsmn`.....fluid submodel name containing lump to be checked, in single quotes
`lump`.....identifier of the lump to be checked
`Tdew`.....the returned dew point of the lump, in user (not absolute) temperature units

Subroutine Name: HUMD2X

Description: This routine returns the vapor mass fraction (XGx) for the condensible species corresponding to the input dew point in a mixture consisting of a condensible/volatile substance plus one or more noncondensible gases.

This routine is normally used to set a lump to the prescribed dew point by passing the returned mass fraction as the XG of the condensible species (e.g., water). A lump must therefore be passed to HUMD2X to be used as a basis for temperature, pressure, and initial mixture ratio.

Caution: If more than one gas species exists, the amount of each gas is assumed to be adjusted in proportion to their current partial pressures in the lump: partial pressures (PPGx) are normalized. This is not exactly the same as what happens in a subsequent call to CHGLMP (resetting the vapor mass fraction) or exactly the reverse of DEWPT. CHGLMP normalizes on the basis of mass fraction, not partial pressure. Two or three repeated calls to HUMD2X and CHGLMP (in that order) may be required to exactly set the relative humidity of such a mixture of three or more constituents, and/or the proportion of one noncondensible gas to the rest may require additional CHGLMP calls setting those mass fractions.

Calling Sequence:

```
CALL HUMD2X('fmsn', lump, Tdew, X)
```

where:

`fsmn` ..... fluid submodel name containing lump to be used as a basis, in single quotes
`lump` ..... identifier of the lump to be used as a basis
`Tdew` ..... the input dew point temperature in user units (not absolute temperature)
`X` ..... the returned mass fraction of the condensible species to the total. *This is not the same as the specific humidity*, which is X/(1-X)

Example:

```
      CALL HUMD2X('pipes', 101, TL101-10.0, XTEST)
C if "w" is the condensible species, then X may be used
C in a subsequent call to CHGLMP. For example:
      CALL CHGLMP('pipes', 101, 'xgw', XTEST, 'pl') $ example only
```

Caution: If the input dew point is too close to the temperature of the lump, or is above it, then the temperature of the lump will rise in a subsequent call to CHGLMP to accommodate the excess condensible phase requested. In other words, *HUMD2X does not return the saturation level of the mass fraction as an upper limit.*

Subroutine Name: HUMR2X

Description: This routine returns the vapor mass fraction (XGx) for the condensible species corresponding to the input relative humidity in a mixture consisting of a condensible/volatile substance plus one or more noncondensible gases.

This routine is normally used to set a lump to the prescribed relative humidity by passing the returned mass fraction as the XG of the condensible species (e.g., water). A lump must therefore be passed to HUMR2X to be used as a basis for temperature, pressure, and initial mixture ratio.

Caution: If more than one gas species exists, the amount of each gas is assumed to be adjusted in proportion to their current partial pressures in the lump: partial pressures (PPGx) are normalized. This is not exactly the same as what happens in a subsequent call to CHGLMP (resetting the vapor mass fraction) or exactly the reverse of HUMX2R. Both of those routines normalize on the basis of mass fraction, not partial pressure. Two or three repeated calls to HUMR2X and CHGLMP (in that order) may be required to exactly set the relative humidity of such a mixture of three or more constituents, and/or the proportion of one noncondensible gas to the rest may require additional CHGLMP calls setting those mass fractions.

Calling Sequence:

```
CALL HUMR2X('fmsn', lump, R, X)
```

where:

`fsmn`.....fluid submodel name containing lump to be used as a basis, in single quotes
`lump`.....identifier of the lump to be used as a basis
`R`.....the input relative humidity (partial pressure of the vapor divided by the saturation pressure at the mixture temperature)
`X`.....the returned mass fraction of the condensible species to the total. *This is not the same as the specific humidity*, which is X/(1-X)

Example:

```
      CALL HUMR2X('pipes', 101, 0.9, XTEST)
C if "w" is the condensible species, then X may be used
C in a subsequent call to CHGLMP. For example:
      CALL CHGLMP('pipes', 101, 'xgw', XTEST, 'pl') $ example only
```

Subroutine Name: HUMX2R

Description: This routine returns the relative humidity corresponding to the input vapor mass fraction (XGx) for a condensible species in a mixture that also includes one or more noncondensible gases.

This routine is normally to check the relative humidity within a lump, and so the lump's current vapor mass fraction is a common input. Thus, a fake (illegal) value of the mass fraction will result in the lump's current state being used to calculate relative humidity. Otherwise, the input mass fraction will be used.

Caution: If more than one gas species exists, the amount of each gas is assumed to be adjusted in proportion to their current gas mass fractions in the lump: gas mass fractions (XGx) are normalized. This is what happens in a subsequent call to CHGLMP (resetting the vapor mass fraction), but is not exactly the reverse of HUMX2R, which normalizes on the basis of partial pressure instead.

Calling Sequence:

```
CALL HUMX2R('fmsn', lump, X, R)
```

where:

`fsmn` ..... fluid submodel name containing lump to be used as a basis or checked, in single quotes

`lump` ..... identifier of the lump to be used as a basis or checked

`X` ..... the input mass fraction of the condensible species to the total. *This is not the same as the specific humidity,* which is X/(1-X). If an value less than zero or more than one is used, then the current XG of the condensible species (e.g., water) within the lump is assumed.

`R` ..... the returned relative humidity (partial pressure of the vapor divided by the saturation pressure at the mixture temperature)

Example:

```
CALL HUMX2R('pipes', 101, -1.0, RelHum)
```

7.11.2.5 Simplifying Fluid Properties

In most FLUINT runs, the portion of the code that absorbs the most CPU time is the property library. The code may be significantly accelerated by substituting a limited and simplified description of a working fluid, sacrificing range for the sake of speed. Simplified fluids also facilitate debugging models by preventing unintended regimes and phases from complicating matters. For these reasons, users should consider substituting a 8000 series (noncondensible gas) or 9000 series (nonvolatile liquid) fluid description (see Section 3.21) for a more complex two-phase description (library, 6000 series, or 7000 series) if the fluid does not change phase.

Also, creating a single effective substance is computationally expedient compared to analyzing a full mixture if the concentrations are reasonably invariant. In other words, FLUINT will solve much faster with “air” as a working fluid rather than a mixture of 21% oxygen, 78% nitrogen, and 3% argon since the code can thereby neglect conservation of mass for each species.

Furthermore, 8000 and 9000 series fluids can be used more freely in mixtures. Any number (up to 26) of these fluids may be used in mixtures, whereas limits apply to two-phase (condensible/volatile) species. 8000 series fluids are also the only type allowed to dissolve into liquid phases.

The trade-off for using a simplified fluid is that it will be valid over a narrower range of temperatures and pressures, and will involve more approximations. The user can influence the valid domain to some extent by accepting a smaller or larger error relative to full description,* at least with respect to temperature variations.

Utilities exist† to generate 8000 and 9000 series fluid descriptions (reusable and unit-independent FPROP DATA blocks) from within a SINDA/FLUINT run. The source of these reduced descriptions can include:

1. A library fluid (e.g., FID=717 for ammonia)
2. A 6000 series fluid (real gas or full-range two-phase fluid)
3. A 7000 series fluid (although using the raw data inside such a description would be better)
4. Another 8000 or 9000 series fluid (although usefulness of this feature is questionable)
5. A mixture

In the last case (the mixture), the user must also specify the concentrations to be used. This specification is done indirectly by invoking the PREPMC utility (Section 7.11.2.2), which points to a lump to be used as a reference state. This lump might be a plenum within a fake fluid submodel that was created solely to arrive at the correct mixture ratios. Nonetheless, this usage of PREPMC has the important repercussion that *only valid FLUINT mixtures may be used for generating reduced FPROP DATA blocks*. Furthermore, the calculated properties will obey the somewhat simplified mixture property calculation rules employed by SINDA/FLUINT, and some properties (e.g., critical

* The reader should know that the standard library descriptions, as well as many of the available 6000 series tables, are at best 3% accurate, at worst 10%, and that various references will disagree by at least that margin.

† These utilities render obsolete the RAPPR external program.

pressure and temperature) are roughly estimated as simple mole-fraction weighted averages of the constituent species' properties. If any optional property (e.g., WSRK) is missing for one of the constituents, it will be missing or will contain an illegal value in the generated fluid property block.

Subroutine Name: PR8000

PR8000 is a utility to generate a 8000 series (perfect gas, albeit calorically imperfect) fluid description from a more complete two-phase fluid or mixture of fluids. (See Section 3.21.4.) Thus, the reasons for using it include:

1. Exploiting the speed increase from neglecting the liquid phase if condensation is not a concern;
2. Generating a single effective substance from a mixture of vapors and gases, again for computation efficiency;
3. Generating a gaseous species that can be mixed with less volatile two-phase substance (since mixtures are limited to one species that can change phase);*
4. Generating a soluble gas (since only 8000 series fluids are allowed to dissolve into the liquid phase).

PR8000 generates one complete FPROP DATA block per call. Current model units are employed (other than critical pressure, which will be in absolute pressure units regardless of PATMOS), but the resulting description can be reused in a model with differing unit choices.

The user specifies a pressure along which properties will be calculated. The minimum of this pressure and the saturation pressure (if applicable) will be used.

Mixtures may be invoked indirectly by pointing to a fluid submodel (per the last argument) containing the mixture, *and* by having called PREPMC prior to the PR8000 call to point to a lump with the desired ratio of substances in the gaseous phase. An fatal error will result if this lump lacks a gaseous or vapor phase.

This routine will usually be called only from OPERATIONS. Multiple calls may be made (perhaps at different isobars), perhaps changing the file unit number if a single file with multiple FPROP blocks is not desired.

Some properties (e.g., WSRK, DIFV) may not be available in the original fluid description. Depending on the property, the resulting description might omit the value, or list an illegal value (e.g., a negative value or "NaN"). *It is the user's responsibility to review the generated FPROP block before using it.* The user is especially urged to consider reducing the minimum and maximum temperatures (TMIN, TMAX) as needed.

* For 6000 series fluids, see also the NEVERLIQ (real gas) option. Multiple real gases can be added to mixtures.

Calling Sequence:

```
CALL PR8000(unit, Piso, Err, newID, FI)
```

where:

`unit`.....Integer file unit number; should in general be NUSER1, NUSER2, or some unit number assigned using a prior USRFIL call.

`Piso`.....Pressure to use (in user units). Properties will be generated along this isobar. If this value is too high for condensible species, saturation values will be used instead: using a large value returns all saturation values.

`Err`.....Approximate percent error between generated properties and original properties. This should be on the order of 1.0 to 10.0 (1% to 10%). Small values will generate larger AT (interpolation with temperature) subblocks.

`newID`.....Fluid ID to assign to generated FPROP block. This should be an integer between 8000 and 8999, inclusive.

`FI`.....input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
call pr8000(nuser1, 200.0, 5.0, 8717, fi717)

call prepmc('fake', 100)
call pr8000(nuser2, 1.0e30, 2.0, 8002, fake.fi)
```

Notes on Mixtures: The user is cautioned to use extra care in reviewing generated property descriptions for mixtures. Some properties (e.g. DIFV, WSRK, MOLW, PCRIT, TCRIT) are simple mole-fraction-weighted averages of the properties of the constituents, and therefore have questionable physical meaning. Therefore, the resulting fluid should not be used as part of a new mixture without reconsidering these values or the implications of their approximation.

If any property (e.g., WSRK) is missing from a constituent, it will generate either property errors or will result in illegal or absent values in the generated block.

The resulting FPROP block will be limited to the smallest overlapping range of all liquid species present in the lump pointed to by the preceding PREPMC call (Section 7.11.2.2).

Subroutine Name: PR9000

PR9000 is a utility to generate a 9000 series (nonvolatile liquid) fluid description from a more complete two-phase fluid or mixture of fluids. (See Section 3.21.5.) Thus, the reasons for using it include:

1. Exploiting the speed increase from neglecting the vapor phase if boiling is not a concern;
2. Generating a single effective substance from a mixture of liquids, again for computation efficiency;
3. Generating a liquid species that can be mixed with a more volatile two-phase substance (since mixtures are limited to one species that can change phase);
4. Generating a high pressure liquid, in which the normal use of saturation properties for specific heat (and therefore enthalpy) and density are inappropriate.

PR9000 generates one complete FPROP DATA block per call. Current model units are employed (other than critical pressure, which will be in absolute pressure units regardless of PATMOS), but the resulting description can be reused in a model with differing unit choices.

The user specifies a pressure along which properties will be calculated. The maximum of this pressure and the saturation pressure (if applicable) will be used. Since most liquid properties are pressure-independent, this choice generally has little effect. However, if the original fluid description has pressure-dependent densities and energy properties (i.e., C_p) available, these are used instead of saturation.

Mixtures may be invoked indirectly by pointing to a fluid submodel (per the last argument) containing the mixture, *and* by having called PREPMC prior to the PR9000 call to point to a lump with the desired ratio of substances in the liquid phase. An fatal error will result if this lump lacks a liquid phase. For mixtures, only saturation liquid densities are available.

This routine will usually be called only from OPERATIONS. Multiple calls may be made (perhaps at different isobars), perhaps changing the file unit number if a single file with multiple FPROP blocks is not desired.

Some properties (e.g., WSRK) may not be available in the original fluid description. Depending on the property, the resulting description might omit the value, or list an illegal value (e.g., a negative value or “NaN”). *It is the user’s responsibility to review the generated FPROP block before using it.* The user is especially urged to consider reducing the minimum and maximum temperatures (TMIN, TMAX) as needed.

Calling Sequence:

```
CALL PR9000(unit, Piso, Err, newID, FI)
```

where:

`unit`.....Integer file unit number; should in general be NUSER1, NUSER2, or some unit number assigned using a prior USRFIL call.

`Piso`.....Pressure to use (in user units). Properties will be generated along this isobar. If this value is too low for volatile species, saturation values will be used instead: using a small value or zero returns all saturation values.

`Err`.....Approximate percent error between generated properties and original properties. This should be on the order of 1.0 to 10.0 (1% to 10%). Small values will generate larger AT (interpolation with temperature) subblocks.

`newID` .....Fluid ID to assign to generated FPROP block. This should be an integer between 9000 and 9999, inclusive.

`FI` .....input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
call pr9000(nuser1, 200.0, 5.0, 9717, fi717)

call prepmc('fake', 100)
call pr9000(nuser2, 0.0, 2.0, 9002, fake.fi)
```

Notes on Mixtures: The user is cautioned to use extra care in reviewing generated property descriptions for mixtures. Some properties (e.g. WSRK, MOLW, PCRIT, TCRIT) are simple mole-fraction-weighted averages of the properties of the constituents, and therefore have questionable physical meaning. Therefore, the resulting fluid should not be used as part of a new mixture without reconsidering these values or the implications of their approximation.

If any property (e.g., WSRK) is missing from a constituent, it will generate either property errors or will result in illegal or absent values in the generated block. Other properties such as COMP are zero by default: if COMP is missing in one species will reduce the entire mixture's compressibility. Compressed liquid densities are unavailable in mixtures: saturation densities are used instead.

Liquid mixture properties are calculated using simple correlations that may or may not be appropriate. Refer to Section 3.19.1, "Mixture Property Rules," for more details.

The resulting FPROP block will be limited to the smallest overlapping range of all liquid species present in the lump pointed to by the preceding PREPMC call (Section 7.11.2.2).

7.11.3 Heat Transfer Correlation Functions and Routines

This section describes the heat transfer correlation routines used in FLUINT that are available to the advanced user. The references for these routines and their technical descriptions are found in Appendix B. See also Sample Problems C and E.

This section describes the following heat transfer routines:

STDHTC. . . . . Central call to forced convection correlation
DITTUS . . . . . Single-phase laminar and turbulent correlation
ROHSEN . . . . Condensation correlation (standard)
ACKERS . . . . Condensation correlation (alternate)
SHAH . . . . . Condensation correlation (alternate)
CHENNB . . . . Nucleate boiling correlation
HTCMIX . . . . Two-phase multiple-constituent heat correlation
HTCDIF . . . . . Degradation of condensation due to diffusion through gases

Subroutine Name: STDHTC

Description: This subroutine is the standard heat transfer correlation routine set for HTN, HTNC (Section 3.6.3.2), and HTNS ties (Section 3.6.4.2), as described in Appendix B. Given the inner wall temperature and the geometry of a pipe segment (length, flow area, and hydraulic diameter), the mass flow rate and flow quality through the segment and the thermodynamic state of the fluid (pressure, temperature, concentrations, and void fraction), STDHTC calculates the heat transfer conductance UA from the inner pipe wall to the bulk fluid. DITTUS, ROHSEN, CHENNB, HTCMIX, and HTCDIF are called as needed to calculate the heat transfer coefficient. STDHTC assumes thermodynamic and hydrodynamic equilibrium. The area used for heat exchange is based on the wetted perimeter: 4.0*TLEN*AF/DH.

If required, the heat rate may be calculated as the product of the returned UA coefficient times the difference in temperature between the node and lump: UA*(TW-TL).

UA is estimated in the case where the temperature difference is zero (TL=TW) but the flow rate is nonzero. If the flow rate is zero, the routine will use the laminar Nusselt number as a limit. If the flow rate is zero and the fluid is two-phase (and single-constituent), vapor properties will be used for heating and liquid properties for cooling.

Restrictions: All arguments are assumed in standard, *absolute* units according to the unit flag UID selected in global control data. The thermodynamic state must be within the range of fluid properties.

Restrictions: PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
CALL STDHTC (UA,FR,DH,AF,TLEN,PL,TL,XL,ALPH,TW,FI)
```

where:

UA . . . . . returned heat transfer coefficient times the wetted heat transfer area 4.0*TLEN*AF/DH.
FR . . . . . mass flow rate through the segment
DH . . . . . hydraulic diameter of the segment
AF . . . . . cross-sectional (flow) area of the segment
TLEN. . . . . length of the segment
PL . . . . . absolute pressure of the fluid
TL . . . . . absolute temperature of the fluid
XL . . . . . flow quality
ALPH. . . . . void fraction of the fluid
TW . . . . . inner wall temperature of the segment (absolute)
FI . . . . . fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where “x” is the desired fluid constituent letter identifier.*

* In Version 4.5 and earlier, this final argument was different: it was the internal sequence number of the fluid submodel, as returned by MFLTRN. The code attempts to autodetect such outdated usage. This note is intended as an aid for users who are updating an old model.

Function Name: DITTUS

Description: This function returns the heat transfer coefficient for single phase flow based on the Dittus-Boelter correlation for turbulent flow or the isothermal circular Nusselt number (3.66) for laminar flow. Hausen's transition correlation is used for Reynolds numbers between approximately 2000 and 6400. This is the condensation correlation used in HTN, HTNC, and HTNS ties.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
H = DITTUS (FR,DH,AF,TW,PL,TL,XL,FI)
```

where:

H	heat transfer coefficient
FR	mass flow rate through the segment (positive)
DH	hydraulic diameter of the segment
AF	cross-sectional (flow) area of the segment
TW	inner wall temperature (compared with TL only, magnitude is not important)
PL	absolute pressure of the fluid
TL	absolute temperature of the fluid
XL	quality or phase flag, 1.0 for vapor, 0.0 for liquid
FI	fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where "x" is the desired fluid constituent letter identifier

Function Name: ROHSEN

Description: This function returns the condensation heat transfer coefficient for two-phase flow based on Rohsenow's correlation (see Appendix B). This is the condensation correlation used in HTN, HTNC, and HTNS ties.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine. TW must be less than TL. XL and ALPH must be between 0.0 and 1.0, exclusive. FR must be greater than zero.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
H = ROHSEN (FR,DH,AF,TW,PL,TL,XL,ALPH,FI)
```

where:

H	heat transfer coefficient
FR	mass flow rate through the segment (positive)
DH	hydraulic diameter of the segment
AF	cross-sectional (flow) area of the segment
TW	inner wall temperature (must be less than TL)
PL	absolute pressure of the fluid
TL	absolute temperature of the fluid
XL	flow quality: greater than zero, less than one
ALPH	void fraction: greater than zero, less than one
FI	fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where "x" is the desired fluid constituent letter identifier

Function Name: ACKERS

Description: This alternate function returns the condensation heat transfer coefficient for two-phase flow based on Acker's correlation (see American Society of Heating, Refrigeration, and Air Conditioning Engineers (ASHRAE) Fundamentals Handbook).

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine. TW must be less than TL. FR and XL must be greater than zero.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
H = ACKERS (FR,DH,AF,TW,PL,TL,XL,FI)
```

where:

H	heat transfer coefficient
FR	mass flow rate through the segment (positive)
DH	hydraulic diameter of the segment
AF	cross-sectional (flow) area of the segment
TW	inner wall temperature (must be less than TL)
PL	absolute pressure of the fluid
TL	absolute temperature of the fluid
XL	flow quality: greater than zero, less than one
FI	fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where "x" is the desired fluid constituent letter identifier

Function Name: SHAH

Description: This alternate function returns the condensation heat transfer coefficient for two-phase flow based on Shah's correlation. This highly empirical correlation is extremely simplistic, but is favored by some analysts.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine. TW must be less than TL. FR and XL must be greater than zero. The critical pressure is used within this correlation.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
H = SHAH (FR,DH,AF,TW,PL,TL,XL,FI)
```

where:

H	heat transfer coefficient
FR	mass flow rate through the segment (positive)
DH	hydraulic diameter of the segment
AF	cross-sectional (flow) area of the segment
TW	inner wall temperature (must be less than TL)
PL	absolute pressure of the fluid
TL	absolute temperature of the fluid
XL	flow quality: greater than zero, less than one
FI	fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where "x" is the desired fluid constituent letter identifier

Function Name: CHENNB

Description: This function returns the nucleate boiling heat transfer coefficient for two-phase flow based on Chen's correlation (see Appendix B). This is the condensation correlation used in HTN, HTNC, and HTNS ties.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine. TW must be greater than TL. XL and ALPH must be less than 1.0, exclusive. FR must be greater than zero.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
H = CHENNB (FR,DH,AF,TW,PL,TL,XL,ALPH,FI)
```

where:

H	heat transfer coefficient
FR	mass flow rate through the segment (positive)
DH	hydraulic diameter of the segment
AF	cross-sectional (flow) area of the segment
TW	inner wall temperature (must be greater than TL)
PL	absolute pressure of the fluid
TL	absolute temperature of the fluid
XL	flow quality: less than one
ALPH	void fraction: less than one
FI	fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where "x" is the desired fluid constituent letter identifier

Function Name: HTCMIX

Description: This function returns the two-phase multiple-constituent heat transfer coefficient assuming negligible phase change (see Appendix B). This is the condensation correlation used in HTN, HTNC, and HTNS ties.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data.TW must be greater than TL. XL and ALPH must be less than 1.0, exclusive. FR must be greater than zero.

Restrictions: FI must be of the form smn.FI where smn refers to a two-phase multiple constituent fluid submodel. FI must be translated: do *not* place an F in column 1 when calling this routine.

Restrictions: PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations.

Calling Sequence:

```
H = HTCMIX (FR,DH,AF,TW,PL,TL,XL,ALPH,FI,METH)
```

where:

H. . . . . heat transfer coefficient
FR . . . . . mass flow rate through the segment (positive)
DH . . . . . hydraulic diameter of the segment
AF . . . . . cross-sectional (flow) area of the segment
TW . . . . . inner wall temperature (must be greater than TL)
PL . . . . . absolute pressure of the fluid
TL . . . . . absolute temperature of the fluid
XL . . . . . flow quality: less than one
ALPH. . . . . void fraction: less than one
FI . . . . . fluid identifier: smn.FI where smn is the desired fluid submodel name. smn must be a two-phase multiple-constituent submodel
METH. . . . . Integer method (regime) identifier:
1 - effective fluid (bubbly)
2 - options 1 and 3 weighted by void fraction (slug)
3 - phasic resistances in series (annular)
4 - phasic resistances in parallel (stratified)

Example:

```
      CALL PREPMC('GILL',3030)
      HTEST = HTCMIX(FR10,DH10,AF10,NODE.T2020-ABSZRO,PL3030
     +   ,TL3030,XL3030,AL3030,GILL.FI,1)
```

Function Name: HTCDIF

Description: This function corrects a condensation heat transfer coefficient for two-phase flow based on the effects of diffusion through a noncondensible gas (see Appendix B.5). This is the correlation used in HTN, HTNC, and HTNS ties.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine. TW must be less than TL. XL and ALPH must be between 0.0 and 1.0, exclusive. FR must be greater than zero.

Restrictions: PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations.

Calling Sequence:

```
HEFF = HTCDIF (HCON,FR,DH,AF,TW,PL,TL,XL,ALPH,FI)
```

where:

HEFF . . . . . heat transfer coefficient, corrected (W/m^2-K or BTU/hr-ft^2-F)
HCON . . . . . heat transfer coefficient, uncorrected (perhaps calculated via a previous call to ROHSEN or SHAH, for example) (W/m^2-K or BTU/hr-ft^2-F)
FR. . . . . mass flow rate through the segment (positive)
DH. . . . . hydraulic diameter of the segment
AF. . . . . cross-sectional (flow) area of the segment
TW. . . . . inner wall temperature (must be less than TL)
PL. . . . . absolute pressure of the fluid
TL. . . . . absolute temperature of the fluid
XL. . . . . flow quality: greater than zero, less than one
ALPH . . . . . void fraction: greater than zero, less than one
FI. . . . . fluid identifier: smn.FI where smn is the desired fluid submodel name. The submodel must contain a mixture that includes a condensible species.

HTCDIF iteratively calculates an interfacial temperature (T_i) and mass transfer rate per the methods outlined in Appendix B.5, where h_{film} in the Appendix is the same as "HCON" above, T_{bulk} is "TL" and h_v is calculated internally using a call to the DITTUS function. The interface temperature is restricted to the range $TW < T_i < TL$.

Example:

```
      CALL PREPMC('MYMOD',203)
C RAW (PURE SUBSTANCE) CONDENSATION COEFFICIENT:
      HTEST = ROHSEN(FR10,DH10,AF10,NODE.T22-ABSZRO,PL203-PATMOS
     +   ,TL203-ABSZRO,XL203,AL203,MYMOD.FI)
C DEGRADED COEFFICIENT TAKING INTO ACCOUNT DIFFUSION BARRIER
      CTEST = HTCDIF(HTEST,FR10,DH10,AF10,NODE.T22-ABSZRO,PL203-PATMOS
     +   ,TL203-ABSZRO,XL203,AL203,MYMOD.FI)
```

Subroutine Name: HTUDIF

Description: This subroutine corrects a condensation heat transfer coefficient for two-phase heat transfer based on the effects of diffusion through a noncondensible gas (see Appendix B.5). This correction is applied automatically for HTP ties during condensation mode, but is also available for direct user calls. The wall temperature (TW) should be less than the bulk fluid temperature, and the dew point should be above the wall temperature. This correction is intended for FLUINT ties. For a more generalized liquid/vapor interface solution that can include vaporization (e.g., evaporative cooling) or drying, and which can apply to FTIEs, see HTFDIF below.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine.

If TL is greater than TW, but TW is hotter than the dew point, or if T=TW, then HTUDIF returns HEFF=HVAP.

If the freestream flow is very dry (low relative humidity), the results will be questionable since vaporization may occur instead of condensation, yet there is no supply of liquid. In that case HTFDIF methods should be used instead (see below).

Restrictions: PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations.

Calling Sequence:

```
CALL HTUDIF (HEFF,HFILM,HVAP,TW,PL,TL,FI)
```

where:

HEFF	returned heat transfer coefficient, corrected (W/m^2-K or BTU/hr-ft^2-F)
HFILM.....	input condensation or vaporization heat transfer coefficient with no gas (i.e., uncorrected coefficient, W/m^2-K or BTU/hr-ft^2-F). This coefficient is often based on conduction through a liquid film.
HVAP	input vapor/gas (single-phase) heat transfer coefficient (interface to gas bulk, W/m^2-K or BTU/hr-ft^2-F)
TW.....	absolute wall temperature (degrees R if UID=ENG, else degrees K).
PL.....	absolute pressure of the fluid (psia if UID=ENG, else Pa)
TL.....	absolute temperature of the bulk fluid (degrees R if UID=ENG, else degrees K).
FI.....	fluid identifier: smn.FI where smn is the desired fluid submodel name. The submodel must contain a mixture that includes a condensible species.

HTUDIF iteratively calculates an interfacial temperature (T_i) and mass transfer rate per the methods outlined in Appendix B.5, where h_{film} in the Appendix is the same as "HFILM" above, T_{bulk} is "TL" and h_v is "HVAP." The interface temperature is restricted to the range TW < T_i < TL. This routine is essentially the same as HTCDIF except that HVAP is provided by the user rather than based on convection, so it is appropriate for HTU or HTUS ties.

Guidance: HEFF, HFILM, and HVAP can also use units of conductance (W/K or BTU/hr-F, with the area already multiplied in), as long as all three parameters use the same units.

Example:

```
C application to an HTU tie:
C
      CALL PREPMC('MYMOD',203)
C DEGRADED COEFFICIENT TAKING INTO ACCOUNT DIFFUSION BARRIER
      CALL HTUDIF(UB2203, HFILM, HVAP, NODE.T22-ABSZRO, PL203-PATMOS
     +   ,TL203-ABSZRO, MYMOD.FI)
```

Subroutine Name: HTFDIF

Description: This subroutine solves for the liquid/vapor interface condition (temperature and vaporization or condensation mass flow rate), including the effects of diffusion through a noncondensible gas (see Appendix B.5). It is a generalization of the HTCDIF and HTUDIF corrections, and unlike those solutions, HTFDIF is applicable to drying or vaporization* in the presence of freestream vapor/gas mixtures low dew points. HTCDIF and HTUDIF apply to fluid-to-wall heat transfer, and walls can only accumulate liquid during condensation: they cannot supply liquid during

* *Boiling* at a wall does not require a diffusion degradation calculation for diffusion since the vapor is propelled into the gas phase. This routine is therefore only applicable to *vaporization* off of a liquid surface to a subsaturated gas. (It may also be used for condensation.)

vaporization. HTFDIF applies instead to lump-to-lump heat and mass transfer, where one lump is assumed to contain some liquid at temperature TF, and another is assumed to have vapor and/or gas at temperature TL. HTCDIF and HTUDIF return “corrected” heat transfer coefficients that fold in the effects of mass transfer, whereas HTFDIF returns the interface condition itself, including the temperature and mass transfer rate.

This calculation is applied automatically for the heat and mass transfer at the main interface of twinned tanks when both a condensible and noncondensible gas are present, as needed to define the QF of the CONSTQ FTIE and the mass transfer rate of vaporization or condensation at the interface (which is applied to the TSPECx subpath).

HTFDIF is also available for direct user calls perhaps as needed in the modeling the drying of a liquid film, evaporative cooling or humidification of the freestream gas, or the build-up of liquid due to condensation. For a user call, the temperature of TF need not correspond to a real lump, and in fact TF can represent a superheated state (e.g., heating through a thin liquid film, or a liquid that is constrained from boiling by a porous structure such as a wick). However, if TF is warmer than the maximum liquid temperature allowed (usually T_{crit} for the condensible species), a zero mass transfer solution will be returned (EME=HFG=0.0).

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FI must be translated; do *not* place an F in column 1 when calling this routine.

The sign of (TF-TL) influences but does not determine whether vaporization or condensation results. Vaporization can result, for example, if the liquid is exposed to hot but dry (low dew point) vapor.

Restrictions: PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in the vapor (freestream) side corresponding to PL and TL.

Guidance: A vaporizing solution (EME>0) can use liquid injection from perhaps an MFRSET, but a condensing solution (EME<0) may require species specific suction in an MFRSET, which restricts the solution to one using tanks and avoiding STEADY (FASTIC) solutions.

Calling Sequence:

```
CALL HTFDIF (EME,TI,HFG,HFILM,HVAP,TF,PL,TL,FI)
```

where:

EME . . . . . returned mass transfer rate per unit area (flux), positive for evaporation (note this is the opposite sign convention for the m” parameter in Appendix B.5: m” = -EME). Units are kg/s-m^2 or lb_m/hr-ft^2

TI . . . . . returned *absolute* liquid/vapor interface temperature (degrees R if UID=ENG, else degrees K).

HFG . . . . . returned heat of vaporization h_{fg} at TI (J/kg or BTU/hr-lb_m). This value may be zero if no mass transfer is possible.

HFILM..... input condensation or vaporization heat transfer coefficient with no gas (i.e., uncorrected coefficient, W/m^2-K or BTU/hr-ft^2-F). This coefficient represents the heat transfer from the wall or bulk liquid to the interface, and is often based on conduction through a liquid film.

HVAP ..... input vapor/gas (single-phase) heat transfer coefficient (interface to gas bulk, W/m^2-K or BTU/hr-ft^2-F)

TF..... absolute liquid bulk (or wall) temperature (degrees R if UID=ENG, else degrees K).

PL..... absolute pressure of the bulk or freestream fluid (psia if UID=ENG, else Pa)

TL..... absolute temperature of the bulk or freestream fluid (degrees R if UID=ENG, else degrees K).

FI..... fluid identifier: smn.FI where smn is the desired fluid submodel name. The submodel must contain a mixture that includes a condensible species.

HTFDIF iteratively calculates an interfacial temperature (T_i) and mass transfer rate per the methods outlined in Appendix B.5, where h_{film} in the Appendix is the same as "HFILM" above, T_{bulk} is "TL" and h_v is "HVAP," and the condensation mass flux m" is the same as the negative of EME

Guidance: HFILM, and HVAP can also use units of conductance (W/K or BTU/hr-F, with the area already multiplied in), as long as both parameters use the same units. In this case, EME will contain the mass flow rate (kg/s or lb_m/hr) rather than the mass flux (mass flow rate per unit area).

Example:

```
C Liquid in lump 11 is vaporized off its surface by hot dry gas flowing
C past it in lump 33. Path 1112 is an MFRSET from 11 to 12 with STAT=LRV
C (plus appropriate species suction to only pull the condensible gas
C during condensation, if applicable). The surface are is SurfArea.
C QL terms are adjusted below, but a CONSTQ FTIE is an alternative.
C
      CALL PREPMC('FILM_DRY',33)
      CALL HTFDIF(ETEST, TTEST, HTEST, Hfilm, Hgas
     +   ,TL11-ABSZRO, PL33-PATMOS, TL33-ABSZRO, FILM_DRY.FI)

C if Hfilm and Hgas ("HVAP" above) have conductance units (e.g., W/K) then:
      smfr1112 = ETEST
      if(etest .gt. 0.0)then   $ evaporating
            QL11 = Hfilm*(TTEST - (TL11-ABSZRO))
      else                     $ condensing
            QL11 = Hgas*((TL33-ABSZRO) - TTEST)
      endif
      QL33 = -QL11
```

```
C else if Hfilm and Hgas have film coefficient units (e.g., W/m2-K) then:
      smfr1112 = ETEST*SurfArea
      if(etest .gt. 0.0)then   $ evaporating
            QL11 = Hfilm*SurfArea*(TTEST - (TL11-ABSZRO))
      else                     $ condensing
            QL11 = Hgas*SurfArea*((TL33-ABSZRO) - TTEST)
      endif
      QL33 = -QL11
```

7.11.4 Auxiliary Heat Transfer Correlations

This section documents user-callable correlations helpful for modeling specific conditions. These correlations and corrections are not used by SINDA/FLUINT unless specifically invoked by the user.

7.11.4.1 Pool Boiling and Limits

Subroutine Name: POOLBOIL

Description: This routine returns an estimate of the effective heat transfer coefficient for nucleate pool boiling. It is the basis for HTP ties (Section 3.6.3.3), which also cover post-CHF and non-boiling modes.

If the current heat flux is input, the wall temperature corresponding to that flux is returned as well. Otherwise, the current wall temperature is used as an input (and not overwritten) if the input heat flux is zero.

This routine also returns rough estimates of the range within which a nucleate boiling assumption is valid. This range information includes the maximum heat flux (also called critical heat flux, CHF, the maximum flux for pure nucleate boiling), the maximum wall temperature for pure nucleate boiling, the minimum heat flux for pure film boiling, and the Leidenfrost temperature (the minimum temperature for pure film boiling). Unlike an HTP tie, however, this routine does not simulate those regimes.

The heat transfer coefficient is estimated using Rohsenow's correlation* for pool boiling:

$$U = \mu_l \left(\frac{\Delta T}{h_{fg}}\right)^2 \left(\frac{g(\rho_l - \rho_v)}{\sigma}\right)^{\frac{1}{2}} \left[\frac{C_{p,\,l}}{Pr_l^S \cdot C_{sf}}\right]^3$$

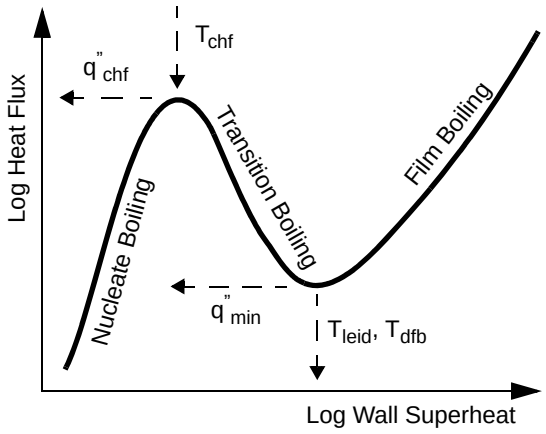
where S and C_{sf} are empirical corrections that depend on the fluid and the surface (see below).

Rohsenow's pool boiling correlation predicts that the heat flux is proportional to the temperature difference cubed, and therefore the effective heat transfer coefficient U is proportional to the current temperature difference squared, as shown in the above equation. In other words, the behavior is highly nonlinear: a clear violation of the assumptions of Newton's Law of Cooling upon which the concept of a heat transfer coefficient is based. Updating the UA of an HTU tie according to the returned U of this routine, as called perhaps in FLOGIC 0, will cause numerical stability problems: nonconvergence in steady states and small time steps and/or noisy UA predictions in transients.

* W.M. Rohsenow, "A Method of Correlating Heat-Transfer Data for Surface Boiling Liquids," *Transactions of ASME,* vol. 74, pp. 969-975, 1952.

One alternative in steady-states is to provide the heat flux, if known, and have the routine return the wall temperature. A better alternative that is also appropriate in transients is to use the returned UB and UEDT values directly in the tie (specifying the heat transfer area, AHT, separately). For this routine, UEDT = 2.0 and UB = $U/\Delta T^2$.

The assumption of pure nucleate boiling cannot be made above the critical heat flux nor above the corresponding critical wall temperature. Above that point, transition boiling will exist with degraded heat transfer. At high enough wall temperatures (the Leidenfrost point), all nucleate boiling will cease and only film boiling will remain. Because of the low heat transfer coefficients associated with film boiling, the heat flux at the Leidenfrost point is substantially lower than the critical heat flux even though the wall temperature is substantially higher, and is therefore often called the "minimum heat flux" (meaning minimum for pure *film* boiling). For a given heat flux between the minimum and maximum points, multiple wall temperature solutions exist as depicted above at the right.



In addition to returning heat transfer coefficients in the nucleate boiling regime, POOLBOIL returns information about the above limits. The critical heat flux and Leidenfrost heat flux are calculated using Zuber's estimates (Section 3.6.7).

$$q''_{CHF} = \frac{\pi}{24}\rho_v h_{fg}\left[\frac{\sigma g(\rho_l-\rho_v)}{\rho_v^2}\right]^{\frac{1}{4}}\left(\frac{\rho_l+\rho_v}{\rho_l}\right)^{\frac{1}{2}}$$

The Leidenfrost temperature is calculated according to the formula proposed by Berenson, as listed in Van P. Carey (p. 316, Hemisphere, 1992):

$$T_{leid} = 0.127\left(\frac{\rho_v h_{fg}}{k_v}\right)\left[\frac{g(\rho_l-\rho_v)}{(\rho_l+\rho_v)}\right]^{\frac{2}{3}}\left(\frac{\sigma}{g(\rho_l-\rho_v)}\right)^{\frac{1}{2}}\left(\frac{\mu_v}{g(\rho_l-\rho_v)}\right)^{\frac{1}{3}} + T_{sat}$$

This estimate often yields excessive temperatures (e.g., above the critical point), *and should not be used without considering the upper limit of the departure from film boiling* (T_{dfb}) as described in Section 3.6.7:

$$T_{dfb} = 0.97 \cdot T_{crit}\left(0.932 + 0.077\left(\frac{T_{sat}}{T_{crit}}\right)^9\right) + 0.03 \cdot T_{sat}$$

Keeping in mind the rough nature of all estimations related to boiling, if the CHF is exceeded (i.e., if U*(T-TL) > Q_{chf} or T > T_{chf}), pure nucleate boiling cannot be sustained. Past the CHF limit, the heat transfer coefficient will degrade rapidly through the transition boiling regime. If the surface is hotter than the Leidenfrost temperature, no nucleate boiling will exist at all, only film boiling. The surface temperature must be below the Leidenfrost temperature (T < T_{leid}) before *any* nucleate boiling can occur. *Neither transitional nor film boiling heat transfer coefficients are provided by POOLBOIL. See HTP ties instead (Section 3.6.3.3).* Rather, the returned heat fluxes and Leidenfrost temperature are simply made available so that the user can check the limits of appropriateness for the other returned results, which assume pure nucleate boiling.

The presumed units are dictated by UID, ABSZRO, and PATMOS in CONTROL DATA, GLOBAL. This routine should be called in FLOGIC 0.

Calling Sequence:

```
      CALL POOLBOIL(U,UB,UEDT,QCHF,TCHF,QMIN,TLEID
     +             ,T,TL,CSF,S,QF,FI)
```

where:

`U` . . . . . returned heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

`UB` . . . . . returned tie UB parameter

`UEDT` . . . . . returned tie UEDT parameter

`QCHF` . . . . . returned maximum (critical) flux (user units: W/m^2 or BTU/hr-ft^2). Above this flux, pure nucleate boiling cannot exist: the heat transfer coefficients predicted by this routine cannot be used without modification.

`TCHF` . . . . . returned maximum (critical) temperature (user units: R, F, C, or K). Above this wall temperature, pure nucleate boiling cannot exit: the heat transfer coefficients predicted by this routine cannot be used without modification. Returned temperatures above the thermodynamic critical point (see above) should be disregarded.

`QMIN` . . . . . returned minimum flux for film boiling (user units: W/m^2 or BTU/hr-ft^2). Below this flux, only nucleate boiling can exist.

`TLEID` . . . . . returned Leidenfrost temperature (user units: R, F, C, or K). Above this wall temperature, only film boiling can exist and therefore the heat transfer coefficients predicted by this routine cannot be used at all. The resulting temperature may be high compared to the actual, or compared to other criteria for limits on film boiling (see text above). Returned temperatures above the thermodynamic critical point (see above) should be disregarded.

`T` . . . . . input temperature (in user units) of the wall (nonpositive QF)
returned temperature (in user units) of the wall (positive QF)
If input, this value must be below the thermodynamic critical point.

`TL` . . . . . input saturation temperature (in user units) of the fluid. Fluid properties will be evaluated at this temperature, including saturation pressure.

CSF . . . . . input surface-fluid coefficient (see C_{sf} in Table 15.1 of the Handbook of Heat Transfer, 3rd Ed., McGraw Hill, for example). If nonpositive, an internal value of 0.01 will be used.

S. . . . . input Prandtl number exponent. 1.0 for water, 1.7 for other fluids. If nonpositive, an internal value of 1.7 will be used.

QF . . . . . optionally input heat flux (user units: W/m^2 or $BTU/hr\text{-}ft^2$). If positive, this will be used to set the wall temperature, T, accordingly. U, UB, and UEDT will also be set in this mode.

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier. The chosen fluid must be two-phase (i.e., not 8000 series, 9000 series, nor 6000 NEVERLIQ).

Caution: If QF is positive, T will be overwritten and should be a variable (e.g., not a constant or arithmetic expression).

Caution: If T is less than or equal to TL, the returned heat transfer coefficient will be zero.

Examples:

```
C the following calculates the heat transfer coefficient, HTEST, and
C uses it to set the conductance of a FLUINT tie. Node and lump temperature
C are used to specify the wall and fluid, and QF=0.0 so STEEL.T100 will
C not be overwritten. QTEST will contain the CHF estimate, and it will be
C used to determine the appropriateness of the resulting values.
C STEEL.T100 could similarly have been compared with TCHF, depending on
C which was more stable in a given model, temperature or flux.
C
       CALL POOLBOIL(HTEST, BTEST, ETEST, QTEST, TCHF, RTEST, TTEST
      +           ,STEEL.T100, WET.TL100, 0.013, 1.7, 0.0, WET.FI)
       IF(HTEST*(STEEL.T100-WET.TL100) .LT. QTEST)THEN
            WET.UA102   = HTEST*HAREA
       ELSE
            ... transition or film boiling


C instead of setting UA, it is better to set UB and UEDT for the tie,
C assuming AHT=HAREA ...
C
       CALL POOLBOIL(HTEST, UB102, UEDT102, QTEST, TMAX, RTEST, TTEST
      +           ,STEEL.T100,WET.TL100, 0.013, 1.7, 0.0, WET.FI)
```

```
C the following calculates the heat transfer coefficient, UCOEF, and
C the wall temperature, TABS, based on the input value of heat flux,
C QTEST. The returned value of TWALL is used to set the temperature of
C a SINDA node. QCRIT will contain the CHF estimate. Note that values
C of C and S have been defaulted.
C
       QTEST      = 150.0
       CALL POOLBOIL(UCOEF, UB66, UEDT66, QCRIT, TCRIT, QMIN, TLEID,
      +            TWALL, TFLOW, 0.0, 0.0, QTEST, FI717)
       WALL.T301 = TWALL
```

7.11.4.2 External Flow Heat Transfer Correlations

This section documents heat transfer correlations available for external flow past an object (a plate, a cylinder, etc.) Low speed, incompressible flow (Mach < 0.4 and low Eckert number: Ec = $V^2/C_p\Delta T$) is assumed. The available routines are:

EXTPLATE . . . External flow past a flat plate
EXTCYL . . . . . External flow past a cylinder
EXTSPH . . . . . External flow past a sphere
EXTOBJ . . . . . External flow past an object (user-supplied coefficients)
EXTUBANK . . Tube banks and pin fin heat sinks

Subroutine Name: EXTPLATE

Description: This routine returns an estimate of the average heat transfer coefficient over a flat plate parallel to the flow. It may be used for the whole plate (D1=0.0) or for a section of the plate from distance D1 to D2 from the leading edge.

The source for the underlying correlations is Heat Transfer, L. Thomas, Prentice-Hall, 1993, Table 8-9. Uniform wall temperature is assumed. Prandtl numbers should be greater than 0.6 for turbulent flow, but limits are not enforced in the interest of producing first-order estimations lacking more appropriate correlations.

Calling Sequence:

```
CALL EXTPLATE(U,P,T,X,TW,VEL,D1,D2,FI)
```

where:

U. . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F) for the section from D1 to D2 (distance from leading edge, see below).
P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.
T. . . . . input freestream fluid temperature (user units: R, F, C, or K).
X. . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
TW . . . . . input surface temperature (user units: R, F, C, or K).
VEL . . . . . input freestream velocity (user units: m/s or ft/hr ... *not* ft/s)
D1 . . . . . input distance from edge, start of segment (user units: m or ft). May be zero.
D2 . . . . . input distance from edge, end of segment (user units: m or ft). D2 > D1

FI..... input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
CALL EXTPLATE(UTEST, 0.0, T100, 1.0, T200, Velocity, 0.01,0.02,
.       fi8729)
G100200 = UTEST * 0.01*length   $ Apply as conductor
```

Subroutine Name: EXTCYL

Description: This routine returns an estimate of the average heat transfer coefficient over a semi-infinite circular cylinder subject to perpendicular external flow. The source of this correlation is the Handbook of Applied Thermal Design, E.C. Guyer (ed.), McGraw-Hill, 1989, Section 4.9.

For very low Pr (less than 0.03, which is approximately true of liquid metals), a separate correlation is invoked that is valid for high Reynolds numbers (100 to 2.0e7). Otherwise, the default correlation is valid for normal gases and liquids, and Reynolds number limitations are not available.

Calling Sequence:

```
CALL EXTCYL(U,P,T,X,TW,VEL,D,FI)
```

where:

U..... returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

P..... input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T..... input freestream fluid temperature (user units: R, F, C, or K).

X..... input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW..... input surface temperature (user units: R, F, C, or K).

VEL ..... input freestream velocity (user units: m/s or ft/hr ... *not* ft/s)

D..... input diameter (user units: m or ft)

FI..... input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
CALL EXTCYL(UTEST, 0.0, T100, 1.0, T200, Velocity, DIAM,
.       fi8729)
G100200 = UTEST * pi *diam* length   $ Apply as conductor
```

Subroutine Name: EXTSPH

Description: This routine returns an estimate of the average heat transfer coefficient over a sphere subject to external flow. The source of this correlation is the Handbook of Applied Thermal Design, E.C. Guyer (ed.), McGraw-Hill, 1989, Section 4.9.

The limits of applicability are as follows:

$3.5 < Re < 1.0e5$ . . . . . Reynolds number
$0.7 < Pr < 380$ . . . . . fluid Prandtl number*
$1 < \mu/\mu_w < 3.2$ . . . . . freestream to wall viscosity

Calling Sequence:

```
CALL EXTSPH(U,P,T,X,TW,VEL,D,FI)
```

where:

U. . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)
P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.
T. . . . . input freestream fluid temperature (user units: R, F, C, or K).
X. . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
TW . . . . . input surface temperature (user units: R, F, C, or K).
VEL . . . . . input freestream velocity (user units: m/s or ft/hr ... *not* ft/s)
D. . . . . input diameter (user units: m or ft)
FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

* For very low Pr (less than 0.03, which is approximately true of liquid metals), a separate correlation is invoked that is valid for high Reynolds numbers (4.0e4 to 2.0e5).

Example:

```
CALL EXTSPH(UTEST, 0.0, T100, 1.0, T200, Velocity, DIAM,
.       fi8729)
G100200 = UTEST * 4.0*pi*(diam/2.)**2   $ Apply as conductor
```

Subroutine Name: EXTOBJ

Description: This routine returns an estimate of the average heat transfer coefficient for a two-dimensional (semi-infinite) object subject to external flow. The description of the object is provide by the user via a characteristic dimension plus coefficients and exponents in a generalized Nusselt number equation:

$$U = k*Nu/D = k/D *B*Re^{REX}*Pr^{PEX}$$

where U is the returned heat transfer coefficient, k is the thermal conductivity, Nu is the Nusselt number (based on the provided characteristic length D), Re is the Reynolds number and Pr is the Prandtl number. B, REX, and PEX are user-supplied coefficients based on the shape of the object. All properties are evaluated at the average (film) temperature.

Usually, PEX is 1/3. For a flat plate facing (perpendicular to) the flow, B=0.228 and REX=0.731 and D is the width of the plate (the length is infinite). For a square (with one side of length D facing the flow), B=0.177 and REX=0.699 for low Reynolds numbers (Re < 8000) and for high Reynolds numbers (Re > 5000*), B=0.102 and REX=0.675. See for other example references such as Table 4.11 in Handbook of Applied Thermal Design, E.C. Guyer (ed.), McGraw-Hill, 1989.

These correlations are usually only valid for gases. However, this restriction is not enforced and therefore quality X is a required argument.

Calling Sequence:

```
CALL EXTOBJ(U,P,T,X,TW,VEL,D,B,REX,PEX,FI)
```

where:

U . . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

* These ranges overlap between a Reynolds number of 5000 and 8000. To avoid introducing a discontinuity, an interpolation is recommended for the input values of B and REX within this ambiguous range.

X. . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
TW . . . . . input surface temperature (user units: R, F, C, or K).
VEL . . . . . input freestream velocity (user units: m/s or ft/hr ... *not* ft/s)
D. . . . . input characteristic diameter (user units: m or ft)
B. . . . . input Nusselt number coefficient
REX . . . . . input Reynolds number coefficient
PEX . . . . . input Prandtl number coefficient
FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
CALL EXTOBJ(UTEST, 0.0, T100, 1.0, T200, Velocity, DIAM,
.      0.228, 0.731, 1./3., fi8729)
G100200 = UTEST * Diam* length$ Apply as conductor
```

Subroutine Name: EXTUBANK

Description: This routine returns an estimate of the average heat transfer coefficient over in-line or staggered tube banks subject to perpendicular external flow. The result may also be applied to pin fin heat sinks, whether ducted or with by-pass flow.

The source of this correlation is the Handbook of Applied Thermal Design, E.C. Guyer (ed.), McGraw-Hill, 1989, Section 4.9. The results of the correlation are intended for deep banks: 20 rows or more of tubes or pin fins. For smaller sets, the heat transfer coefficient should be degraded since the returned result will represent an overestimate in such cases.

For pin fin heat sinks with by-pass, the result of this routine can be modified as follows:

$$U_{\text{pin fin}} = U_{\text{extubank}} * (0.67 + 0.33h/H) \qquad \text{for } h/H > 0.3$$

where h is the pin height and H is the total height of the passage (and therefore H-h is the free space above the tops of the pins). If h=H, then no by-pass exists and therefore no correction is needed. U_{extubank} should be calculated using the velocity upstream of the heat sink. The above correction is valid for h/D in the range of 1 to 3 where D is the pin diameter, and $Re_{D,max}$ (the Reynolds number corrected for the decreased flow area through the pin section) is in the range of 1000 to 9000.

Calling Sequence:

```
CALL EXTUBANK(U,P,T,X,TW,VEL,D,SL,ST,SD,FI)
```

where:

U . . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW. . . . . input surface temperature (user units: R, F, C, or K).

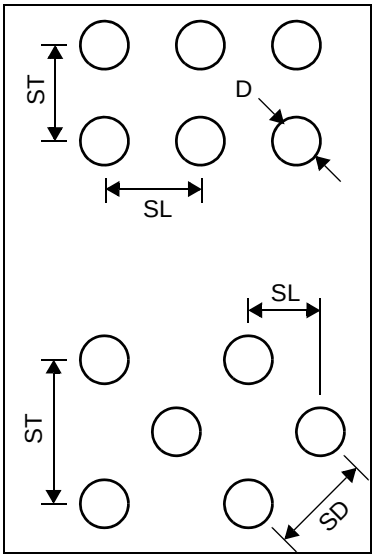
VEL . . . . . input freestream (upstream) velocity (user units: m/s or ft/hr ... *not* ft/s)

D . . . . . input diameter (user units: m or ft)

SL. . . . . input flow-wise distance between rows, whether in-line or staggered (user units: m or ft)

ST. . . . . input transverse distance between columns within same row (user units: m or ft)

SD. . . . . input diagonal distance between adjacent staggered tubes or pins. *Input zero for in-line.* If SL = ST/2, then SD = ±SQRT(2)*SL = ST/SQRT(2) (user units: m or ft).



FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
      CALL EXTUBANK(UTEST, 0.0, T100, 1.0, T200, Velocity, DPIN,
     .       0.0005, 0.001, 0.0005*SQRT(2.0), fi8729)$ staggered
      G100200 = UTEST * pi*dpin*hpin*npins   $ Apply as conductor
C correct for pin fin bypass. Pins are HPIN high in a passage TGAP wide
      G100200 = G100200*(0.67 + 0.33*HPIN/TGAP)
```

7.11.4.3 Jet Impingement Cooling

Correlations for gas jet impingement cooling of a surface perpendicular to the jet centerline(s) are presented, as listed in the Handbook of Applied Thermal Design, E.C. Guyer (ed.), McGraw-Hill, 1989, Section 4.10 pp. 1-70 to 1-72.

Reported limits to these correlations should be obeyed in order to achieve accuracies on the order of 20%. However, with one exception these limits are not enforced by this family of routines in the interest of estimation. That one exception is the minimum distance from the centerline (for single jets) and the maximum spacing of arrayed jets. In those cases, reported limits are applied internally to avoid numerical problems.

The available routines are as follows:

JETSRN. . . . . Single rounded (circular) nozzle
JETSSN . . . . . Single slotted (2D) nozzle
JETARN. . . . . 2D arrays of rounded nozzles (square or offset pattern)
JETASN . . . . . Rows (“arrays”) of slotted nozzles

Subroutine Name: JETSRN

Description: JETSRN calculates average heat transfer for a single rounded nozzle (SRN) gas jet. For a given mass flow rate FR ejecting through a nozzle of diameter D a distance H from a perpendicular surface, JETSRN calculates the average heat transfer coefficient on that surface over a disk of radius R from the impingement point.

The limits of applicability are as follows:

2000 < Re < 4.0e5 . . . . . Nozzle Reynolds number
2.5 < R/D < 7.5 . . . . . relative size of disk
2 < H/D < 12 . . . . . relative distance between surface and nozzle

In general, these limits are recommended but not enforced, with one exception: too-small an input value of R will be replaced with the limit of 2.5D.

The presumed units are dictated by UID, ABSZRO, and PATMOS in CONTROL DATA, GLOBAL.This routine should be called in FLOGIC 0.

Calling Sequence:

```
CALL JETSRN(U, P, T, FR, D, H, R, FI)
```

where:

U . . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T . . . . . input fluid temperature (user units: R, F, C, or K). Ideally, this should be the average temperature of the incoming fluid and the wall.

FR. . . . . nozzle mass flow rate (user units: kg/s or lb_m/hr)

D . . . . . nozzle diameter (user units: m or ft)

H . . . . . distance from nozzle to surface (user units: m or ft)

R . . . . . size of disk for which average will be returned (user units: m or ft)

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
      CALL JETSRN(UTEST, 0.0, FLM.TL100, FLM.FR201, NozzD,
     .       NozzH, Rdisk, fi8729)
      FLM.UA3291 = UTEST * 3.1416*Rdisk**2$ Apply as HTU tie (as example)
```

Subroutine Name: JETSSN

Description: JETSSN calculates average heat transfer for a single slotted nozzle (SSN) gas jet. For a given mass flow rate per unit length, FRL, ejecting through a nozzle of width B a distance H from a perpendicular surface, JETSSN calculates the average heat transfer coefficient on that surface over a rectangle of width 2X centered about the impingement point.

The limits of applicability are as follows:

3000 < Re < 9.0e4 . . . . . Nozzle Reynolds number

4 < X/B < 50. . . . . relative size of plate

4 < H/B < 20. . . . . relative distance between surface and nozzle

In general, these limits are recommended but not enforced, with one exception: too-small an input value of X will be replaced with the limit of 4B.

The presumed units are dictated by UID, ABSZRO, and PATMOS in CONTROL DATA, GLOBAL. This routine should be called in FLOGIC 0.

Calling Sequence:

```
CALL JETSSN(U, P, T, FRL, B, H, X, FI)
```

where:

U. . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)

P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T. . . . . input fluid temperature (user units: R, F, C, or K). Ideally, this should be the average temperature of the incoming fluid and the wall.

FRL . . . . . nozzle mass flow rate *per unit slot length* (user units: kg/s-m or lb_m/hr-ft)

B. . . . . nozzle width (user units: m or ft)

H. . . . . distance from nozzle to surface (user units: m or ft)

X. . . . . distance from center of impingement for which average will be returned (user units: m or ft)

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
CALL JETSSN(UTEST, Modl.PL100, Modl.TL100, Modl.FR201/Length, Width,
.      NozzH, Plate/2., Modl.fi)
FLM.UA3291 = UTEST * Length*Plate$ Apply as HTU tie (as example)
```

Subroutine Name: JETARN

Description: JETARN calculates average heat transfer for arrayed rounded nozzle (ARN) gas jets. For a given mass flow rate per nozzle FR ejecting through nozzle diameters D a distance H from a perpendicular surface, JETARN calculates the average heat transfer coefficient on that surface.

The nozzles may be placed in a square pattern (aligned) or they may be offset (in a triangular or hexagonal grid). In either case, the total surface area per nozzle A is required as an input to define relative nozzle spacing.

The limits of applicability are as follows:

$2000 < Re < 1.0e5$ . . . . . Nozzle Reynolds number
$0.004 < (\pi D^2/4)/A < 0.04$ . . . . . relative spacing
$2 < H/D < 12.$ . . . . . relative distance between surface and nozzles

In general, these limits are recommended but not enforced, with one exception: too-large an input value of A will be replaced with the limit ($0.004*4/\pi D^2$).

The presumed units are dictated by UID, ABSZRO, and PATMOS in CONTROL DATA, GLOBAL. This routine should be called in FLOGIC 0.

Calling Sequence:

```
CALL JETARN(U, P, T, FR, D, H, A, FI)
```

where:

U . . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)
P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.
T . . . . . input fluid temperature (user units: R, F, C, or K). Ideally, this should be the average temperature of the incoming fluid and the wall.
FR. . . . . nozzle mass flow rate *per nozzle* (user units: kg/s or lb_m/hr)
D . . . . . nozzle diameter (user units: m or ft)
H . . . . . distance from nozzle to surface (user units: m or ft)
A . . . . . area served by each nozzle (user units: m^2 or ft^2)
FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Example:

```
CALL JETARN(UTEST, 0.0, FLM.TL100, FLM.FR201/NumNoz, NozzD,
.      NozzH, NozzA, fi8729)
FLM.UA3291 = UTEST * NozzA*NumNoz $ Example: as HTU tie
```

Subroutine Name: JETASN

Description: JETASN calculates average heat transfer for arrayed slotted nozzle (ASN) gas jet. For a given mass flow rate per unit length per slot, FRL, ejecting through a nozzle of width B a distance H from a perpendicular surface, JETASN calculates the average heat transfer coefficient on that surface.

The limits of applicability are as follows:

$1500 < Re < 4.0e4$ . . . . . Nozzle Reynolds number
$0.004 < B/S < 2.5C_r$. . . . . relative spacing, $C_r = \{60+4H^2/(2B-2)^2\}^{-1/2}$
$2 < H/B < 80$ . . . . . relative distance between surface and nozzles

In general, these limits are recommended but not enforced, with one exception: too-large an input value of S will be replaced with the limit of B/0.004.

The presumed units are dictated by UID, ABSZRO, and PATMOS in CONTROL DATA, GLOBAL. This routine should be called in FLOGIC 0.

Calling Sequence:

```
CALL JETASN(U, P, T, FRL, B, H, S, FI)
```

where:

U. . . . . returned average heat transfer coefficient (user units: W/m^2-K or BTU/hr-ft^2-F)
P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.
T. . . . . input fluid temperature (user units: R, F, C, or K). Ideally, this should be the average temperature of the incoming fluid and the wall.
FRL . . . . . nozzle mass flow rate *per unit length per slot* (user units: kg/s-m or lb_m/hr-ft)
B. . . . . nozzle width (user units: m or ft)
H. . . . . distance from nozzle to surface (user units: m or ft)
S. . . . . distance between slots (user units: m or ft)
FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Example:

```
      CALL JETASN(UTEST, Modl.PL100, Modl.TL100,
     .      Modl.FR201/Length/NumSlots, Width, NozzH, Spacing, Modl.fi)
      FLM.UA3291 = UTEST * Length*Spacing*Numslots$ Example: as HTU tie
```

7.11.4.4 Heat Transfer Enhancement for Rough Walls

By the Reynolds' analogy, increases in pressure drop and heat transfer go together. A rough wall will have a higher heat transfer coefficient than a smooth wall in turbulent flow. (In laminar flow, roughness has negligible impact on either pressure drop or heat transfer.)

The default heat transfer routines assume smooth walls. A routine, HTCWRF, is available for estimating the augmentation factor for rough walls, which is best applied as XNTM, the turbulent Nusselt number multiplier for each tie.

Subroutine Name: HTCWRF

Description: HTCWRF returns the heat transfer augmentation factor for turbulent single-phase flows in rough pipes. For a single path (HTN and HTNS ties), this routine returns the result $(f_r/f_s)^n$ where $n=0.68Pr^{0.215}$ where Pr is the Prandtl number and f_r/f_s represents the ratio of rough to smooth friction factors. An upper limit of 4 is placed on this ratio, since no additional augmentation occurs past that point.* This correlation is not limited to an assumption regarding the boundary condition (e.g., isothermal wall vs. constant heat flux). For multiple paths (HTNC ties), averaging is used. The fiction ratio is determined by internal calls to FRICT using actual path WRF factors and zero roughness (i.e., smooth).

This routine should be called in FLOGIC 0. It must be used on an HTN, HTNC, or HTNS tie whose MODW value is 2 or -2. It will function without error for two-phase flows, but the results are valid only for single-phase flow. For two-phase flows with any phase change, or for any tie with MODW not equal to 2 or -2, a unit multiplier is returned.

Calling Sequence:

```
CALL HTCWRF('fmsn', idt, ratio)
```

where:

`fmsn`.....fluid submodel name, in single quotes

`idt`.....User ID of the HTN, HTNC, or HTNS tie (integer). Furthermore, MODW must be 2 or -2 for this tie, otherwise a result of unity will be returned.

`ratio`.....returned augmentation ratio for heat transfer (real). This should ideally be set to XNTM for this tie, but if there are other effects (entrance lengths, bends, etc.) it may be set to another temporary variable used to multiply into XNTM.

* Kays and Crawford, Convective Heat and Mass Transfer, 2nd Ed. McGraw-Hill, 1980.

Examples:

```
      CALL HTCWRF('wahwah', 10, wahwah.xntm10)  $ apply as XNTM
c
      CALL HTCWRF('wahwah', 11, xtest)
      wahwah.xntm11 = xtest*wahwah.xntm11$ combine with other factors
```

7.11.4.5 Heat Transfer Enhancement for Curved Tubes

Secondary flows in curved tubes generate a heat transfer enhancement in both laminar and turbulent flows. The default heat transfer routines assume straight tubes. A routine, HTCURVE, is available for estimating the augmentation factor for curved tubes. The results are best applied as XNLM and XNTM, the laminar and turbulent Nusselt number multipliers for each tie, respectively.

Subroutine Name: HTCURVE

Description: HTCURVE returns the heat transfer augmentation factors for laminar and turbulent single-phase flows in curving pipes of circular cross section. For a single path (HTN and HTNS ties), this routine returns the ratios of curved to straight Nusselt numbers in both the laminar and turbulent regimes.* For multiple paths (HTNC ties), averaging is used.

This routine should be called in FLOGIC 0. It must be used on an HTN, HTNC, or HTNS tie whose MODW value is 2 or -2. The underlying correlations for laminar are based on an isothermal wall (vs. constant heat flux) boundary condition; in turbulent flow no such restriction exists. HTCURVE will function without error for two-phase flows, but the results are valid only for single-phase flow. For two-phase flows with any phase change, or for any tie with MODW not equal to 2 or -2, unit multipliers are returned.

Note that this routine is intended to be used for long segments of (perhaps helically) curved circular tubing, with an isothermal (vs. constant heat flux) wall condition. Application to any other situation will degrade the usefulness of the results, except as a first approximation.

Specifically, the user should note that *this routine cannot be realistically applied to bends, elbows, or tees*, since fully developed secondary flows are assumed. In a bend (e.g., 90° or 180° long radius elbow), a length of about 15 diameters is required to establish the secondary flows, and a similar length of straight exhaust piping is required to dissipate those flows: the heat transfer augmentation is shifted toward the outlet of the elbow, and may not achieve as high an order of magnitude of augmentation as is predicted by HTCURVE.

Calling Sequence:

```
CALL HTCURVE('fmsn', idt, rad, ratlam, raturb)
```

* Handbook of Heat Transfer, Rohsenow, 3rd Ed. 1998. For laminar, Manlapaz-Churchill, Eqn. 5.269-70. For turbulent, Schmit & Pratt, Eqn. 5.279-80.

where:

`fmsn` ..... fluid submodel name, in single quotes

`idt` ..... User ID of the HTN, HTNC, or HTNS tie (integer). Furthermore, MODW must be 2 or -2 for this tie, otherwise a result of unity will be returned.

`rad` ..... radius of curvature (measured from the pipe centerline)

`ratlam` .... returned augmentation ratio for laminar heat transfer (real). This should ideally be set to XNLM for this tie, but if there are other effects (entrance lengths, roughness, etc.) it may be set to another temporary variable used to multiply into XNLM.

`raturb` .... returned augmentation ratio for turbulent heat transfer (real). This should ideally be set to XNTM for this tie, but if there are other effects (entrance lengths, roughness, etc.) it may be set to another temporary variable used to multiply into XNTM.

Examples:

```
      CALL HTCURVE('danandi', 10, rad, xnlm10, xntm10)
                              $ apply as XNLM,XNTM directly
c
      CALL HTCURVE('danandi', 11, rad, xtest, ytest)
      xnlm11 = xtest*xnlm11   $ combine with other factors
      xntm11 = ytest*xntm11   $ combine with other factors
```

7.11.4.6 Heat Transfer Enhancement for Developing Flows (Entrances)

At entrances, high pressure drops and heat transfer rates are experienced as boundary layers develop. In moderate to high Prandtl number fluids, heat transfer rates continue to be higher than in the fully developed region even after momentum boundary layers are fully formed ("hydrodynamic developed flow") until the temperature profiles are also fully developed ("thermal developing flow"). The default heat transfer routines assume fully developed thermal and hydrodynamic flows. A routine, HTCENTR, is available for estimating the augmentation factor for entrance effects. ("Entrance" need not be a literal opening, but can be applied to any region where significant disruption of the boundary layer has occurred, such as the exhaust of orifices and, to a lesser extent, the outlets of mitered joints.) The results are best applied as XNLM and XNTM, the laminar and turbulent Nusselt number multipliers for each tie, respectively.

Subroutine Name: HTCENTR

Description: HTCENTR returns the heat transfer augmentation factors for laminar and turbulent single-phase flows in the entrance regions of pipes of circular cross section. For a single path (HTN and HTNS ties), this routine returns the ratios of developing to fully developed Nusselt numbers in both the laminar and turbulent regimes.* For multiple paths (HTNC ties), averaging is used.

Both a hydrodynamic and thermal entrance is assumed, and two regimes are recognized: developing thermal and developing hydrodynamic, and developing thermal with fully developed hydrodynamic. Fully developed hydrodynamic flow is assumed to exist at a distance $1.359*DH*Re^{1/4}$ from the inlet for turbulent flow (Zhiqing†), where Re is the Reynolds number. For laminar flow, the length required for fully developed flow is 0.056*Re*DH, with corrections by Chen for Re<400. If the flow is also thermally fully developed, no distinction is made by HTCENTR. Rather, the multiplying factors simply approach unity.

This routine should be called in FLOGIC 0. It must be used on an HTN, HTNC, or HTNS tie whose MODW value is 2 or -2. The underlying correlations are based on an isothermal (vs. constant heat flux) boundary condition. HTCENTR will function without error for two-phase flows, but the results are valid only for single-phase flow. For two-phase flows with any phase change, or for any tie with MODW not equal to 2 or -2, unit multipliers are returned.

HTCENTR operates on a piping segment that extends from a length of X1 to a length of X2 from the entrance. X1 and X2 may be entered directly, but it is often more convenient to refer to distance of the tied lump from the inlet, perhaps in terms of the coordinate location of the lump. As a signal to the routine that this mode is selected, the values of X1 and X2 are set equal: both to the distance of the lump from the inlet.

---

* Laminar: Heat Transfer, L. Thomas, Prentice-Hall, 1993, eqn 8-9, 8-12.
Turbulent: Handbook of Heat Transfer, Rohsenow, 3rd Ed. 1998, eqn 5.85-6.

† Handbook of Heat Transfer, Rohsenow, 3rd Ed. 1998, eqn 5.84.

Calling Sequence:

```
      CALL HTCENTR('fmsn',idt, x1, x2, Aper, CEE, EN
+          , ratlam, raturb)
```

where:

`fmsn` ..... fluid submodel name, in single quotes

`idt` ..... User ID of the HTN, HTNC, or HTNS tie (integer). Furthermore, MODW must be 2 or -2 for this tie, otherwise a result of unity will be returned.

`x1` ..... the input distance from the inlet for the start of the segment, must be non-negative and real.

`x2` ..... the input distance from the inlet for the end of the segment. x2 > x1 generally, but if x1=x2, then they both are assumed to represent the distance of the lump from the inlet. In this special mode, internal values of x1 and x2 are calculated using path lengths (TLEN) and tie area fractions, assuming that the latter represent axial partitions (see Aper below if this is not true).

`Aper` ..... the fractional periphery of this tie. This factor is only needed if x1=x2 *and* if more than one node is used circumferentially to represent the pipe wall (or if part of the wall is adiabatic). Axial distances are calculated as $\Delta x = TLEN*A_{fract}/Aper$, where A_{fract} is the specified heat transfer area fraction for each path used by the tie. Thus, $0.0 < Aper < 1.0$.

`CEE` ..... the coefficient "C" in the equation $Nu/Nu_{fd} = raturb = 1.0 + C/(x/DH)^n$, which is applied for combined thermal and hydrodynamic developing flows. If C is nonpositive, a default value of 6 is used. Values of C and the exponent "n" (the next optional argument) are provided in various references.* For the fully developed hydrodynamic regime, an internal value of C is applied (using the Al-Arabi correlation†).

`EN` ..... the exponent "n" in the equation $Nu/Nu_{fd} = raturb = 1.0 + C/(x/DH)^n$, which is applied for combined thermal and hydrodynamic developing flows. If n is nonpositive, a default value of 1 is used. For the fully developed hydrodynamic regime, an internal value of n is applied.

`ratlam` .... returned augmentation ratio for laminar heat transfer (real). This should ideally be set to XNLM for this tie, but if there are other effects (curvature, roughness, etc.) it may be set to another temporary variable used to multiply into XNLM.

`raturb` .... returned augmentation ratio for turbulent heat transfer (real). This should ideally be set to XNTM for this tie, but if there are other effects (curvature, roughness, etc.) it may be set to another temporary variable used to multiply into XNTM.

---

* Handbook of Heat Transfer, Rohsenow, 3rd Ed. 1998, Table 5.13. See also Kays and Crawford, Convective Heat and Mass Transfer, 2nd Ed. McGraw-Hill, 1980.

† Handbook of Heat Transfer, Rohsenow, 3rd Ed. 1998, eqn 5.86.

Examples:

```
c default C and n, input start and end points (zero to Length). The resulting
c factors are average augmentations over that distance.
      CALL HTCENTR('startup', 10, 0.0, Length, 1.0, 0.0, 0.0
     +      ,xnlm10, xntm10)  $ apply as XNLM,XNTM directly
c
c C and n are for a 90 degree miter at the inlet. The axial location
c of the lump (#101) has been specified, so HTCENTR will calculate X1 and
C X2 based on path(s) TLEN and area fractions. But since this tie
C represents only half of the periphery, Aper=0.5. (Otherwise, the
C length of the segment would have been 50% too small if Aper had
C been input as 1.0).
      CALL HTCENTR('startup', 11, CZ101, CZ101, 0.5, 2.0152, 0.614
     +      ,xtest, ytest)
      xnlm11 = xtest*xnlm11   $ combine with other factors
      xntm11 = ytest*xntm11   $ combine with other factors
```

7.11.4.7 Heat Transfer Enhancement: Internal Fins or Protrusions

Heat transfer (and frictional pressure drop) in turbulent flow can be enhanced using transverse or helical fins, wires, crimps, or other protrusions on the insides of tubing (Figure 7-6). As a minimum, these protrusions should extend past the laminar sublayer, which can be estimated as $25*DH*Re^{-0.875}$ for a smooth-walled tube, where DH is the diameter and Re is the Reynolds number. Extending past approximately six times that sublayer causes friction to increase even faster than any heat transfer benefit, and therefore this size usually represents an upper limit to this strategy.

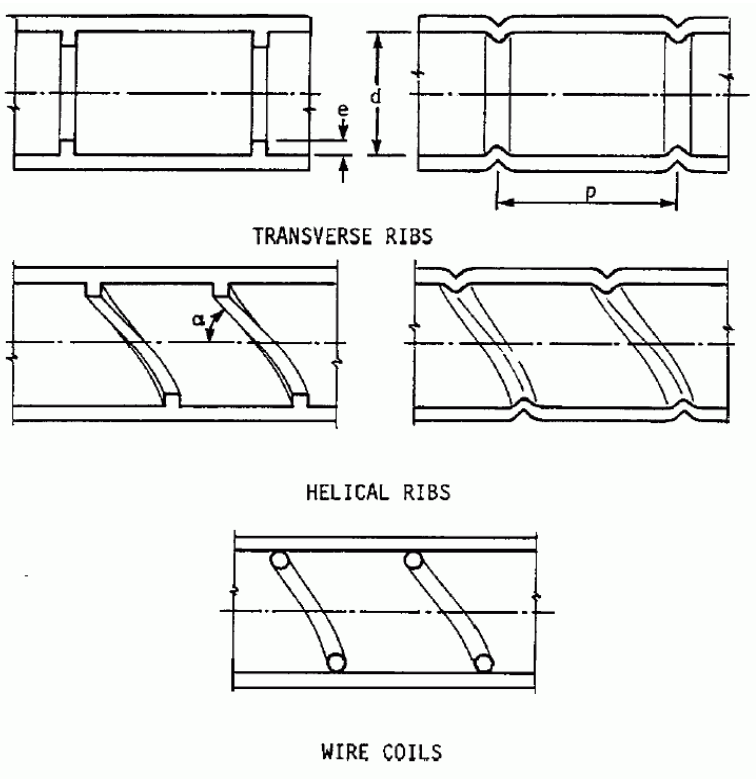


Figure 7-6 Transverse and Helical Ribs, Crimps, Wires, etc.

The default heat transfer routines assume smooth walls. A routine, HTCENH, is available for *roughly* estimating the augmentation factor for transverse or helical fins, wires, etc. The result is best applied as XNTM, the turbulent Nusselt number multiplier for each tie.

HTCENH is based upon the statistical power-law correlation of Ravigururajan and Bergles.* It represents a rough estimate (perhaps off by a factor of two!) that may be used for sizing, or for cases where vendor data is lacking and appropriate margins are therefore applied.

Subroutine Name: HTCENH

Description: HTCENH returns a rough estimate of the heat transfer and friction augmentation factors for turbulent single-phase flows in pipes with transverse or helical (but not axial) grooves, fins, wires, ribs, etc. For a single path (HTN and HTNS ties), this routine returns the ratios of enhanced to smooth Nusselt numbers and friction factors in the turbulent regime. For multiple paths (HTNC ties), averaging is used. HTCENH is not limited to an assumption regarding the boundary condition (e.g., isothermal wall vs. constant heat flux).

* T.S. Ravigururajan, "A Comparative Study of Thermal Design Correlations for Turbulent Flow in Helical-Enhanced Tubes," pp.54- 70, *Heat Transfer Engineering,* Vol. 20 No. 1, 1999.

This routine should be called in FLOGIC 0. It must be used on an HTN, HTNC, or HTNS tie whose MODW value is 2 or -2. It will function without error for two-phase flows, but the results are valid only for single-phase flow. For two-phase flows with any phase change, or for any tie with MODW not equal to 2 or -2, a unit multiplier is returned.

Calling Sequence:

```
      CALL HTCENH('fmsn', idt, hgt, h6, pitch, alpha, beta
     +          ,fratio, ratio)
```

where:

`fmsn`.....fluid submodel name, in single quotes

`idt`.....User ID of the HTN, HTNC, or HTNS tie (integer). Furthermore, MODW must be 2 or -2 for this tie, otherwise a result of unity will be returned.

`hgt`.....input protrusion height (real variable, user units: ft or m): e in Figure 7-6. If zero, unit multiplying factors are returned. If greater than 0.5*DH an error will result. If negative, |hgt| will be used as a scaling factor times the laminar sublayer: $hgt_{actual} = |hgt_{input}|$*h6/6, where h6 is six times the size of the laminar sublayer

`h6`.....returned "maximum" protrusion height: six times the size of the laminar sublayer. Less than h6/6. and heat transfer augmentation is negligible. Greater than h6 and friction grows faster than any heat transfer enhancement.

`pitch`.....input distance between consecutive ribs or wires etc., (real variable, user units: ft or m): p in Figure 7-6.

`alpha`.....input angle of the ribs, fins, or wires (degrees): α in Figure 7-6. 90 degrees means transverse (perpendicular to the flow). Zero degrees is illegal since it means axial grooves, for which this correlation is inapplicable.

`beta`.....input angle of the protrusion to the wall (degrees). This is used only for predicting the friction ratio, not the heat transfer ratio. However, a value of zero should not be used since this is equivalent to "no rib or fin." A value of 90 degrees means a square rib or fin.

`fratio`.....returned friction augmentation ratio (enhanced divided by smooth wall). The number of grooves in the total system is assumed to be very large (e.g., small pitch, large segment, far downstream of the start of enhanced section, etc.). fratio may be supplied as the FCTM value of adjacent tube or STUBE connectors, but *caution should be employed since this value can be very large.*

`ratio`.....returned augmentation ratio for heat transfer (enhanced divided by smooth wall). This should ideally be set to XNTM for this tie, but if there are other effects (entrance lengths, bends, etc.) it may be set to another temporary variable used to multiply into XNTM.

Examples:

```
C use “max” height of 6X lam sublayer, 45 degree square-finned helix
      CALL HTCENH(’bigga’, 10, -6.0, etest, 0.5*diam, 45., 90.,
     +  ftest, bigga.xntm10)  $ apply as XNTM
c
c use hgt/DH of 0.01, tighter pitch (0.3=p/D) transverse (alpha=90) rib
      CALL HTCENH(’bigga’, 11, 0.01*bigga.dh101, htest, 0.3*bigga.dh101
     +      , 90., 90., ftest, xtest)
      bigga.xntm11 = xtest*bigga.xntm11$ combine with other factors
```

7.11.4.8 Natural (Free) Convection Correlations

A variety of correlations for natural convection heat transfer coefficients are available. These may either be applied to thermal conductors (if the freestream fluid temperature is represented by a node), or to fluid ties (if the freestream temperature is represented by a lump).

Unless noted, most correlations are intended for moderate to high Prandtl number fluids (i.e., they are not appropriate for highly conductive or inviscid fluids such as liquid metals). Reasonable ranges of temperature differences and Rayleigh numbers are presumed but not enforced. A Nusselt number of 1.0 (essentially pure conduction in a stagnant fluid) is used as a lower limit for all natural convection routines. For Pr < 20 (fluids such as air, water), an average film temperature is used for properties. For Pr > 20 (e.g., oils) a weighted average is applied using 75% of the wall temperature and 25% of the freestream temperature. Also, Earth gravity is assumed by default (see NCSET).

NCPROP . . . . . Auxiliary routine for properties (Rayleigh number, Prandtl number, etc.)
NCSET. . . . . Auxiliary routine for setting control defaults (gravity, etc.)
NCVFPT . . . . . Vertical (or near-vertical) flat plates, isothermal
NCVFPF . . . . . Vertical (or near-vertical) flat plates, isoflux
NCVCT . . . . . Vertical cylinder, isothermal
NCVCF . . . . . Vertical cylinder, isoflux
NCHSU . . . . . Horizontal flat surface, upward heated or downward cooled
NCHSD . . . . . Horizontal flat surface, downward heated or upward cooled
NCHCT . . . . . Horizontal cylinder, isothermal
NCPPT. . . . . Parallel vertical plates, isothermal (heated plates)
NCPPF. . . . . Parallel vertical plates, isoflux (heated plates)
NCFINS. . . . . Rectangular fins, isothermal (air)
NCRENC. . . . . Thin rectangular enclosure (panel cavity), horizontal, vertical, or tilted
NCCONCYL . . Between concentric, horizontal, isothermal cylinders
NCCONSPH . . Between concentric isothermal spheres
NCSPHERE. . . Gas inside a hotter or colder spherical vessel

Unless otherwise noted, the reference for all correlations is Guyer.*

For information purposes, the coefficients for the laminar and turbulent regimes are often provided even though these are not often used directly as a result. The effective coefficient is usually calculated as $UEFF = (ULAM^m + UTURB^m)^{1/m}$ where “m” is a coefficient specific to each correlation, in the typical range of 4 to 10. This has the effect of using the highest value of ULAM and UTURB, meaning that for short characteristic lengths, laminar flow produces a greater heat transfer rate (and is therefore used) whereas at longer lengths turbulent flow produces a greater rate. This is typical of natural convection analysis: the mode with the highest convection rate tends to persist.

* E.C. Guyer et al, Handbook of Applied Thermal Design, Part 1: Chapter 3, McGraw-Hill, 1989.

Subroutine Name: NCPROP

Description: NCPROP is a utility routine used to calculate fluid property information that is important for natural convection calculations: Rayleigh number,* Prandtl number, and thermal conductivity. It is called internally from all of the SINDA/FLUINT natural convection routines, and is made available to users as a starting point for the addition of their own natural convection correlations.

Calling Sequence:

```
CALL NCPROP(P, TE, X, RAB, PR, COND, BETA, FI)
```

where:

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

TE . . . . . input effective film temperature (user units: R, F, C, or K). Usually, this is the average temperature of the freestream fluid (T_{inf}) and the wall (T_{wall}). For oils and other high Prandtl number fluids (Pr > 20), TE = 0.75*T_{wall} + 0.25*T_{inf} is often used.

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

RAB . . . . . returned base of the Rayleigh number (user units: 1/K-m^3 or 1/°F-ft^3). This is not the Rayleigh number, but rather it contains the fluid-dependent factors that comprise the Rayleigh number. The Rayleigh number can be calculated as follows:

$Ra = RAB*\Delta T*L_c^3$ (constant temperature)

$Ra^* = RAB*q"*L_c^4/COND$ (constant flux)

where ΔT is the temperature difference ($\Delta T = |T_{wall} - T_{inf}|$), L_c is the characteristic length (in meters or feet), q" is the heat flux (W/m^2 of BTU/hr-ft^2), and COND is the thermal conductivity (see below).

PR . . . . . returned Prandtl number of the fluid (unitless).

COND. . . . . returned thermal conductivity (user units: W/m-K or BTU/hr-ft-°F)

BETA . . . . . returned coefficient of thermal expansion (user units: 1/K or 1/R)

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

* the product of the Grashof number and the Prandtl number: Ra = Gr*Pr

Example:

```
TTEST       = 0.5*(TWALL + TFLUID)
CALL NCPROP(0.0, TTEST, 1.0, RTEST, PTEST, CTEST, BTEST, FI8729)
RTEST       = RTEST*ABS(TWALL-TFLUID)*DIAM**3   $ Rayleigh number
```

Subroutine Name: NCSET

Description: NCSET is a utility that changes underlying control constants and physical factors for the set of natural convection correlations. Unless otherwise states, these changes are global to all calls of all natural convection routines documented within this section.

NCSET will normally be called from OPERATIONS before any solutions are invoked, but it can be called as often as needed, including between individual calls to natural convection routines.

Currently, there are two parameters that can be changed:

`GEEF`.....Gravity factor (default 1.0: Earth surface gravity). While this value can be changed to different positive number, the user is cautioned that all correlations were developed for Earth standard conditions, and that any attempt to use them for other circumstances must be viewed as a first-order extrapolation lacking better data or methods. Changing this factor will affect later calls to NCPROP as well as calls to other natural convection correlation routines. It does *not* affect the single-phase mode of HTP ties, which use their own local value (ACCM).

`MINDTF`.....Minimum temperature difference factor (default 1.0e-5). To avoid unrealistically small film coefficients when the temperature difference between the wall and the fluid goes to zero, a minimum is used: 1.0e-5 times the average or characteristic absolute temperature (the exact definition of which varies from routine to routine). This tolerance factor is used mostly for numerical stability since the results of any of the correlations are inappropriate for such small temperature differences. This factor applies to correlations based on constant wall temperature, and not to those based on constant flux (e.g., it applies to NCVFPT but not to NCVFPF).

Calling Sequence:

```
CALL NCSET(NAME, VALUE)
```

where:

`NAME`.....The name of the default control variable, in single quotes:
‘GEEF’ . . .Earth gravity multiplier (1.0 by default)
‘MINDTF’.minimum temperature difference factor (1.0e-5 by default)

`VALUE` .....The value to set (real)

Example:

```
CALL NCSET( 'GEEF', 0.38)     $ Approximate Mars surface gravity
```

Subroutine Name: NCVFPT

Description: NCVFPT calculates the average effective heat transfer coefficient for a vertical flat plate. It may also be applied to off-vertical plates as long as either the top side is hot or the bottom side is cold. Isothermality is assumed, although as a first approximation the results can be applied to a nonisothermal surface by providing the area-weighted average temperature as the effective plate temperature.

Calling Sequence:

```
   CALL NCVFPT(ULAM, UTURB, UEFF, UB, UEDT
+          ,HEIGHT, THETA, P, T, X, TW, FI)
```

where:

ULAM . . . . . returned laminar heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG)

UTURB. . . . . returned turbulent heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW\text{-}T|^{UEDT}$)

HEIGHT . . . . . input length (real, user units: m or ft)

THETA . . . . . input angle *from vertical*, degrees. Zero means a vertical one-sided surface. For an upward heated or downward cooled surface, the angle off vertical may be specified. THETA must be less than 85 degrees.

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW . . . . . input wall temperature (user units: R, F, C, or K).

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
C HEIGHT by WIDTH PLATE. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCVFPT( UTEST, WTEST, UA22, UB22, UEDT22, HEIGHT, 0.0,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = HEIGHT*WIDTH
C
C Apply as a SINDA conductor, 45 degrees off vertical. Let pressure
C “default” to atmospheric
C
      CALL NCVFPT( UTEST, WTEST, G44, BTEST, DTEST, HEIGHT, 45.0,
     +        0.0, TFLOW, 1.0, PLATE.T2044, FI8729)
      G44         = G44*HEIGHT*WIDTH
```

Subroutine Name: NCVFPF

Description: NCVFPF calculates the average effective heat transfer coefficient for a vertical flat plate. It may also be applied to off-vertical plates as long as either the top side is hot or the bottom side is cold. Isoflux (constant heat flux) is assumed, although as a first approximation the results can be applied to a variable flux surface by providing the area-weighted average flux as the effective plate flux.

For fluid property calculations, the user is also required to supply an first estimate of the wall temperature. No internal closure iterations will be made: this guess is assumed sufficient. However, for fluids with strong property variations as a function of temperature, if T + FLUX/UEFF is not roughly equal to TW, the user should consider repeated calls of NCVFPF with improved values of TW. In this case, the UEDT result can help stabilize iterations if needed.

Calling Sequence:

```
      CALL NCVFPF(ULAM, UTURB, UEFF, UB, UEDT
     +          ,HEIGHT, THETA, P, T, X, TW, FLUX, FI)
```

where:

ULAM . . . . . returned laminar heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UTURB . . . . . returned turbulent heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

HEIGHT . . . . . input length (real, user units: m or ft)

THETA . . . . . input angle *from vertical*, degrees. Zero means a vertical one-sided surface. For an upward heated or downward cooled surface, the angle off vertical may be specified. THETA must be less than 85 degrees.

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW . . . . . input estimate of wall temperature (user units: R, F, C, or K). If this estimate is not roughly equal to T + FLUX/UEFF for fluids with strong property variations, iterations may be required.

FLUX . . . . . input wall flux (user units: W/m^2 or BTU/hr-ft^2).

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
C HEIGHT by WIDTH PLATE. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN. Use WALL.T203 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCVFPF( UTEST, WTEST, UA22, UB22, UEDT22, HEIGHT, 0.0,
     +        PL101, TL101, 1.0, WALL.T203, FLUX, FLOMOD.FI)
      AHT22       = HEIGHT*WIDTH
C
C Apply as a SINDA conductor, 45 degrees off vertical. Let pressure
C "default" to atmospheric. Use PLATE.T2044 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCVFPF( UTEST, WTEST, G44, BTEST, DTEST, HEIGHT, 45.0,
     +        0.0, TFLOW, 1.0, PLATE.T2044, FLUX, FI8729)
      G44         = G44*HEIGHT*WIDTH
```

Subroutine Name: NCVCT

Description: NCVCT calculates the average effective heat transfer coefficient for the sides of a vertical cylinder. Isothermality is assumed, although as a first approximation the results can be applied to a nonisothermal surface by providing the area-weighted average temperature as the effective cylinder temperature.

Calling Sequence:

```
      CALL NCVCT(ULAM, UTURB, UEFF, UB, UEDT
     +          ,HEIGHT, DIAM, P, T, X, TW, FI)
```

where:

ULAM . . . . . returned laminar heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UTURB . . . . . returned turbulent heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB . . . . . returned UB for an HTU tie

UEDT. . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

HEIGHT . . . . . input height (real, user units: m or ft)

DIAM. . . . . input diameter (real, user units: m or ft)

P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T. . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW. . . . . input wall temperature (user units: R, F, C, or K).

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Examples:

```
C XLEN by DIAM CYLINDER. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCVCT( UTEST, WTEST, UA22, UB22, UEDT22, XLEN, DIAM,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = XLEN*3.1416*DIAM
C
C Apply as a SINDA conductor. Let pressure “default” to atmospheric.
C
      CALL NCVCT(UTEST, WTEST, G44, BTEST, DTEST, XLEN, DIAM,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44         = G44*XLEN*3.1416*DIAM
```

Subroutine Name: NCVCF

Description: NCVCF calculates the average effective heat transfer coefficient for the sides of a vertical cylinder. Isoflux (constant heat flux) is assumed, although as a first approximation the results can be applied to a variable flux surface by providing the area-weighted average flux as the effective cylinder flux.

For fluid property calculations, the user is also required to supply an first estimate of the wall temperature. No internal closure iterations will be made: this guess is assumed sufficient. However, for fluids with strong property variations as a function of temperature, if T + FLUX/UEFF is not roughly equal to TW, the user should consider repeated calls of NCVCF with improved values of TW. In this case, the UEDT result can help stabilize iterations if needed.

Calling Sequence:

```
      CALL NCVCF(ULAM, UTURB, UEFF, UB, UEDT
     +         ,HEIGHT, DIAM, P, T, X, TW, FLUX, FI)
```

where:

ULAM . . . . . returned laminar heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UTURB. . . . . returned turbulent heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * |TW-T|^{UEDT})

HEIGHT . . . . . input height (real, user units: m or ft)
DIAM. . . . . input diameter (real, user units: m or ft)
P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.
T. . . . . input freestream fluid temperature (user units: R, F, C, or K).
X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
TW. . . . . input estimate of wall temperature (user units: R, F, C, or K). If this estimate is not roughly equal to T + FLUX/UEFF for fluids with strong property variations, iterations may be required.
FLUX. . . . . input wall flux (user units: W/m^2 or $BTU/hr\text{-}ft^2$).
FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
C XLEN BY DIAM CYLINDER. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN. Use WALL.T203 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCVCF( UTEST, WTEST, UA22, UB22, UEDT22, XLEN, DIAM,
     +        PL101, TL101, 1.0, WALL.T203, FLUX, FLOMOD.FI)
      AHT22       = XLEN*3.1416*DIAM
C
C Apply as a SINDA conductor. Let pressure
C "default" to atmospheric. Use CYL.T2044 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCVCF( UTEST, WTEST, G44, BTEST, DTEST, XLEN, DIAM,
     +        0.0, TFLOW, 1.0, CYL.T2044, FLUX, FI8729)
      G44         = G44*XLEN*3.1416*DIAM
```

Subroutine Name: NCHSU

Description: NCHSU calculates the natural convection heat transfer coefficient for a finite horizontal surface with either the hot side up or the cool side down. (See NCHSD for the reverse situation.) Isothermality is assumed, although as a first approximation the results can be applied to a nonisothermal surface by providing the area-weighted average temperature as the effective surface temperature. Since no isoflux version is available, such problems can be solved iteratively (using SINDA to update the wall temperature) using NCHSU as an approximation.

Calling Sequence:

```
      CALL NCHSU(ULAM, UTURB, UEFF, UB, UEDT
     +          , ATOP, P, T, X, TW, FI)
```

where:

ULAM . . . . . returned laminar heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UTURB. . . . . returned turbulent heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

ATOP . . . . . input characteristic length (real, user units: m or ft). This should be calculated as the surface area divided by the perimeter: A/P.

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW . . . . . input wall temperature (user units: R, F, C, or K).

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Examples:

```
C SQUARE PLATE WID X WID. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C A/P = WID**2/(4.*WID) = 0.25*WID
C
      CALL NCHSU( UTEST, YTEST, UA22, UB22, UEDT22, 0.25*WID,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = WID**2
C
C Apply as a SINDA conductor. Let pressure “default” to atmospheric.
C
      CALL NCHSU(UTEST, YTEST G44, BTEST, DTEST, 0.25*WID,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44   = G44*WID**2
```

Subroutine Name: NCHSD

Description: NCHSD calculates the natural convection heat transfer coefficient for a finite horizontal surface with either the hot side down or the cool side up. (See NCHSU for the reverse situation.) Isothermality is assumed, although as a first approximation the results can be applied to a nonisothermal surface by providing the area-weighted average temperature as the effective surface temperature. Since no isoflux version is available, such problems can be solved iteratively (using SINDA to update the wall temperature) using NCHSD as an approximation.

The returned coefficient corresponds to the laminar regime, which persists up to very high Rayleigh numbers.

Calling Sequence:

```
      CALL NCHSD(UEFF, UB, UEDT
     +          , ATOP, P, T, X, TW, FI)
```

where:

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB . . . . . returned UB for an HTU tie

UEDT. . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

ATOP . . . . . input characteristic length (real, user units: m or ft). This should be calculated as the surface area divided by the perimeter: A/P.

P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T. . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW. . . . . input wall temperature (user units: R, F, C, or K).

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Examples:

```
C DISK OF DIAMETER DIAM. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C A/P = 0.25*PI*DIAM**2/(PI*DIAM) = 0.25*DIAM
C
      CALL NCHSD( UA22, UB22, UEDT22, 0.25*DIAM,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = 0.25*3.1416*DIAM**2
C
C Apply as a SINDA conductor. Let pressure "default" to atmospheric.
C
      CALL NCHSD( G44, BTEST, DTEST, 0.25*DIAM,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44   = G44*0.25*3.1416*DIAM**2
```

Subroutine Name: NCHCT

Description: NCHCT calculates the effective external heat transfer coefficient for the sides of a horizontal (semi-infinite) cylinder. Isothermality is assumed, although as a first approximation the results can be applied to a nonisothermal surface by providing the area-weighted average temperature as the effective cylinder temperature. Since no isoflux version is available, such problems can be solved iteratively (using SINDA to update the wall temperature) using NCHCT as an approximation.

Calling Sequence:

```
      CALL NCHCT(UEFF, UB, UEDT
     +          ,DIAM, P, T, X, TW, FI)
```

where:

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW\text{-}T|^{UEDT}$)

DIAM . . . . . input diameter (real, user units: m or ft)

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW. . . . . .input wall temperature (user units: R, F, C, or K).

FI . . . . . .input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
C DIAM CYLINDER. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCHCT( UA22, UB22, UEDT22, DIAM,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = XLEN*3.1416*DIAM
C
C Apply as a SINDA conductor. Let pressure "default" to atmospheric.
C
      CALL NCHCT( G44, BTEST, DTEST, DIAM,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44         = G44*XLEN*3.1416*DIAM
```

Subroutine Name: NCPPT

Description: NCPPT calculates the average effective heat transfer coefficient inside relatively closely spaced, heated vertical parallel plates. It also calculates the optimum spacing in order to maximize heat transfer in a minimum volume (assuming multiple thin plates). Isothermality is assumed within each plate, but each plate may have a different temperature. As a first approximation, the results can be applied to a nonisothermal plate by providing the area-weighted average temperature as the effective plate temperature.

Calling Sequence:

```
   CALL NCPPT(UFD, UTURB, UEFF, UB, UEDT
  +          ,HEIGHT, GAP, GAPOPT, P, T, X, TW1, TW2, FI)
```

where:

UFD . . . . . .returned fully developed heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG)*

UTURB . . . . . .returned turbulent heat transfer coefficient, external flow (entrance) (for information purposes, W/m^2-K if UID=SI or $BTU/hr\text{-}ft^2$-R if UID=ENG)

* Unlike laminar vs. turbulent regimes in external flow, where the regime with the *greatest* heat transfer coefficient dominates, in such open cavity flows the *lowest* coefficient is returned: the mechanism resulting in the bottleneck.

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB. . . . . returned UB for an HTU tie

UEDT . . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

HEIGHT . . . . . input height (real, user units: m or ft)

GAP . . . . . input gap between plates (real, user units: m or ft)

GAPOPT. . . . . *returned* optimum gap size resulting in greatest energy densities (real, user units: m or ft) assuming thin plates and equal plate temperatures. Corresponds to a Nusselt number ($UEFF_{opt}$*GAPOPT/COND) of *about* 1.31.

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T . . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW1 . . . . . input average wall temperature, one plate (user units: R, F, C, or K). Must be greater than T.

TW2 . . . . . input average wall temperature, other plate (user units: R, F, C, or K). Must be greater than T.

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Examples:

```
C HEIGHT by WIDTH PLATES GAP apart.
C Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN.
C
      CALL NCPPT( UTEST, WTEST, UA22, UB22, UEDT22, HEIGHT, GAP,
     +        PL101, TL101, 1.0, WALL.T201, WALL.T202, FLOMOD.FI)
      AHT22       = HEIGHT*WIDTH
      AHT23       = AHT22           $ Apply to second tie (second wall)
      UB23        = UB22
      UEDT23      = UEDT22
C
C Apply as a SINDA conductor. Let pressure “default” to atmospheric.
C
      CALL NCPPT( UTEST, WTEST, G44, BTEST, DTEST, HEIGHT, GAP,
     +        0.0, TFLOW, 1.0, PLATE.T2044, PLATE.2055, FI8729)
      G44         = G44*HEIGHT*WIDTH
      G55         = G44             $ Apply to other plate too
```

Subroutine Name: NCPPF

Description: NCPPF calculates the average effective heat transfer coefficient inside relatively closely spaced, heated vertical parallel plates. It also calculates the optimum spacing in order to maximize heat transfer in a minimum volume (assuming multiple thin plates). Isoflux (constant heat flux) is assumed within each plate, but each plate may have a different flux. As a first approximation, the results can be applied to a variable flux surface by providing the area-weighted average flux as the effective plate flux.

For fluid property calculations, the user is also required to supply an first estimate of the wall temperature (measured at mid height). No internal closure iterations will be made: this guess is assumed sufficient. However, for fluids with strong property variations as a function of temperature, if T + FLUX/UEFF is not roughly equal to TW, the user should consider repeated calls of NCPPF with improved values of TW. In this case, the UEDT result can help stabilize iterations if needed.

Calling Sequence:

```
      CALL NCPPF(UFD, UTURB, UEFF, UB, UEDT
     +          ,HEIGHT, GAP, GAPOPT, P, T, X, TW, FLUX, FI)
```

where:

UFD . . . . . returned fully developed heat transfer coefficient (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)*

UTURB . . . . . returned turbulent heat transfer coefficient, external flow (entrance) (for information purposes, W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG)

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB . . . . . returned UB for an HTU tie

UEDT. . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

HEIGHT . . . . . input height (real, user units: m or ft)

GAP . . . . . input gap between plates (real, user units: m or ft)

GAPOPT . . . . . *returned* optimum gap size resulting in greatest energy densities (real, user units: m or ft) assuming thin plates and symmetric fluxes. Corresponds to a Nusselt number ($UEFF_{opt}$*GAPOPT/COND) of *about* 0.49.

P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T. . . . . input freestream fluid temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

---

* Unlike laminar vs. turbulent regimes in external flow, where the regime with the *greatest* heat transfer coefficient dominates, in such open cavity flows the *lowest* coefficient is returned: the mechanism resulting in the bottleneck.

TW . . . . . input estimate of mid-height wall temperature (user units: R, F, C, or K). If this estimate is not roughly equal to T + FLUX/UEFF for fluids with strong property variations, iterations may be required.

FLUX . . . . . input average wall flux (user units: W/m^2 or $BTU/hr\text{-}ft^2$). If the flux on each plate differs, use $0.5*(q"_1 + q"_2)$. Must be positive.

FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Examples:

```
C HEIGHT by WIDTH PLATES GAP apart.
C Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN. Use WALL.T203 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCPPF( UTEST, WTEST, UA22, UB22, UEDT22, HEIGHT, GAP,
     +        PL101, TL101, 1.0, WALL.T203, FLUX, FLOMOD.FI)
      AHT22       = 2.*HEIGHT*WIDTH   $ Node represents both plates
C
C Apply as a SINDA conductor. Let pressure
C "default" to atmospheric. Use PLATE.T2044 AS
C A FIRST ESTIMATE (THE CURRENT TEMPERATURE FROM PREVIOUS SINDA SOLUTION)
C
      CALL NCPPF( UTEST, WTEST, G44, BTEST, DTEST, HEIGHT, GAP,
     +        0.0, TFLOW, 1.0, PLATE.T2044, FLUX, FI8729)
      G44         = G44*HEIGHT*WIDTH
      G55         = G44             $ Apply to other plate too
```

Subroutine Name: NCFINS

Description: NCFINS calculates the effective heat transfer coefficient for natural convection in U-shaped channels: a finned heat sink mounted sideways such that air can flow upwards (Figure 7-7). Isothermality is assumed within the entire structure (i.e., a high fin efficiency is presumed if the base is heated)*. As a first approximation, the results can be applied to a nonisothermal structure by providing the area-weighted average temperature as the effective structure temperature.

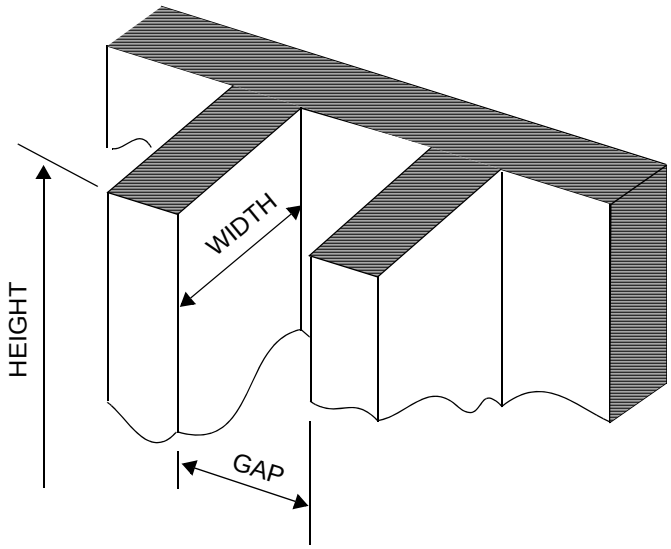


Figure 7-7 Geometry for Subroutine NCFINS

This routine is limited to air, though this is not enforced for the purposes of first-order estimates. Similarly, recommended limits of $0.3 < W/G < 4$ (width to gap ratio), $10 < H/G < 42$ (height to gap ratio), and $0.5 < Ra < 100$ (Rayleigh number†) are suggested but not enforced.

Calling Sequence:

```
      CALL NCFINS(UEFF, UB, UEDT
     +          ,HEIGHT, WIDTH, GAP, P, T, X, TW, FI)
```

where:

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB . . . . . returned UB for an HTU tie

UEDT. . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

* An alternate statement is that fin *inefficiency* is neglected. Finite fin efficiencies should be included separately as a degradation factor if the overall heat transfer conductance from the base to the fluid is desired. Note also that radiation is neglected and should be accounted for in parallel.

† The user can check this using the NCPROP routine, multiplying the resulting RAB by ΔT and by L_c^4/H, where H is the height and L_c is the characteristic length defined by 2WG/(2W+G) where W is the width and G is the gap. For reasonable widths, the gap size for which this correlation is valid is on the order of the GAPOPT values returned from NCPPF and NCPPT. Indeed, testing has shown that the smaller NCPPF value is often closer to the optimum for NCFINS, and might be considered as a starting point when designing a new finned heat sink.

HEIGHT . . . . . input height (parallel to flow) (real, user units: m or ft)
WIDTH. . . . . input width: the length of each fin (perpendicular to flow) (real, user units: m or ft)
GAP . . . . . input gap between fins: approximately the spacing for a thin fin (real, user units: m or ft)
P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.
T . . . . . input freestream fluid temperature (user units: R, F, C, or K).
X . . . . . input fluid quality: must be 1.0 (gas/vapor). This argument is present only for consistency with other routines and future expansion.
TW . . . . . input wall temperature (user units: R, F, C, or K).
FI. . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier. *The correlation is only valid for air, though this is not enforced.*

Examples:

```
C Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCFINS( UA22, UB22, UEDT22, HEIGHT, WIDTH, GAP,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = NUMGAPS*HEIGHT*(2.*WIDTH + GAP)   $ ends neglected
C
C Apply as a SINDA conductor. Let pressure "default" to atmospheric.
C
      CALL NCFINS( G44, BTEST, DTEST, HEIGHT, WIDTH, GAP,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44         = G44*NUMGAPS*HEIGHT*(2.*WIDTH + GAP)  $ ends neglected
```

Subroutine Name: NCRENC

Description: NCRENC calculates the natural convection heat transfer coefficient for a fluid inside a thin rectangular enclosure where one side of the gap (the bottom-most if not vertical) is hotter than the other. Examples of such enclosures include double-paned glass and glass-covered solar panels.

NCRENC may be used for a variety of tilts (angles from horizontal). Isothermality is assumed in each wall. The length (XLEN) is assumed to be greater than the gap (GAP), and should be much larger than the gap for horizontal panels. The width in the horizontal direction is assumed to be much greater than the gap: semi-infinite panel.

This routine invokes a variety of correlations internally, and the dividing lines between them are not always smooth: discontinuities in results may occur as inputs are gradually changed.

Note that NCRENC deals with an enclosure, not with external flow. *The two temperatures are both wall (structural) temperatures, not fluid temperatures.* In other words, the returned coefficient is defined relative to the wall-to-wall temperature difference, not wall-to-fluid. The reference temperature is therefore always the average of hot and cold temperatures regardless of Prandtl number.

The returned U ("UEFF") may be multiplied by the area of a panel (XLEN times width) and used directly as a thermal conductor G between the two wall nodes. If it is instead used as the UA of a tie, then presumably two ties in series are present, and if so the output U should be doubled*: UA = 2*UEFF*AHT where AHT is the heat transfer area (XLEN times width). If UB and UEDT are used to define the tie, then NCRENC presumes that two ties are in series and returns appropriate values. In other words, with most *other* SINDA/FLUINT natural convection routines:

$$UEFF = UB*\Delta T^{UEDT}$$

since a wall-to-fluid conductance is the result. Compare this with the returned values of NCRENC, which presumes the following to be true:

$$UEFF = 0.5*UB*(\Delta T/2)^{UEDT}$$

Specifically for the UB and UEDT values, two ties are assumed to exist in series through a junction (or at least a tank whose temperature does not deviate significantly from the average).

In summary, if UEFF is applied to a SINDA conductor, or if UB and UEDT are applied to two ties in series, then no special modifications or treatment are necessary. On the other hand, *if UEFF is applied to the UA of two tie in series, and if UB=AHT=0.0, then a factor of two must be inserted.*

Calling Sequence:

```
      CALL NCRENC(UEFF, UB, UEDT
     +          ,XLEN, GAP, THETA, P, X, TC, TH, FI)
```

where:

UEFF.....returned effective overall heat transfer coefficient *from TH to TC* (when multiplied by the area, may be used as the G of a conductor or as *half* the UA of an HTU tie if the coefficient is used to connect to a lump in the middle of the enclosure representing the fluid). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB .....returned UB for two HTU ties in series

UEDT.....returned UEDT for two HTU ties in series (UEFF = 0.5 * UB * (0.5*(TH-TC))^{UEDT})

XLEN.....input height (parallel to flow) (real, user units: m or ft)

GAP .....input gap between fins: approximately the spacing for a thin fin (real, user units: m or ft). GAP should be less than XLEN.

* NCRENC cannot know to what use UEFF is applied by the user, so it is the user's responsibility to apply it appropriately. See the examples at the end of this routine's description.

THETA..... input angular tilt *from horizontal*[*] (degrees). Zero means horizontal, 90 means vertical. THETA must within [0.0, 90.0] inclusive.

P..... input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

X..... input fluid quality: must be either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TC..... input cold wall temperature (user units: R, F, C, or K). If tilted or horizontal (THETA < 90), this is assumed to be the topside temperature.

TH..... input hot wall temperature (user units: R, F, C, or K). If tilted or horizontal (THETA < 90), this is assumed to be the bottomside temperature. If TH<TC, zeroes are returned.

FI..... input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier.

Examples:

```
C A vertical plate XLEN tall by WID wide.
C Apply results as a fluid HTU ties #22 and #23 (in series!!)
C Since UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCRENC( UTEST, UB22, UEDT22, XLEN, GAP, 90.0,
     +        PL101, 1.0, WALL.T202, WALL.T203, FLOMOD.FI)
      AHT22       = XLEN*WID
      AHT23       = AHT22           $ Fluid is in center, tie to each side
C the returned coefficient is defined as the film coefficient from wall
C to wall. To instead insert a lump in between, one must double the
C conductance since they'll be in series. THIS IS ALREADY presumed
C by NCRENC for UB and UEDT but not for UEFF, which is wall-to-wall.
      UB23        = UB22            $ this isn't doubled!
      UEDT23      = UEDT23          $ this isn't doubled!
C just for clarity, or in case UB and UEDT are not used:
      UA22        = 2.0*UTEST*XLEN*WID    $ this IS doubled!
      UA23        = UA22
C
C Horizontal plate
C Apply as a SINDA conductor. Let pressure "default" to atmospheric.
C
      CALL NCRENC( GTEST, BTEST, DTEST, XLEN, GAP, 0.0,
     +        0.0, 1.0, CYL.T2045, CYL.T2044, FI8729)
      G44         = GTEST*XLEN*WID  $ this is wall-to-wall conductance
```

* Note that this is not the same angle THETA as defined in NCVFPT or NCVFPF.

Subroutine Name: NCCONCYL

Description: NCCONCYL calculates the natural convection heat transfer coefficient between long, horizontal, concentric, isothermal cylinders.* Examples include gas-gap insulated pipes. "Long" means that the length (XLEN) should be much larger than the gap size (0.5*[DOUT-DIN]), although unit lengths may be input in order to calculate per-unit-length heat flows.

Note that NCCONCYL deals with an enclosure, not with external flow. *The two temperatures are both wall (structural) temperatures, not fluid temperatures.* In other words, the returned coefficient is defined relative to the wall-to-wall temperature difference, not wall-to-fluid. The reference temperature is therefore always the average of hot and cold temperatures regardless of Prandtl number.

Unlike most other routines in this section, the returned argument is UA ("UAEFF"), not U: *area is already contained within the result.* It may therefore used directly as a thermal conductor G between the two wall nodes. If it is instead used as the UA of a tie, then presumably two ties in series are present, and if so the output U should be doubled:† UA = 2*UAEFF. If UB and UEDT are used to define the tie, then NCCONCYL presumes that two ties are in series and returns appropriate values. In other words, with most *other* SINDA/FLUINT natural convection routines:

$$UEFF = UB*\Delta T^{UEDT}$$

since a wall-to-fluid conductance is the result. Compare this with the returned values of NCCONCYL, which presumes the following to be true:

$$UAEFF = 0.5*UBA*(\Delta T/2)^{UEDT}$$

Specifically for the "UBA" (=UB*AHT) and UEDT values, two ties are assumed to exist in series through a junction (or at least a tank whose temperature does not deviate significantly from the average).

In summary, if UAEFF is applied to a SINDA conductor, or if UBA and UEDT are applied to two ties in series, then no special modifications or treatment are necessary (other than the specification of AHT=1.0 for the ties since the UBA result contains the area term already). On the other hand, *if UAEFF is applied to the UA of two tie in series, and if UB=AHT=0.0, then a factor of two must be inserted.*

Calling Sequence:

```
      CALL NCCONCYL(UAEFF, UBA, UEDT
     +          ,DOUT, DIN, XLEN, P, X, TC, TH, FI)
```

* Fundamentals of Heat and Mass Transfer, 5th Ed., F.P. Incropera and D.P. DeWitt, Section 9.8.2 pp 564-5.

† NCCONCYL cannot know to what use UAEFF is applied by the user, so it is the user's responsibility to apply it appropriately. See the examples at the end of this routine's description. Note that a simple doubling implies an equal weighting between the two surfaces, which is an approximation appropriate for small gap sizes. Ideally, the outer diameter should be weighted more because of the greater surface area.

where:

- UAEFF..... returned effective overall heat transfer conductance *from TH to TC* (may be used as the G of a conductor or as *half* the UA of an HTU tie if the coefficient is used to connect to a lump in the middle of the enclosure representing the fluid). Units of W/K if UID=SI or BTU/hr-R if UID=ENG.
- UBA ..... returned UB*AHT for two HTU ties in series
- UEDT ..... returned UEDT for two HTU ties in series (UAEFF = 0.5 * UBA * $(0.5*(TH-TC))^{UEDT}$)
- DOUT ..... input outer diameter (real, user units: m or ft)
- DI..... input inner diameter (real, user units: m or ft)
- XLEN ..... input length (real, user units: m or ft). Specify 1.0 if a per-unit-length result is desired.
- P ..... input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.
- X ..... input fluid quality: must be either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
- TC..... input cold wall temperature (user units: R, F, C, or K).
- TH..... input hot wall temperature (user units: R, F, C, or K). If TH<TC, zeroes are returned.
- FI..... input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier.

Examples:

```
C DESIRED RESULT IS A SINDA CONDUCTOR:
C
      CALL NCCONCYL( CYLS.G1020, atest, btest, Douter, Dinner, SectLen,
     +        -1.0, 1.0, CYLS.T20, CYLS.T10, FI8729)
```

```
C DESIRED RESULT IS INPUTS FOR TWO FLUINT HTU TIES
C
      CALL NCCONCYL( htest, UB101, UEDT101, Douter, Dinner, SectLen,
     +        pl30, xl10, CYLS.T20, CYLS.T10, HOTPIPE.FI)
C
      AHT101 = 1.0      $ these are already contained in “atest”
      AHT201 = 1.0
C the returned coefficient is defined as the film coefficient from wall
C to wall. To instead insert a lump in between, one must double the
C conductance since they’ll be in series. THIS IS ALREADY presumed
C by NCCONCYL for UB and UEDT but not for UEFF, which is wall-to-wall.
      UB201       = UB101           $ this isn’t doubled!
      UEDT201     = UEDT101         $ this isn’t doubled!
C just for clarity, or in case UB and UEDT are not used:
      UA101       = 2.0*HTEST           $ this IS doubled!
      UA201       = UA101
```

Subroutine Name: NCCONSPH

Description: NCCONSPH calculates the natural convection heat transfer coefficient between concentric isothermal spheres.*

Note that NCCONSPH deals with an enclosure, not with external flow. *The two temperatures are both wall (structural) temperatures, not fluid temperatures.* In other words, the returned coefficient is defined relative to the wall-to-wall temperature difference, not wall-to-fluid. The reference temperature is therefore always the average of hot and cold temperatures regardless of Prandtl number.

Unlike most other routines in this section, the returned argument is UA (“UAEFF”), not U: *area is already contained within the result.* It may therefore used directly as a thermal conductor G between the two wall nodes. If it is instead used as the UA of a tie, then presumably two ties in series are present, and if so the output U should be doubled:† UA = 2*UAEFF. If UB and UEDT are used to define the tie, then NCCONSPH presumes that two ties are in series and returns appropriate values. In other words, with most *other* SINDA/FLUINT natural convection routines:

$$UEFF = UB*\Delta T^{UEDT}$$

since a wall-to-fluid conductance is the result. Compare this with the returned values of NCCONSPH, which presumes the following to be true:

$$UAEFF = 0.5*UBA*(\Delta T/2)^{UEDT}$$

* Fundamentals of Heat and Mass Transfer, 5th Ed., F.P. Incropera and D.P. DeWitt, Section 9.8.3 pp 565-6.

† NCCONSPH cannot know to what use UAEFF is applied by the user, so it is the user’s responsibility to apply it appropriately. See the examples at the end of this routine’s description. Note that a simple doubling implies an equal weighting between the two surfaces, which is an approximation appropriate for small gap sizes. Ideally, the outer diameter should be weighted more because of the greater surface area.

Specifically for the “UBA” (=UB*AHT) and UEDT values, two ties are assumed to exist in series through a junction (or at least a tank whose temperature does not deviate significantly from the average).

In summary, if UAEFF is applied to a SINDA conductor, or if UBA and UEDT are applied to two ties in series, then no special modifications or treatment are necessary (other than the specification of AHT=1.0 for the ties since the UBA result contains the area term already). On the other hand, *if UAEFF is applied to the UA of two tie in series, and if UB=AHT=0.0, then a factor of two must be inserted.*

Calling Sequence:

```
      CALL NCCONSPH(UAEFF, UBA, UEDT
     +          ,DOUT, DIN, P, X, TC, TH, FI)
```

where:

UAEFF.....	returned effective overall heat transfer conductance *from TH to TC* (may be used as the G of a conductor or as *half* the UA of an HTU tie if the coefficient is used to connect to a lump in the middle of the enclosure representing the fluid). Units of W/K if UID=SI or BTU/hr-R if UID=ENG.
UBA	returned UB*AHT for two HTU ties in series
UEDT	returned UEDT for two HTU ties in series (UAEFF = 0.5 * UBA (0.5*(TH-TC))^{UEDT})
DOUT	input outer diameter (real, user units: m or ft)
DI.....	input inner diameter (real, user units: m or ft)
P	input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.
X	input fluid quality: must be either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.
TC.....	input cold wall temperature (user units: R, F, C, or K).
TH.....	input hot wall temperature (user units: R, F, C, or K). If TH<TC, zeroes are returned.
FI.....	input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier.

Examples:

```
C DESIRED RESULT IS A SINDA CONDUCTOR:
C
      CALL NCCONSPH( CYLS.G1020, atest, btest, Douter, Dinner,
     +        -1.0, 1.0, CYLS.T20, CYLS.T10, FI8729)
```

```
C DESIRED RESULT IS INPUTS FOR TWO FLUINT HTU TIES
C
      CALL NCCONSPH( htest, UB101, UEDT101, Douter, Dinner,
     +        pl30, xl10, CYLS.T20, CYLS.T10, HOTPIPE.FI)
C
      AHT101 = 1.0      $ these are already contained in “atest”
      AHT201 = 1.0
C the returned coefficient is defined as the film coefficient from wall
C to wall. To instead insert a lump in between, one must double the
C conductance since they’ll be in series. THIS IS ALREADY presumed
C by NCCONSPH for UB and UEDT but not for UEFF, which is wall-to-wall.
      UB201       = UB101           $ this isn’t doubled!
      UEDT201     = UEDT101         $ this isn’t doubled!
C just for clarity, or in case UB and UEDT are not used:
      UA101       = 2.0*HTEST           $ this IS doubled!
      UA201       = UA101
```

Subroutine Name: NCSPHERE

Description: NCSPHERE calculates the natural convection heat transfer coefficient for a gas inside a spherical container that is hotter or colder. Isothermality is assumed in the wall. As a rough estimate, NCSPHERE may be used for nonspherical shapes using an appropriate characteristic length as the “diameter” such as $V^{1/3}$ or perhaps 6*V/A where V is the volume and A is the surface area.

Rayleigh numbers* should be within the range 10^4 to 10^{12}, although no limits are enforced in the interest of producing first-order estimates outside that range.†

NCSPHERE is also employed by HTP ties (Section 3.6.3.3) in the single-phase mode when no port path has been defined. Thus, HTP ties may be an easier way to invoke this convection estimate even in cases where no phase change is expected.

Calling Sequence:

```
      CALL NCSPHERE(UEFF, UB, UEDT
     +          ,DIAM, P, T, X, TW, FI)
```

where:

UEFF . . . . . returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

UB . . . . . returned UB for an HTU tie

UEDT. . . . . returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

* The user can check this using the NCPROP routine, multiplying the resulting RAB by ΔT and by $DIAM^3$.

† The referenced correlation covers two Rayleigh number regimes, with one equation above Ra=1.0e9 and another below that threshold. This transition has been changed to Ra=7.64e7 to preserve numerical continuity.

DIAM . . . . . input sphere diameter (real, user units: m or ft).

P . . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., “-1.0”) will cause atmospheric pressure to be used instead.

T . . . . . input gas temperature (user units: R, F, C, or K).

X . . . . . input fluid quality: must be 1.0 (gas/vapor). This argument is present only for consistency with other routines and future expansion.

TW . . . . . input wall temperature (user units: R, F, C, or K).

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn’t contain a mixture ... use PREPMC before if it does), or smn.FIx where “x” is the desired fluid constituent letter identifier

Examples:

```
C SPHERE OF DIAMETER DIAM. Apply results as a fluid HTU tie #22. Since
C UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C A/P = 0.25*PI*DIAM**2/(PI*DIAM) = 0.25*DIAM
C
      CALL NCSPHERE( UA22, UB22, UEDT22, DIAM,
     +        PL101, TL101, 1.0, WALL.T203, FLOMOD.FI)
      AHT22       = 4.*3.1416*(0.5*DIAM)**2
C
C Apply as a SINDA conductor. Let pressure “default” to atmospheric.
C
      CALL NCSPHERE( G44, BTEST, DTEST, DIAM,
     +        0.0, TFLOW, 1.0, CYL.T2044, FI8729)
      G44   = G44*4.*3.1416*(0.5*DIAM)**2
```

7.11.4.9 Mixed Convection for Filling and Emptying Cylinders

The primary purpose of the NCCYLIN routine is to estimate the average heat transfer coefficient between the wall of a vertical cylindrical pressure vessel and the fluid it contains during pressurization or discharge. However, it can also be used to estimate the heat transfer coefficient in the closed state, of horizontal cylinders, and even noncylindrical vessels.

The reference for this routine is Woodfield (2007),* which in turn references Daney (1976)† for pure natural convection (closed or discharging).

Woodfield et al experimentally determined the heat transfer coefficient in a fixed geometry vertical cylinder while being charged (and therefore heated) from an inlet at the top, or when being discharged (and therefore cooled) from the same port. The cylinder height to diameter ratio was 2.8, and the ratio of the cylinder diameter to the inlet port diameter was 7.5. Despite the fixed geometry used, the results can be generalized because three different gases (hydrogen, nitrogen, and argon) were used.‡

Normally, this combination of natural convection and forced convection is treated by superposition:

$$(Nu_{eff})^n = (Nu_{forced})^n \pm (Nu_{nat'l})^n$$

where the two terms on the right hand side of the equation are summed when the effects are in the same direction (e.g., downward flow from a top port, combined with cold wall) and subtracted when the effects are in opposition to each other (e.g., upward flow from a bottom port, combined with a cold wall). As will turn out to be significant, Woodfield determined that n=1 (linear superposition) was adequate.

Furthermore, Woodfield recommended that pure natural convection estimates (per Daney) were sufficient for the discharge. Daney, in turn, is applicable to horizontal cylinders and indeed other noncylindrical geometries, provided that the boundary layer is turbulent (i.e., strong temperature gradients).

Thus, while NCCYLIN can be used with reasonable confidence to charging, discharging, or sealed vertical cylinders whose charge ports are located on the top, it can also be applied as a first estimate to other situations including horizontal cylinders, cylinders with fill ports on the bottom, and to other noncylindrical pressure vessel geometries as well.

Finally, note that neither the Woodfield nor Daney correlations consider the aspect ratio of the cylinder: its diameter is not an input. Therefore, for very long, thin pressure vessels (in which natural convection is negligible and for which axial velocity gradients might be significant), consider explicitly modeling the cylinder as a FLUINT duct macro.

---

* P. L. Woodfield et al, "Measurement of Averaged Heat Transfer Coefficients in High-Pressure Vessel during Charging with Hydrogen, Nitrogen or Argon Gas," *Journal of Thermal Science and Technology*, Vol. 2, No. 2, 2007, pp.180-189.

† D. E. Daney, "Turbulent Natural Convection of Liquid Deuterium, Hydrogen, and Nitrogen within Enclosed Vessels," *International Journal of Heat and Mass Transfer*, Vol. 19, 1976, pp. 431-441.

‡ Limits were stated in Woodfield as $3.0e5 < Ra_{height} < 4.2e11$, and $270 < Re_{Din} < 1.2e5$

Subroutine Name: NCCYLIN

Description: NCCYLIN calculates the effective internal heat transfer coefficient for a vertical cylinder as it is being charged or discharged, based on both natural convection and forced convection. Isothermality is assumed, although as a first approximation the results can be applied to a noniso-thermal surface by providing the area-weighted average temperature as the effective cylinder temperature.

The intent of this routine is to model the pressurization or discharge of pressure vessels, and to that purpose NCCYLIN can be used as a first approximation for other geometries and orientations (see Guidance below).

NCCYLIN will usually be called from within FLOGIC 0, with the current pressure and temperature of the bulk fluid, the temperature of the wall, as well as the flow rate of an optional fill path, being updated as a function of time during a transient pressurization or discharge event.

The natural convection portion of the NCCYLIN calculations responds to adjustments made in NCSET (Section 7.11.4.8).

NCCYLIN is also employed by HTP ties (Section 3.6.3.3) in the single-phase mode when a port path has been defined. Thus, HTP ties may be an easier way to invoke this convection estimate even in cases where no phase change is expected.

Calling Sequence:

```
      CALL NCCYLIN(UEFF, UB, UEDT,
     +          Height, Din, FRin, Mode, P, T, X, TW, FI)
```

where:

`UEFF` ..... returned effective overall heat transfer coefficient (when multiplied by the area, may be used as the G of a conductor or UA of an HTU tie). Units of W/m^2-K if UID=SI or BTU/hr-ft^2-R if UID=ENG.

`UB` ..... returned UB for an HTU tie

`UEDT` ..... returned UEDT for an HTU tie (UEFF = UB * $|TW-T|^{UEDT}$)

`Height` .... input cylinder length (real, user units: m or ft).

`Din` ..... input flow port diameter (real, user units: m or ft). Ignored if Mode=0. Note: Din is *not* the diameter of the cylinder, but rather the diameter of the port or entrance.

`FRin` ..... input in-flowing mass flow rate (real, user units: kg/s or lb_m/hr). Ignored if Mode=0. Zero or negative values are legal, but treated as if Mode=0 (discharge or hold condition, with only natural convection resulting).

`Mode` ..... input integer mode signal:

- 1: Flow port connected on top of cylinder
- 0: No flow (or negligible flow, or discharge)
- -1: Flow port connected on the bottom of the cylinder

P. . . . . input fluid pressure (user units: psia, psig, Pa). Illegal values such as zero (0.0) or negative (e.g., "-1.0") will cause atmospheric pressure to be used instead.

T. . . . . input (internal) bulk fluid temperature (user units: R, F, C, or K).

X. . . . . input fluid quality: either 0.0 (liquid) or 1.0 (gas/vapor). No fractional (two-phase) values are allowed.

TW . . . . . input wall temperature (user units: R, F, C, or K).

FI . . . . . input fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name (if it doesn't contain a mixture ... use PREPMC before if it does), or smn.FIx where "x" is the desired fluid constituent letter identifier

Guidance: In the case of sealed (zero flow) vessels, consider also NCSPHERE (Section 7.11.4.8) for small temperature differences, since that routine handles the laminar regime.

Guidance: In the case of horizontal cylinders, use *Mode*=-1.

Guidance: For micro-gravity cases, use a prior call to NCSET to lower the gravity level to a small value, in which case *Mode* should not be zero and *FRin* should be positive, since forced convection is all that will remain.

Examples:

```
C Apply results as a fluid HTU tie #430, connecting
C TANK 101 in this submodel (FLOMOD) with node WALL.101. Path 3301,
C a tube or STUBE (because DH was assigned), flows into lump 101.
C Since UB AND AHT ARE SPECIFIED, UA WILL BE OVERWRITTEN
C
      CALL NCCYLIN( utest, UB430, UEDT430, Xleng, DH3301, FR3301,
     +        1, PL101, TL101, 1.0, WALL.T101, FLOMOD.FI)
      AHT430      = Height*3.1416*Dcyl + 0.25*3.1416*Dcyl**2
C
C Apply as a SINDA conductor. Let pressure be 10 bar (SI units),
C and ignore flow in/out (discharging, or holding).
C
      CALL NCCYLIN( G403, btest, dtest, Hcyl, 0.0, 0.0, 0,
     +       10.0D5, TFLOW, 1.0, CYL.T2044, FI8729)
      G403        = G403*3.1416*(Hcyl*Dcyl + 0.25*Dcyl*Dcyl)
```

7.11.4.10 Heat Transfer Correlations for Flow Between Rotating and Stationary Disks

Heat transfer coefficient estimations are provided for rotor-stator systems: radial flow in axial gaps between disks, where one disk is rotating and the other is stationary. The coefficients are best applied between the fluid and the rotating disk, but may be applied to the stationary disk as well. The reference for all of these subroutines is Owen (1989).* The user is cautioned about the limited ranges of applicability for these correlations, which are usually based on air data and/or numerical studies of special cases. They can be used, perhaps, to help quantify the uncertainty in the default SINDA/FLUINT calculations for convection ties applied to paths with nonzero ROTR and RVR<1.

Refer to Section 7.11.7.1 for correlations for torque and perhaps RVR for such flows.

Subroutine Name: HTCRSNF

Description: This routine can be used to calculate local and average heat transfer coefficients for fluid between a rotating and a stationary disk, with no net ("superimposed") flow. Three correlations are employed: one for local laminar coefficients, one for local turbulent coefficients, and one for average turbulent coefficients. Properties and dimensions are not input directly. Instead, they are acquired indirectly by specifying a lump and a path.

Applicability: For the returned HLR and HTR values, the gap size (DH/2) should be between 0.018 and 0.085 of the radius (R = max(RADI,RADJ)), and $Re_\phi < 10^6$ where $Re_\phi = \rho\omega R^2/\mu$. For the returned HTA value, the gap size should be about 0.0646R, and $10^5 < Re_\phi < 3x10^6$.

The formula applied within the routine are *not* functions of gap size:

$$HLR = 0.675 * (k/R) * Re_\phi^{1/2}$$

$$HTR = 0.0217 * (k/R) * Re_\phi^{4/5}$$

$$HTA = 0.0168 * (k/R) * Re_\phi^{4/5}$$

Calling Sequence:

```
CALL HTCRSNF (MODNAM, LUMPNO, PATHNO, HLR, HTR, HTA)
```

* J.M. Owen and R.H. Rogers, Flow and Heat Transfer in Rotating-Disc Systems, Volume 1 - Rotor-Stator Systems, Chapter 8, John Wiley & Sons, 1989.

where:

MODNAM . . . . . fluid submodel name containing lump and path to be used for dimensions and thermodynamic states, input in single quotes such as 'LOOP'

LUMPNO . . . . . integer ID of the lump whose current thermodynamic state is to be used for calculating fluid properties. This lump should be single-phase. If it is two-phase, vapor properties are used.

PATHNO . . . . . integer ID of the path to be used for radius R and rotation rate ω (R=max(RADI,RADJ) and ω is calculated from ROTR). This path can be any type, and it does not need to be adjacent to LUMPNO.

HLR . . . . . output local heat transfer coefficient for laminar flow (calculated at the point on the disk corresponding to R). Treatment of static versus adiabatic surface temperature is unknown.

HTR . . . . . output local heat transfer coefficient for turbulent flow (calculated at the point on the disk corresponding to R). Treatment of static versus adiabatic surface temperature is unknown.

HTA . . . . . output average heat transfer coefficient for turbulent flow (calculated from zero radius to radius R). This particular value assumes that the adiabatic surface temperature has been taken into account.

Example:

```
CALL HTCRSNF ('spinner', 102, 1344, atest, btest, ctest)
```

Subroutine Name: HTCRSIO

Description: This routine can be used to calculate average heat transfer coefficients for fluid between a rotating and a stationary disk. Two correlations are employed: one for radial inflow, and one for radial outflow. Properties, flow rates, and dimensions are not input directly. Instead, they are acquired indirectly by specifying a lump and a path.

Applicability: For the returned HTI (inward flow) value, the gap size (s = DH/2) should be between 0.11 and 0.178 of the radius (R = max(RADI,RADJ)). Furthermore, $2x10^5 < R*C_w/s < 6x10^5$, where $C_w = |FR|/(\mu R)$, and $0.3 < R*Re_\phi/(C_w*s) < 1.3$ where $Re_\phi = \rho\omega R^2/\mu$. The Prandtl number should be near 0.71 (air).

For the returned HTO (outward flow) value, the gap size (s = DH/2) should be between 0.016 and 0.065 of the radius (R = max(RADI,RADJ)). Furthermore, $0.37 < X < 0.47$ where X = min(RADI,RADJ)/max(RADI,RADJ). Also, $5x10^5 < Re_\phi < 4x10^6$, and $0.6 < K_o < 7$ where $K_o = 2\pi*s*Re_\phi*X^2/(C_w*R)$. If X is zero (e.g., because either RADI or RADJ is zero), then HTO will be returned as zero.

Calling Sequence:

```
CALL HTCRSIO (MODNAM, LUMPNO, PATHNO, HTI, HTO)
```

where:

MODNAM .... fluid submodel name containing lump and path to be used for dimensions and thermodynamic states, input in single quotes such as 'LOOP'

LUMPNO .... integer ID of the lump whose current thermodynamic state is to be used for calculating fluid properties. This lump should be single-phase. If it is two-phase, vapor properties are used.

PATHNO .... integer ID of the path to be used for radius R and rotation rate ω (R=max(RADI,RADJ) and ω is calculated from ROTR). The flow rate |FR| of this path will be used to calculate C_w, its DH will be used to calculate the gap size s (s = DH/2), and X will be calculated using R_i = min(RADI,RADJ) for outward flow. This path must be a tube or STUBE connector, but it does not need to be adjacent to LUMPNO.

HTI ..... output average heat transfer coefficient for inward flow. The fluid temperature is assumed static (no correction for recovery).

HTO ..... output local heat transfer coefficient for turbulent flow (calculated at the point on the disk corresponding to R). The fluid temperature is assumed static (no correction for recovery).

Example:

```
CALL HTCRSIO ('spinner', 102, 1344, Hinward, Houtward)
```

7.11.5 Pressure Drop Correlation Functions and Routines

This section describes the frictional pressure drop correlation routines used in FLUINT that are available to the user. The references for these routines are found in Appendix A.

This section describes the following pressure drop routines:

FRICT . . . . . Single-phase laminar and turbulent correlation (Moody Chart fit)
MCADAM. . . . Two-phase Frictional Pressure Gradient (McAdam's Homogeneous)
LOCMAR . . . . Two-phase Frictional Pressure Gradient (Lockhart-Martinelli's)
BAROC . . . . . Two-phase Frictional Pressure Gradient (Baroczy's)
FRIEDL . . . . . Two-phase Frictional Pressure Gradient (Friedel's)

Function Name: FRICT

Description: Returns the Darcy friction factor based on a Reynolds number and a wall roughness ratio. This is a curve fit to the Moody chart.

Calling Sequence:

```
F = FRICT (RE, WRF)
```

where:

F. . . . . Darcy friction factor
RE . . . . . fluid Reynolds number
WRF . . . . . wall roughness fraction (may be zero for smooth walls)

Subroutine Name: MCADAM

Description: This routine returns the two-phase frictional pressure gradient based on McAdam's homogenous flow correlation. See Appendix A.

Restrictions: All units are assumed in standard, absolute form according to the unit flag UID selected in global control data. FR must be greater than zero.

Calling Sequence:

```
 CALL MCADAM (DPZ, VISL, VISV, RHOL, RHOV, FR, XL,
+ DH, AF, WRF)
```

where:

DPZ ..... returned two-phase frictional pressure gradient (pressure per unit length)
VISL ..... liquid viscosity
VISV ..... vapor viscosity
RHOL ..... liquid density
RHOV ..... vapor density
FR..... mass flow rate (positive)
XL..... flow quality
DH..... hydraulic diameter
AF..... flow area
WRF ..... wall roughness fraction

Subroutine Name: LOCMAR

Description: This routine returns the two-phase frictional pressure gradient based on the Lockhart-Martinelli correlation with curve fits by Chisolm. See Appendix A

Restrictions: All units are assumed in standard, absolute form according to the unit flag UID selected in global control data.

Calling Sequence:

```
CALL LOCMAR (DPZ, DPL, DPV, REL, REV)
```

where:

DPZ ..... returned two-phase frictional pressure gradient (pressure per unit length)
DPL ..... liquid pressure gradient if no vapor were present
DPV ..... vapor pressure gradient if no liquid were present
REL ..... liquid Reynolds number if no vapor were present
REV ..... vapor Reynolds number if no liquid were present

Subroutine Name: BAROC

Description: This routine returns the two-phase frictional pressure gradient based on Baroczy's with curve fits by Chisolm. See Appendix A.

Restrictions: All units are assumed in standard, absolute form according to the unit flag UID selected in global control data. FR must be greater than zero.

Calling Sequence:

```
CALL BAROC (DPZ,DPLO,DPVO,FR,AF,XL)
```

where:

DPZ.....returned two-phase frictional pressure gradient (pressure per unit length)
DPLO.....pressure gradient if all mass flow were liquid
DPVO.....pressure gradient if all mass flow were vapor
FR .....mass flow rate (positive)
AF .....flow area
XL .....flow quality

Subroutine Name: FRIEDL

Description: This routine returns the two-phase frictional pressure gradient based on Friedel's correlation. See Appendix A.

Restrictions: All units are assumed in standard, *absolute* form according to the unit flag UID selected in global control data. FR must be greater than zero. FI must be translated; do *not* place an F in column 1 when calling this routine.

Restrictions: If the last argument is of the form "smn.FI", then PREPMC (Section 7.11.2.2) must be called beforehand to define concentrations in multiple-constituent submodels.

Calling Sequence:

```
 CALL FRIEDL (DPZ, RHOL, RHOV, VISL, VISV, XL,
+ FLO, FVO, FR, AF, DH, TL, FI)
```

where:

DPZ.....returned two-phase frictional pressure gradient (pressure per unit length)
RHOL.....liquid density
RHOV.....vapor density
VISL.....liquid viscosity
VISV.....vapor viscosity
FLO.....Darcy friction factor if all mass flow were liquid
FVO.....Darcy friction factor if all mass flow were vapor
XL .....flow quality
FR .....mass flow rate through the segment (positive)
AF .....cross-sectional (flow) area of the segment
DH .....hydraulic diameter of the segment

TL..... absolute temperature of the fluid

FI..... fluid identifier: FIn where n is the fluid ASHRAE number or user-defined fluid number, or smn.FI where smn is the desired fluid submodel name, or smn.FIx where “x” is the desired fluid constituent letter identifier

7.11.6 Auxiliary Pressure Drop Correlations

This section documents user-callable correlations helpful for modeling specific conditions. These correlations and corrections are not used by SINDA/FLUINT unless specifically invoked by the user.

7.11.6.1 Friction Correction for Heated or Cooled Ducts

Subroutine Name: FCPVAR

Description: FCPVAR returns friction correction factors for laminar and turbulent single-phase flows in heated or cooled pipes ("variable property correction"). This routine returns the ratio T_{wall}/T_{bulk} (where T_{wall} is the absolute wall temperature and T_{bulk} is the bulk mean fluid temperature, perhaps corrected for high speed flow effects) along with the viscosity ratio evaluated at those temperatures: μ_{wall}/μ_{builk}. These ratios can be used to calculate custom correction factors if desired. Otherwise, correction factors for at least one recommended method* are also supplied and may be used directly as the FCLM and FCTM factors.

FCPVAR may only be used on tubes or STUBE connectors. Furthermore, the paths must themselves be referenced by an HTN, HTNC, and/or HTNS tie, otherwise wall temperature information is missing. If the path is referenced by multiple ties, area-weighted average temperatures are used. If two-phase flow exists, the temperature ratio will be calculated but all other factors will be returned as unity without errors nor cautions.

This routine should be called in FLOGIC 0.

Calling Sequence:

```
CALL FCPVAR('fmsn', pid, Trat, Vrat, Flam, Fturb)
```

where:

`fmsn`.....fluid submodel name, in single quotes
`pid`.....User ID of the tube or STUBE connector (integer). Furthermore, this path must be used by one or more HTN, HTNC, or HTNS ties.
`Trat`.....returned ratio of absolute temperatures (wall to bulk fluid, real)
`Vrat`.....returned ratio of wall to bulk fluid dynamic viscosity (real)

* E.C. Guyer et al, Handbook of Applied Thermal Design, Chapter 4.8, McGraw-Hill, 1989. See also pp. 477-8 of Heat Transfer, Professional Version by Lindon C. Thomas, 1993.

`Flam`..... returned laminar friction multiplier for heat transfer (real). This should ideally be set to FCLM for this path, but if there are other simultaneous effects it may be set to a temporary variable used to multiply into FCLM.

`Fturb`..... returned turbulent friction multiplier.for heat transfer (real). This should ideally be set to FCTM for this path, but if there are other simultaneous effects it may be set to a temporary variable used to multiply into FCTM.

Examples:

```
c apply as FCLM, FCTM
      CALL FCFVAP('drip', 10, ttest, vtest, drip.fclm10, drip.fctm10)
c combine with other factors
      CALL FCFVAP('drip', 11, ttest, vtest, xtest, ytest)
      drip.fclm11 = xtest*drip.fclm11
      drip.fctm11 = ytest*drip.fctm11
```

7.11.6.2 Friction Correction for Mixed and Rarefied Flows

Subroutine Name: FCRAREF

Description: FCRAREF returns the laminar friction correction factor for rarefied (or mixed rarefied/continuum) single-phase vapor or gas flow. It also returns the Knudsen number, the ratio of mean free path to the characteristic diameter (DEFF). The friction correction factor can be used to calculate custom correction factors* or may be used directly as the FCLM factor.

FCRAREF may only be used on tubes or STUBE connectors. Upstream conditions are used for calculating certain properties such as viscosity, otherwise an average of up and downstream pressures is used for calculations such as density.

For small Knudsen numbers (continuum flow), the returned factor approaches 1.0 (no correction). For Kn > 0.179, a purely rarefied (free molecular flow) correction is applied: Flam = 3π/(128Kn). In between, a mixed-flow correction of Flam = 1/(1+8Kn) is applied. The reference for these correction factors is Barron†, which recommended a break-point between mixed and rarefied flow at Kn=0.3. FCRAREF uses a break-point of Kn=0.179 instead for numerical continuity, given the somewhat arbitrary nature of the physical distinction.

If liquids are present (quality less than 1.0), they are ignored. If no vapor or gas is present, Kn=0.0 and Flam=1.0 is returned without any warning messages.

This routine should be called in FLOGIC 0.

---

* No equivalent correction for heat transfer is available. However, presuming Reynolds' analogy holds for rarefied flow, the returned factor might also be applied as XNLM for adjacent ties.

† Randall Barron, Cryogenic Systems, McGraw-Hill. Sect 8-1, pp 536-541.

Calling Sequence:

```
CALL FCRAREF('fmsn', pid, Xkn, Flam)
```

where:

`fmsn`.....fluid submodel name, in single quotes
`pid`.....User ID of the tube or STUBE connector (integer).
`Xkn`.....returned Knudsen number (real)
`Flam`.....returned laminar friction multiplier for heat transfer (real). This should ideally be set to FCLM for this path, but if there are other simultaneous effects it may be set to a temporary variable used to multiply into FCLM.

Example:

```
c apply as FCLM directly
      CALL FCRAREF('rare', 11, xtest, rare.fclm11)
c combine with other factors
      CALL FCRAREF('rare', 12, xtest, ftest)
      rare.fclm12 = ftest*rare.fclm12
```

7.11.7 Auxiliary Correlations for Torques in Gaps

For rotating paths with walls that are not co-rotating (RVR<1.0, Section 3.26.2), the rotation does not cause pumping or turbine action, but rather dissipation. In this case, INTORQ=YES should be specified for such paths and the TORQ should be input rather than output, per Section 3.26.3.2.

This section provides correlations for dissipative torque, and perhaps even the RVR value itself, for flows in cylindrical gaps (axial flow) or disk-shaped gaps (radial flow).

7.11.7.1 Flows Between Rotating and Stationary Disks

Estimations of torque (and therefore heating rate) and core velocity ratio ("RVR" in SINDA/FLUINT) are provided for rotor-stator systems: radial flow in axial gaps between disks, where one disk is rotating and the other is stationary.* The reference for all of the subroutines that are documented in this subsection is Daily and Nece (1960).† The user is cautioned that these correlations are applicable for aspect ratios (gap divided by radius) of no more than 0.3 (and no more than 0.2 for empirical methods). Single-phase flow is assumed. The user is further cautioned that these correlations were developed for a two-sided disk: use a nominal scaling factor ("TSCALE" argument) of 0.5 when a path represents only one side of a disk.

Three ways are provided to apply these correlations to one or more paths, and a fourth method is applicable for direct user calls without referencing a path:

TORDISK1. . . Determines TORQ and RVR for one tube or one STUBE connector
TORDISKN . . Determines TORQ and RVR for N tubes or STUBE connectors
TORDISKM . . Determines TORQ and RVR for all paths in a duct macro
TORDISKU . . Determines TORQ and RVR given user inputs

All of these routines calculate both TORQ (N-m or lb_f-ft, depending on whether UID=SI or UID=ENG) and RVR values based on the same methods. Most of these routines set INTORQ=YES for the named paths. All routines should be called within FLOGIC 0.‡ The first three are only applicable to tube and STUBE connectors, the last one (TORDISKU) can be applied to any type of path or might be used for other user calculations. When applied directly to a tube or STUBE connector, the hydraulic diameter DH is assumed to be twice the gap size, and the RADI, RADJ, and ROTR values are assumed to have been set before calling these routines. (TCF is also relevant since the heating rate QTMK will be later calculated on the basis of the TORQ value that these routines calculate.) The velocity angles (VAI, VAJ, VXI, and VXJ) are ignored, allowing these routines to be applied to cases that are not perfectly represented by purely radial flows. However, RADI and RADJ should not be identical (see Section 7.11.7.2 for cylindrical correlations) otherwise zero torque

* Refer to Section 7.11.4.9 for correlations for heat transfer between the fluid and wall for such flows.

† J.W. Daily and R.E. Nece, "Chamber Dimension Effects on Induced Flow and Frictional Resistance of Enclosed Rotating Disks," Journal of Basic Engineering, March 1960, pp 217-232. The assistance of D. Mohr of D&E Propulsion & Power is gratefully acknowledged.

‡ As will be noted later, if TORDISK1 is used in a query mode instead of a calculation mode, it can be called from any location, including OPERATIONS and OUTPUT CALLS.

will be returned. No presumption is made about positive or negative flow rates, nor the relative sizes of RADI and RADJ (i.e., RADI may be greater than RADJ or vice versa).

The net torque on an annular section of disk is calculated by calculating the torque for a whole disk of radius R_o = max(RADI,RADJ) and subtracting from it the torque of a smaller disk of radius R_i = min(RADI,RADJ). The torque will be a positive value since it is imposed upon the fluid. The core velocity ratio, RVR ($K=\beta/\omega$ in the original reference), will be calculated at R_o since that outer location has much more effect on the flow and on the torque.

Before the scaling factor TSCALE is applied, torques are calculated via torque coefficients as follows (for a whole, two-sided disk):

$$TORQ = 0.5*C_m*\rho*\omega^2*a^5$$

where C_m is the torque coefficient (calculated via correlation), ρ is the density, ω is the rotational speed (rad/sec), and a is the disk radius. As applied by the subroutines within this section to an annular ring, this becomes:

$$TORQ = 0.5*C_{m,Ro}*\rho*\omega^2*R_o^5 - 0.5*C_{m,Ri}*\rho*\omega^2*R_i^5$$

The primary scaling factors for C_m are the gap ratio (s/a, where s is the gap size assumed to be DH/2, and a is the disk radius), and the rotational Reynolds number, $Re_\phi = \rho\omega a^2/\mu$. (Note that this Reynolds number is distinct from the FLUINT parameter REW, which includes the gap size s instead of the radius a as the characteristic dimension: $REW = \rho\omega as/\mu$.)

All of these routines can be used to apply one of three methodologies over one of four regimes. The methodologies include:

METH=1 . . . . . Classic correlation set (the correlations available prior to Daily & Nece)
METH=2 . . . . . "Best Empirical" correlation set (curve fits to available test data)
METH=3 . . . . . "Best Theoretical" correlation set (correlations developed by Daily & Nece using both scaling relationships and Navier-Stokes solutions at cross sections).

The regimes include (following the naming scheme in Daily & Nece):

LREG=1 . . . . . Laminar closely spaced (no core flow: boundary layers meet)
LREG=2 . . . . . Laminar large spacing (boundary layers do not meet)
LREG=3 . . . . . Turbulent closely spaced (no core flow: boundary layers meet)
LREG=4 . . . . . Turbulent large spacing (boundary layers do not meet)

Discerning between these regimes can be difficult. Therefore, the option of letting the subroutines choose the regime is provided via LREG=0. When LREG=0, the boundary layer sizes are not explicitly calculated nor is a transitional Reynold's number applied. Rather, the code simply uses the regime that returns the largest torque value. This simple method provides a computationally fast, physically consistent, and numerically continuous approach for autonomous regime determination.

TORDISK1 applies to a single tube or STUBE connector. It may be used to set the TORQ or RVR values, or to just calculate torques and velocity ratios without actually setting path parameters. If LREG=0 (autonomous regime determination), it returns the identifier of the regime selected.

TORDISK1 is flexible and informative, but can be laborious to apply to many paths. Therefore, other utilities are available to apply the Daily & Nece correlations to all of the paths within a duct macro (TORDISKM) or to all paths within a named sequence (TORDISKN). These multiple-path options do not return regime information, and they set the TORQ and RVR values directly for all listed paths. Therefore, a call to TORDISK1 can be used *in addition to* the multiple-path options to query values for individual paths as needed.

TORDISKU is available for cases in which torque is needed for user calculations, and is not to be applied directly to a tube or STUBE connector. It is also applicable for seals and rubbing passages that are not modeled using tubes or STUBE connectors (e.g., a face, ring, or labyrinth seal modeled using a LOSS or TABULAR device).

Subroutine Name: TORDISK1

Description: This routine can be used to calculate the torque applied to the fluid within a gap between a rotating and a stationary disk. The rotation speed (ROTR) is assumed to be fast enough to dominate the problem compared to net flows (path FR). Single-phase flow is assumed although the routines will not abort for two-phase flows.* The upstream density is used to determine the rotational Reynolds number, $Re_\phi = \rho\omega a^2/\mu$.

The user is cautioned that this routine applies to a two-sided disk: use a nominal scaling factor ("TSCALE" argument) of 0.5 when a path represents only one side of a disk.

As described above, four regimes are available as determined by an input flag (LREG), including an option for autonomous determination of the regime (LREG=0). In addition, three correlation sets are available as determined by an input flag (METH).

METH=1: Classic Correlation Set

For close-spaced cases (LREG=1 or LREG=3), an RVR value of half (0.5) is returned. For LREG=2, an RVR of 0.512 is returned. For LREG=4, RVR will be set to 0.538.

The equations for the torque coefficient, Cm, are as follows:

Regime 1: $C_m = 2\pi / [(s/a) * Re_\phi]$

Regime 2: $C_m = 2.66 / Re_\phi^{1/2}$

Regime 3: $C_m = 0.0622 / [(s/a) * Re_\phi]^{1/4}$

* A McAdam's effective viscosity approach will be used for two-phase flows.

Regime 4: $C_m = 0.0836 / Re_\phi^{1/5}$

METH=2: "Best Empirical" Correlation Set

An RVR value of half (0.5) is returned for close-spaced cases (LREG=1 or LREG=3). For large spacings (LREG=2 and LREG=4), interpolated tables of RVR as a function of s/a are used from approximately $0.05 < s/a < 0.2$ with no extrapolations past those limits.

The equations for the torque coefficient, Cm, are as follows:

Regime 1: $C_m = 2\pi / [(s/a) * Re_\phi]$

Regime 2: $C_m = 3.70 * (s/a)^{1/10} / Re_\phi^{1/2}$

Regime 3: $C_m = 0.080 / [(s/a)^{1/6} * Re_\phi^{1/4}]$

Regime 4:* $C_m = 0.102 * (s/a)^{1/10} / Re_\phi^{1/5}$

METH=3: "Best Theoretical" Correlation Set

An RVR value of half (0.5) is returned for close-spaced cases (LREG=1 or LREG=3). For large spacings (LREG=2 and LREG=4), interpolated tables of RVR as a function of s/a are used from $0.025 < s/a < 0.275$ with no extrapolations past those limits. These tables correspond to Figures 3b (LREG=2) and 5b (LREG=4) in the original reference (Daily & Nece, 1960).

The equations for the torque coefficient, Cm, are as follows:

Regime 1: $C_m = 2\pi / [(s/a) * Re_\phi]$

Regime 2: $C_m = C_1 / Re_\phi^{1/2}$
C_1 is interpolated from a table as a function of s/a, from approximately $0.025 < s/a < 0.275$ with no extrapolations past those limits.
(Figure 3a in Daily & Nece.)

Regime 3: $C_m = 0.0622 / [(s/a) * Re_\phi]^{1/4}$

Regime 4: $C_m = C_1 / Re_\phi^{1/5}$
C_1 is interpolated from a table as a function of s/a, from approximately $0.025 < s/a < 0.275$ with no extrapolations past those limits.
Figure 5a in Daily & Nece.

Calling Sequence:

```
      CALL TORDISK1 (MODNAM, PATHNO, METH, LREG,
     +          LROUT, TSCALE, TORQO, RVRO)
```

* In the original reference, the coefficient is actually "0.0102" which is believed to be a typographical error.

where:

MODNAM .... fluid submodel name containing the path (PATHNO) to be used for dimensions input in single quotes such as 'LOOP'

PATHNO .... integer ID of the path to be used for radius R and rotation rate ω (R_o=max(RADI,RADJ), R_i=min(RADI,RADJ), and ω is calculated from ROTR). This path must be a tube or STUBE connector. The gap size is assumed to be s=DH/2.

METH ..... The input integer correlation set identifier (refer to discussion in introduction):
1: Classic set
2: "Best Empirical" set
3: "Best Theoretical" set

LREG ..... The input integer regime identifier (refer to discussion in introduction):
1: Laminar, closely spaced (Regime I in Daily & Nece)
2: Laminar, large spacing (Regime II in Daily & Nece)
3: Turbulent, closely spaced (Regime III in Daily & Nece)
4: Turbulent, large spacing (Regime IV in Daily & Nece)
0: Autodetermine regime (based on maximum C_m)

LROUT..... The actual regime identifier used (integer). If LREG is nonzero, LROUT=LREG. Otherwise LROUT contains 1 through 4. This is the regime at R_o (outer radius), which may differ from that used at R_i (inner radius).

TSCALE .... Torque scaling factor applied to two-sided torque before returning. Use TSCALE=0.5 to return half the torque, as applicable to a path representing only one side of a disk. This value can also be used to model roughness, or as a parameter to assist correlation to test data.

TORQO..... Output total torque on the fluid (real, units of N-m or lb_f-ft, depending on UID). This may be specified directly as TORQ for the path assuming TCF is the correct value.

RVRO ..... Output core velocity ratio (real). This is the value at R_o (outer radius), which may differ from the velocity ratio at R_i (inner radius).

Caution: Whether or not it is used in the calculation or query mode, this routine sets INTORQ=YES for the path, meaning that TORQ is an input and QTMK (and EFFP) are outputs.

Guidance: Should be called from FLOGIC 0 if the last two arguments are the TORQ and RVR values for the path (calculation mode, Example #1 below). Otherwise can be called from anywhere (perhaps to determine LROUT, query mode, Example #2 below).

Example:

```
CALL TORDISK1 ('loopD', 323, 3, 0, ltest, 0.5, torq323, rvr323) $ Ex 1
CALL TORDISK1 ('Dloop', 441, 3, 0, ltest, 0.5, ttest, rtest)    $ Ex 2
```

Subroutine Name: TORDISKM

Description: This routine is the same as TORDISK1, except (1) it is applied to all paths within a duct macro (Section 3.9.2), and (2) it can only be used in the "calculation mode" since it sets the values of TORQ and RVR directly for all macro paths.

Calling Sequence:

```
CALL TORDISKM (MODNAM, MACID, METH, LREG, TSCALE)
```

where:

MODNAM . . . . . fluid submodel name containing the paths for which TORQ and RVR values are to be set

MACID . . . . . . integer ID of the duct macro containing the paths. These RADI, RADJ, ROTR, and DH values of each such path will be used to determine the TORQ and RVR values as explained above.

METH. . . . . The input integer correlation set identifier (refer to discussion in introduction):
1: Classic set
2: "Best Empirical" set
3: "Best Theoretical" set

LREG. . . . . The input integer regime identifier (refer to discussion in introduction):
1: Laminar, closely spaced (Regime I in Daily & Nece)
2: Laminar, large spacing (Regime II in Daily & Nece)
3: Turbulent, closely spaced (Regime III in Daily & Nece)
4: Turbulent, large spacing (Regime IV in Daily & Nece)
0: Autodetermine regime (based on maximum C_m)

TSCALE . . . . . Torque scaling factor applied to two-sided torque before returning. Use TSCALE=0.5 to return half the torque, as applicable to a path representing only one side of a disk. This value can also be used to model roughness, or as a parameter to assist correlation to test data.

Guidance: Should be called from FLOGIC 0. Since the TORQ value is set directly, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
CALL TORDISKM ('disZ', 31, 2, 0, 0.5)
```

Subroutine Name: TORDISKN

Description: This routine is the same as TORDISK1, except (1) it is applied to all paths within a list, and (2) it can only be used in the "calculation mode" since it sets the values of TORQ and RVR directly for all listed paths. The paths are listed according to their IDs: initial, total, and increment values as with the input option GENPA.

Calling Sequence:

```
CALL TORDISKN (MODNAM, N1, NUM, INC, METH, LREG, TSCALE)
```

where:

MODNAM .... fluid submodel name containing the paths for which TORQ and RVR values are to be set

N1..... integer ID of the first tube or STUBE connector

NUM ..... Total number of paths listed (integer)

INC ..... Increment of the ID, such that the ID of the paths listed are N1, N1+INC, N1+2*INC, ... N1 + (NUM-1)*NINC.

METH ..... The input integer correlation set identifier (refer to discussion in introduction):
1: Classic set
2: "Best Empirical" set
3: "Best Theoretical" set

LREG ..... The input integer regime identifier (refer to discussion in introduction):
1: Laminar, closely spaced (Regime I in Daily & Nece)
2: Laminar, large spacing (Regime II in Daily & Nece)
3: Turbulent, closely spaced (Regime III in Daily & Nece)
4: Turbulent, large spacing (Regime IV in Daily & Nece)
0: Autodetermine regime (based on maximum C_m)

TSCALE .... Torque scaling factor applied to two-sided torque before returning. Use TSCALE=0.5 to return half the torque, as applicable to a path representing only one side of a disk. This value can also be used to model roughness, or as a parameter to assist correlation to test data.

Guidance: Should be called from FLOGIC 0. Since the TORQ value is set directly, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
CALL TORDISKN ('spinHQ', 101,10,1, 3, 0, 0.5)
```

Subroutine Name: TORDISKU

Description: This routine is the same as TORDISK1, except that it is need not reference a tube or STUBE, and requires instead the user to specify dimensions and rotational speed as arguments. It can therefore be used either for user calculations, or the results can be applied to a different connector device such as a LOSS or TABULAR. Because it does not reference a path directly, it does not set INTORQ=YES as do TORDISK1 and the other routines described above; if TORDISKU is to be applied to a path, set INTORQ=YES in FLOW DATA.

NOTE: If the purpose of calculating TORQ is to have the program calculate the dissipative heating term, QTMK, for some path, then set ROTR for the path even if there is no spinning force. Otherwise, QTMK will remain zero if ROTR=0.0, even if INTORQ=YES and TORQ is nonzero.

Even though a path is not referenced directly, thermodynamic state data is still required. This can be supplied by either naming a path of any type, whose current upstream end is to be used as the state for density and viscosity calculations, or by naming a lump directly.

Calling Sequence:

```
      CALL TORDISKU (MODNAM, NPL, GAP, RMIN, RMAX, RPM,
     +          METH, LREG, LROUT, TSCALE, TORQO, RVRO)
```

where:

MODNAM . . . . . Fluid submodel name containing the lump or path to be used as thermodynamic state, in single quotes

NPL . . . . . ID of path whose current upstream state should be used for calculations of density and viscosity. If negative, the ID of a lump that should be used directly

GAP . . . . . Input gap size (in ft or m, depending on UID, real variable)

RMIN. . . . . Input inner radius R_i (in ft or m, depending on UID, real variable), may be zero to calculate for a whole disk

RMAX. . . . . Input outer radius R_o (in ft or m, depending on UID, real variable)

RPM . . . . . Rotational speed in revolutions per minute (RPM)

METH. . . . . The input integer correlation set identifier (refer to discussion in introduction):
1: Classic set
2: “Best Empirical” set
3: “Best Theoretical” set

LREG . . . . . The input integer regime identifier (refer to discussion in introduction):
1: Laminar, closely spaced (Regime I in Daily & Nece)
2: Laminar, large spacing (Regime II in Daily & Nece)
3: Turbulent, closely spaced (Regime III in Daily & Nece)
4: Turbulent, large spacing (Regime IV in Daily & Nece)
0: Autodetermine regime (based on maximum C_m)

LROUT . . . . . The actual regime identifier used (integer). If LREG is nonzero, LROUT=LREG. Otherwise LROUT contains 1 through 4. This is the regime at R_o (outer radius), which may differ from that used at R_i (inner radius).

TSCALE . . . . Torque scaling factor applied to two-sided torque before returning. Use TSCALE=0.5 to return half the torque, as applicable to a one-sided disk. This value can also be used to model roughness, or as a parameter to assist correlation to test data.

TORQO . . . . . Output total torque on the fluid (real, units of N-m or lb_f-ft, depending on UID). This may be specified directly as TORQ for a path assuming TCF is the correct value.

RVRO . . . . . Output core velocity ratio (real). This is the value at R_o (outer radius), which may differ from the velocity ratio at R_i (inner radius).

Guidance: Should be called from FLOGIC 0 if used to set the TORQ or RVR value of a path. If TORQ is to be set for a path, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
      CALL TORDISKU ('labs', -10, 2, 0, 0.01/12., 0.0, 2.3/12., Rspeed,
     +            ltest, 0.5, ttest, rtest)  $ use lump 10's state
```

7.11.7.2 Flows Between Rotating and Stationary Cylinders

Estimations of torque (and therefore heating rate) are provided for flow between concentric cylinders, the inner one rotating and the outer one still (e.g., flow between a rotor shaft and a bearing housing). The reference for all of the subroutines that are documented in this subsection is Bilgen and Boulos (1973).* Single-phase flow is assumed.

Three ways are provided to apply these correlations to one or more paths, and a fourth method is applicable for direct user calls without referencing a path:

TORCYL1 . . . . Determines TORQ for one tube or one STUBE connector
TORCYLN . . . Determines TORQ for N tubes or STUBE connectors
TORCYLM . . . Determines TORQ for all paths in a duct macro
TORCYLU . . . Determines TORQ given user inputs

Note that unlike the TORDISK series of routines, no assistance is available for estimating RVR. Therefore, a value of RVR=0.5 is recommended if no further information is available.

All of these routines calculate TORQ (N-m or lb_f-ft, depending on whether UID=SI or UID=ENG) based on the same methods. Most of these routines set INTORQ=YES for the named paths. All routines should be called within FLOGIC 0. The first three are only applicable to tube and STUBE connectors, the last one (TORCYLU) can be applied to any type of path or might be used for other user calculations. When applied directly to a tube or STUBE connector, the hydraulic diameter DH is assumed to be twice the gap size s (s = DH/2),† the length of the section h is assumed to be the same as TLEN (h = TLEN), and the RADI, RADJ, and ROTR values are assumed to have been set before calling these routines. (TCF is also relevant since the heating rate QTMK will be later calculated on the basis of the TORQ value that these routines calculate.) The velocity angles (VAI, VAJ, VXI, and VXJ) are ignored, allowing these routines to be applied to cases that are not perfectly represented by purely axial flows. Normally, RADI=RADJ, but if those two parameters are unequal then the maximum value is used as the inner (shaft) radius: $R_i = \max(RADI,RADJ)$. No presumption is made about positive or negative flow rates. The torque will be a positive value since it is imposed upon the fluid.

Before the scaling factor TSCALE is applied, torques are calculated via torque coefficients as follows:

$$TORQ = 0.5*\pi*C_m*\rho*\omega^2*R_i^4*h$$

where C_m is the torque coefficient (calculated via correlation), ρ is the density, ω is the rotational speed (rad/sec), R_i is the rotor shaft (inner cylinder) radius, and h is the axial length.

---

* E. Bilgen and R. Boulos, "Functional Dependence of Torque Coefficient of Coaxial Cylinders on Gap Width and Reynolds Number," ASME Journal of Fluids Engineering, March 1973, pp 122-126. The assistance of D. Mohr of D&E Propulsion & Power is gratefully acknowledged.

† In the original reference (Bilgen and Boulos), "s" is actually the axial gap and "T" is the radial gap. In this subsection, "s" is instead used as the gap width to preserve consistency with the prior subsection (Daily & Nece).

The primary scaling factors for C_m are the gap ratio (s/R_i, where s is the gap size, and R_i is the radius of the inner cylinder), and the rotational Reynolds number, $Re_\omega = \rho\omega R_i s/\mu$. (Note that this Reynolds number is the same as the FLUINT parameter REW.)

All of the cylinder torque routines calculate the regime of operation and base the C_m on the appropriate regime: laminar ($Re_\omega < 64$), transition ($64 < Re_\omega < 500$, subdivided as large and small gap), low turbulent ($500 < Re_\omega < 10^4$), and high turbulent ($Re_\omega > 10^4$). The formula applied are as follows:

Laminar:	$C_m = 8\Phi / Re_\omega$	for $Re_\omega < 64$
	where $\Phi = (1+s/R_i)^2/(2+s/R_i)$	
Transition:	$C_m = 2*(s/R_i)^{0.3} / Re_\omega^{0.6}$	for $s/R_i > 0.07$
	$C_m = 2*\Psi / Re_\omega$	for $s/R_i < 0.07$
	where $\Psi = (2+s/R_i)*[1+1.4472*(1-41.3^2*(s/R_i)/Re_\omega^2)]$	
Turbulent I:	$C_m = 1.03*(s/R_i)^{0.3} / Re_\omega^{1/2}$	for $500 < Re_\omega < 10^4$
Turbulent II:	$C_m = 0.065*(s/R_i)^{0.3} / Re_\omega^{1/5}$	for $Re_\omega > 10^4$

TORCYL1 applies to a single tube or STUBE connector. It may be used to set the TORQ values, or to just calculate torques without actually setting path parameters.

TORCYL1 is flexible, but can be laborious to apply to many paths. Therefore, other utilities are available to apply the Bilgen & Boulos correlation to all of the paths within a duct macro (TORCYLM) or to all paths within a named sequence (TORCYLN). These multiple-path options set the TORQ values directly for all listed paths.

TORCYLU is available for cases in which torque is needed for user calculations, and is not to be applied directly to a tube or STUBE connector. It is also applicable for seals and rubbing passages that are not modeled using tubes or STUBE connectors (e.g., a face, ring, or labyrinth seal modeled using a LOSS or TABULAR device).

Subroutine Name: TORCYL1

Description: This routine can be used to calculate the torque applied to the fluid within a gap between a rotating and a stationary cylinder. The rotation speed (ROTR) is assumed to be fast enough to dominate the problem compared to net flows (path FR). Single-phase flow is assumed although the routines will not abort for two-phase flows.* The upstream density is used to determine the rotational Reynolds number, $Re_\omega = \rho\omega R_i s/\mu$

* A McAdam's effective viscosity approach will be used for two-phase flows.

Calling Sequence:

```
CALL TORCYL1 (MODNAM, PATHNO, TSCALE, TORQO)
```

where:

MODNAM . . . . . fluid submodel name containing the path (PATHNO) to be used for dimensions input in single quotes such as 'LOOP'

PATHNO . . . . . integer ID of the path to be used for rotor radius R_i, the length h, and rotation rate ω (R_i=max(RADI,RADJ), h=TLEN, and ω is calculated from ROTR). This path must be a tube or STUBE connector. The gap size is assumed to be s=DH/2.

TSCALE . . . . . Torque scaling factor applied to torque before returning. Specify 1.0 to return or set an unmodified torque value. The TSCALE value can also be used to model roughness, or as a parameter to assist correlation to test data.

TORQO . . . . . Output total torque on the fluid (real, units of N-m or lb_f-ft, depending on UID). This may be specified directly as TORQ for the path assuming TCF is the correct value.

Caution: This routine sets INTORQ=YES for the path, meaning that TORQ is an input and QTMK (and EFFP) are outputs.

Guidance: Should be called from FLOGIC 0 if the last argument is the TORQ value for the path (see example below).

Example:

```
CALL TORCYL1 ('loopD', 323, 1.0, torq323)
```

Subroutine Name: TORCYLM

Description: This routine is the same as TORCYL1, except (1) it is applied to all paths within a duct macro (Section 3.9.2), and (2) it can only be used in the "calculation mode" since it sets the values of TORQ directly for all macro paths.

Calling Sequence:

```
CALL TORCYLM (MODNAM, MACID, TSCALE)
```

where:

MODNAM .... fluid submodel name containing the paths for which TORQ values are to be set

MACID..... integer ID of the duct macro containing the paths. These RADI, RADJ, ROTR, TLEN, and DH values of each such path will be used to determine the TORQ values as explained above.

TSCALE .... Torque scaling factor applied to torque before returning. Specify 1.0 to return or set an unmodified torque value. The TSCALE value can also be used to model roughness, or as a parameter to assist correlation to test data.

Guidance: Should be called from FLOGIC 0. Since the TORQ value is set directly, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
CALL TORCYLM ('disZ', 31, 1.0)
```

Subroutine Name: TORCYLN

Description: This routine is the same as TORCYL1, except (1) it is applied to all paths within a list, and (2) it can only be used in the "calculation mode" since it sets the values of TORQ directly for all listed paths. The paths are listed according to their IDs: initial, total, and increment values as with the input option GENPA.

Calling Sequence:

```
CALL TORCYLN (MODNAM, N1, NUM, INC, TSCALE)
```

where:

MODNAM .... fluid submodel name containing the paths for which TORQ values are to be set

N1..... integer ID of the first tube or STUBE connector

NUM ..... Total number of paths listed (integer)

INC ..... Increment of the ID, such that the ID of the paths listed are N1, N1+INC, N1+2*INC, ... N1 + (NUM-1)*NINC.

TSCALE .... Torque scaling factor applied to torque before returning. Specify 1.0 to return or set an unmodified torque value. The TSCALE value can also be used to model roughness, or as a parameter to assist correlation to test data.

Guidance: Should be called from FLOGIC 0. Since the TORQ value is set directly, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
CALL TORCYLN ('spinHQ', 101,10,1,1.0)
```

Subroutine Name: TORCYLU

Description: This routine is the same as TORCYL1, except that it is need not reference a tube or STUBE, and requires instead the user to specify dimensions and rotational speed as arguments. It can therefore be used either for user calculations, or the results can be applied to a different connector device such as a LOSS or TABULAR. Because it does not reference a path directly, it does not set INTORQ=YES as do TORCYL1 and the other routines described above; if TORCYLU is to be applied to a path, set INTORQ=YES in FLOW DATA.

NOTE: If the purpose of calculating TORQ is to have the program calculate the dissipative heating term, QTMK, for some path, then set ROTR for the path even if there is no spinning force. Otherwise, QTMK will remain zero if ROTR=0.0, even if INTORQ=YES and TORQ is nonzero.

Even though a path is not referenced directly, thermodynamic state data is still required. This can be supplied by either naming a path of any type, whose current upstream end is to be used as the state for density and viscosity calculations, or by naming a lump directly.

Calling Sequence:

```
      CALL TORCYLU (MODNAM, NPL, GAP, RAD, HLEN, RPM,
     +   TSCALE, TORQO)
```

where:

MODNAM .... Fluid submodel name containing the lump or path to be used as thermodynamic state, in single quotes

NPL ..... ID of path whose current upstream state should be used for calculations of density and viscosity. If negative, the ID of a lump that should be used directly

GAP ..... Input gap size (in ft or m, depending on UID, real variable)

RAD ..... Input radius (in ft or m, depending on UID, real variable)

HLEN ..... Input cylinder length (in ft or m, depending on UID, real variable)

RPM ..... Input rotational speed in revolutions per minute (RPM)

TSCALE .... Torque scaling factor applied to torque before returning. Specify 1.0 to return or set an unmodified torque value. The TSCALE value can also be used to model roughness, or as a parameter to assist correlation to test data.

TORQO..... Output total torque on the fluid (real, units of N-m or lb_f-ft, depending on UID). This may be specified directly as TORQ for a path assuming TCF is the correct value.

Guidance: Should be called from FLOGIC 0 if used to set the TORQ or RVR value of a path. If TORQ is to be set for a path, make sure that the TCF values are appropriate for TORQ in units of either N-m or lb_f-ft (according to UID). (TCF defaults to the correct value.)

Example:

```
C USE LUMP UPSTREAM OF PATH 101 AS A REFERENCE, THEN APPLY TO THAT PATH
         CALL TORCYLU ('labs', 123, 1.0E-3, 3.2E-2, 12.E-3, 3000.,
     +         1.0, torq123)
```

7.11.8 FLUINT Output Routines

This section describes fluid submodel output routines. All of these routines send ASCII (plain text) data to the OUTPUT file designated in OPTIONS DATA. The following routines are available:

LMPTAB . . . . . Tabulates basic lump parameters
LMXTAB. . . . . Tabulates extra lump parameters
TTTAB. . . . . Tabulates twinned tank parameters
L2TAB . . . . . Tabulates additional twinned tank parameters
LMCTAB. . . . . Tabulates lump constituent concentrations, first 10 constituents
CNSTAB . . . . . Tabulates detailed data on each constituent
TIETAB . . . . . Tabulates tie parameters
TI2TAB . . . . . Tabulates twinned tie parameters
TIEHTPTAB . . Tabulates HTP tie parameters
PTHTAB . . . . . Tabulates path parameters
TWNTAB . . . . Tabulates twinned path parameters
TUBTAB . . . . . Tabulates tube and STUBE parameters
TB2TAB . . . . . Tabulates tube and STUBE parameters relating to spatial accelerations
ROTTAB . . . . . Tabulates tube and STUBE rotational parameters
PORTTAB . . . . Tabulates path end port locations and depth
LOSSTAB . . . . Tabulates inputs and outputs for loss-type devices
ORIFTAB . . . . Tabulates inputs and status for controlled ORIFICE (as valves)
DPTAB . . . . . Tabulates path forces in terms of pressure differences
PTCTAB . . . . . Tabulates path constituent suction action
MACINTAB . . Tabulates duct macro input parameters
MACROTAB. . Tabulates duct macro output (predicted) parameters
FTTAB. . . . . Tabulates ftie parameters
IFCTAB . . . . . Tabulates iface parameters
HTLTAB . . . . . Tabulates lump internal heat transfer parameters (UVI, ULI ..)
HTPTAB . . . . . Tabulates path internal heat transfer parameters (ULPU, UVPD ...)
GASTAB . . . . . Tabulates dissolved gas mass transfer coefficients for lumps
GASTB2 . . . . . Tabulates dissolved gas homogeneous nucleation parameters for lumps
GASATB . . . . . Tabulates dissolved gas mass transfer area data for paths
GASGTB . . . . . Tabulates dissolved gas mass transfer coefficients for paths
LMPRNT. . . . . Prints fluid lump attributes
QDPRNT . . . . . Prints fluid lump energy flow
TKPRNT . . . . . Prints fluid tank attributes
FRPRNT . . . . . Prints fluid path flow rates
TBPRNT . . . . . Prints fluid tube attributes
STPRNT . . . . . Prints fluid STUBE connector attributes
FLOMAP. . . . . Prints fluid network connectivities
LMPMAP . . . . FLOMAP for one lump
PRPMAP . . . . . Maps working fluid properties
MAPHEN . . . . Maps Henry’s constant for solute/solvent systems
UDFERR . . . . . User error routine for 6NNN fluids
YSMPWK . . . . YSMP workspace utilization report for fluid submodels

Subroutine Name: LMPTAB

Description: This routine tabulates a variety of lump parameters, including TL, PL, XL, AL, DL, HL, and QDOT (listed as "HEAT RATE*"). Two headings requiring further explanation are listed below.

"MASS RATE" — The rate of change of mass for a lump (dM/dt, see Section E.1), or the mass influx for a plenum. This should be nearly zero for a junction unless HLDLMP has been used, and should approach zero for a tank at steady-state. Thus, this can be used as a measure of how far a lump is from steady conditions.

"ENERGY RATE" — The rate of change of internal energy for a lump (dU/dt, see Section E.1), or the energy influx for a plenum. This should be nearly zero for a junction unless HLDLMP has been used, and should approach zero for a tank at steady-state. Thus, this can be used as a measure of how far a lump is from steady conditions.

Calling Sequence:

```
CALL LMPTAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: LMXTAB

Description: This routine tabulates lump parameters excluded from LMPTAB that are useful for model debugging purposes, including lump coordinates (CX, CY, CZ) and body force vector components, the number of paths attached to each lump, whether or not each lump has been held using HLDLMP or HTRLMP, and various tank attributes such as volume, mass, and compliance.

While LMXTAB can be called from any logic block, the fact that the parameters it tabulates do not often change makes it most appropriate to be called from OPERATIONS or PROCEDURE before and/or between solution routines.

Calling Sequence:

```
CALL LMXTAB ('SMN')
```

* Actually, this HEAT RATE may be different from QDOT if this lump is the outlet of a heat exchanger, as described in Section 7.11.10.3.

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: TTTAB

Description: This routine tabulates a variety of twinned tank parameters dealing with the primary interface, including surface areas (input AST, calculated CAST), heat transfer coefficients (input ULT and UVT, calculated CULT and CUVT), and mass transfer coefficients (input GTx, calculated CGTx) for each species.

Calling Sequence:

```
CALL TTTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: L2TAB

Description: This routine tabulates a variety of twinned tanks parameters, including phasic temperatures, volumes, volume fractions, effective overall quality, total added area from paths, total heat rate,* and "VDOTI", the total effective dVOL/dt of the liquid tank including motion of the interface plus COMP and VDOT applied to the pair of tanks.

Calling Sequence:

```
CALL L2TAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

* This heat rate may be different from QDOT if this lump is the outlet of a heat exchanger, as described in Section 7.11.10.3.

Subroutine Name: LMCTAB

Description: This routine tabulates lump parameters excluded from LMPTAB that are useful for multiple-constituent models: the values of XGx and XFx for the first 10 constituents in the submodel.

Calling Sequence:

```
CALL LMCTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: CNSTAB

Description: This routine tabulates detailed constituent concentration information. Whereas LMCTAB is abbreviated and lists data by lump, CNSTAB contains detailed information about all constituents, listed by constituent (and secondarily by lump). Information includes mass concentrations, partial pressures and volumes, and percentages.

Calling Sequence:

```
CALL CNSTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: TIETAB

Description: This routine tabulates a variety of tie parameters, including UA, the QTIE currently impressed on the tied lump, and the names and temperatures of the tied lump and node. For HTN, HTNS, and HTNC ties, the names of the associated paths and the corresponding area fractions are also listed.

The “2P” column lists special two-phase heat transfer status: “S” for subcooled boiling or superheated condensation, “T for transition boiling, and “F” for film boiling. In subcooled boiling and superheated condensation, the TEF will be intermediate between TL and saturation. This status can be queried in logic using the TIE2P routine described next.

Guidance: The heat rate reported in the tie may not always be equal to the UA value times the temperature difference. The reasons for this apparent discrepancy are varied, and include internal damping and other such stability enhancing measures. In general, the discrepancy does not exist upon convergence, or is apparent only for negligibly small heat rates.

Calling Sequence:

```
CALL TIETAB (’SMN’)
```

where:

`SMN` . . . . . a fluid submodel name (use the word ’ALL’ to print all fluid submodels, which is the default if the argument is omitted). Character delineators (’) are required.

Subroutine Name: TIE2P

Description: This routine provides logic block access to the information contained in the “2P”column of TIETAB outputs. This routine is provided in case the status of the boiling regime is needed for plotting or for modeling decisions.

Calling Sequence:

```
CALL TIE2P (’SMN’, ntie, ISS, ICH)
```

where:

`SMN` . . . . . a fluid submodel name (Note that use of the word ’ALL’ is NOT permitted) containing the tie to be queried. Character delineators (’) are required.

`ntie` . . . . . identifier of the tie to be queried

ISS ..... returned integer result for wall status:
0 = normal (TEF is not saturation)
1 = TEF is saturation, or nearly so: subcooled boiling or superheated condensation is active

ICH ..... returned integer result for CHF (critical heat flux) status:
0 = normal (perhaps nucleate boiling)
1 = transition boiling
2 = film boiling

See also the NTWOP output factor (Section 3.6.7.3).

Subroutine Name: TI2TAB

Description: This routine tabulates a variety of twinned tie parameters, including UA, the QTIE currently impressed on the tied twinned lumps, and the names and temperatures of the tied lump and node, and twinned tie parameters FQ (apportionment fraction), XOL and XOH (volume fractions for apportionment control), and ITAC (apportionment control flag).

Guidance: The heat rate reported in the tie may not always be equal to the UA value times the temperature difference. The reasons for this apparent discrepancy are varied, and include internal damping and other such stability enhancing measures. In general, the discrepancy does not exist upon convergence, or is apparent only for negligibly small heat rates.

Calling Sequence:

```
CALL TI2TAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: TIEHTPTAB

Description: This routine tabulates information specific to HTP ties.

Calling Sequence:

```
CALL TIEHTPTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: PTHTAB

Description: This routine tabulates a variety of path parameters, including upstream and downstream lumps, FR, DUPI, DUPJ, STAT, and pressure drop (negative for a pressure rise). Four headings requiring further explanation are listed below.

"XL UPSTRM" — The quality of the upstream lump as viewed by this path, taking into account any phase suction action. This is the same value returned by the utility routine XTRACT.

"REYNOLDS" — (Tubes and STUBEs only.) For single-phase flow, prints the current Reynolds number for the current upstream end.

"FR/FRC" — (Paths with positive AF.) Prints an estimate of the Mach number at the throat: |FR/FRC| if MCH is nonzero. *This value will often differ from XMA, which is calculated at the upstream end.* If the path is choked, a "c" appears after the Mach number. If an asterisk ("*") appears after the Mach number, this means the AFTH is less than the AF, yet MCH=0, so AFTH has been used to calculate a crude estimate of the throat Mach number assuming an incompressible "expansion." This will normally underestimate the throat Mach number, so the asterisk is a reminder that choked flow detection (Section 3.18) should be applied to this path to be safe (i.e., nonzero MCH). If instead a lower case "x" appears after the Mach number, this means the upstream lump uses LSTAT=STAG and the printed Mach number will not have taken into account the expansion from a zero-velocity state, and that again choking should be enabled for this path. If the value is negative, either AF has not been input for this path, or relevant fluid properties (entropy, compliance, etc.) are missing. *This column is largely intended to provide a preliminary estimate of the choking; refer to CHOKETAB for a complete breakdown.*

"REGIME" — (Tubes and STUBEs only.) For single-phase flow at the current upstream end, prints "1 PHASE." If two-phase flow is present, prints the current flow regime at the current upstream end if IPDC=6, else UNKNOWN.

When paths are twinned and active, the primary twin ID will be listed with an 'L' suffix, and the secondary twin ID with a 'V' suffix. In this case, the Reynolds number for each twin corresponds to the superficial velocity (i.e., as if this were the only phase in the duct. Otherwise, for twins in the homogeneous mode only the primary twin will be listed with a suffix of 'H' after the ID.

The PTHTAB routine lists the current STAT flag. Species-specific suction flags are only listed if no phase suction has been set. If more than one species has been excluded and more than one species has been allowed to pass, or if the specifications on the downstream end are too different

from those on the upstream end, then the program cannot represent the current status simply (e.g., SPG, NRSPF, DSPX, etc.). In such cases, the program will simply report "(sp)" to indicate that species specific suction options are active. See also PTCTAB.

Calling Sequence:

```
CALL PTHTAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: CHOKETAB

Description: This routine tabulates path data pertaining to choking (sonic flow limits) including throat states, and expansions to those throat states from the path inlets. The routine lists values such as MCH, HCH, FRC, FR/FRC, XMA, and the throat state PLTH, TLTH, and XLTH.

CHOKETAB lists whether the current upstream lump is stagnant (LSTAT=STAG) or not (LSTAT=NORM), and whether an area reduction from inlet to throat exists via the ratio AFTH/ $AF_{upstream}$ (abbreviated as "AFU" in the header). If either the upstream is stagnant or an area contraction exists, then an isentropic expansion from the inlet state is usually be included in the throat state calculation. The presence of such an expansion is indicated by a "Y" in the "EXP?" column, whereas "N" will appear if no such expansion currently exists.

The input MCH as well as the "used" or effective MCH are both tabulated. For example, if the input MCH were -2 but equilibrium expansion was calculated, the "used" MCH would be listed as positive 2. Similarly, if MCH=-3 were input but the working fluid had no phase change or if no expansion existed (as indicated by "N" in the "EXP?" column), then the "used" MCH might be 2.

For comparison, both FR/FRC and XMA are presented. FR/FRC is the ratio of current to critical flow rate (also listed in PTHTAB if MCH is nonzero), whereas XMA is the Mach number at the inlet.

The XLTH may also differ from the suction quality, "XL UP," which is also listed. XLTH is calculated at the start of the last time step or interval, whereas XL UP is calculated at the time CHOKETAB is called. More significant differences between the two columns can indicate flashing or the effects of equilibrium expansions (i.e., with phase change).

For MCH=-3 and PLTH equal to the downstream pressure ("Y" in the "EXP?" column), XLTH may not be the same as a downstream junction XL. XLTH is calculated using an isentropic assumption, whereas XL is calculated using an adiabatic (isenthalpic) assumption, perhaps with considerable irrecoverable losses.

If MCH=0, CHOKETAB makes an attempt to estimate the FR/FRC ratio (see PTHTAB description above for estimation methods), and provides a warning (via an MCH value of "0!") if this rough estimate exceeds unity. If this warning is evident, the decision to turn off choking should be revisited.

The ratio FR/FRC might also be listed as "(NO AF)" for paths lacking a flow area, or "(NO SOS)" for user-defined fluids lacking enough property information to calculate speeds of sound. In either of these two cases, the remaining columns can be disregarded.

Calling Sequence:

```
CALL CHOKETAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: TWNTAB

Description: This routine tabulates a variety of path parameters for twinned paths, including upstream and downstream lumps, FR, DUPI, DUPJ and STAT. TWNTAB is an alternate way of interpreting the same data presented for twins in PTHTAB, and is intended to be used as a companion to PTHTAB when twins are used. Five headings requiring further explanation are listed below.

"XL FLOW" — The quality of the flow through the passage shared by the pair of twins, calculated by $|FR_v|$ divided by the quantity $|FR_v| + |FR_L|$.

"FR TOTAL" — The total mass flow through the passage shared by the pair of twins, calculated as $FR_v + FR_L$.

"REYNOLDS" — Prints the current Reynolds number for the current upstream end assuming combined properties (summed flow, estimated viscosity, etc.)

"MACH" — Prints an estimate of the Mach number at the upstream end. This may or may not correspond to the limit condition. If the path is choked, a "c" appears after the Mach number. If an asterisk ("*") appears after the Mach number, this means the AFTH is less than the AF, and AFTH has been used to calculate a crude estimate of the throat Mach number assuming an incompressible "expansion." This will normally underestimate the throat Mach number, so the asterisk is a reminder that choked flow detection (Section 3.18) should be applied to this path to be safe (i.e., nonzero MCH). If instead a lower case "x" appears after the Mach number, this means the upstream lump uses LSTAT=STAG and the printed Mach number will not have taken into account the expansion from a zero-velocity state, and that again choking should be enabled for this path.

"REGIME" — For single-phase flow at the current upstream end, prints "1 PHASE." If two-phase flow is present, prints the current flow regime at the current upstream end if IPDC=6, else UNKNOWN.

When paths are twinned and active, the primary twin ID will be listed with an 'L' suffix, and the secondary twin ID with a 'V' suffix. In this case, the flow regimes are printed for the pair if IPDC=6. (Only the upstream end will appear even if UPF=0.5, since twins internally use UPF=1.0 when active.) Otherwise, for twins in the homogeneous mode the primary twin will be listed with a suffix of 'H' after the ID, and the secondary will appear with an 'X' suffix.

Calling Sequence:

```
CALL TWNTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels). Character delineators (') are required.

Subroutine Name: TUBTAB

Description: This routine tabulates tube and STUBE simulation parameters: DH, AF, TLEN, AC_{eff},* HC, FC, FPOW, UPF, IPDC, FK, and WRF. TUBTAB also lists a column designated "IN." If an asterisk appears under this column, it means that this path is an inlet from a stagnant lump, and that an internal acceleration term (equivalent to a K-factor of 1.0) has been added automatically: a total to static pressure "conversion.".

Calling Sequence:

```
CALL TUBTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

* AC_{eff} is the effective AC taking into account extra constant and linearized terms that are normally hidden from the user for some paths.

Subroutine Name: TB2TAB

Description: This routine tabulates tube and STUBE simulation parameters: DH, AF, AFI, AFJ, AFTH, $AC_{eff,}$* up and downstream densities, and the pressure term resulting from the AC factor (positive for acceleration the defined positive flow direction). Note that the "DP ACCEL" term is calculated as AC_{eff}*FR^2 for isolated paths, but for paths within duct macros the underlying formulation is more complicated, involving more terms than just AC.

Calling Sequence:

```
CALL TB2TAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels). Character delineators (') are required.

Subroutine Name: ROTTAB

Description: This routine tabulates path simulation parameters relative to rotation options: ROTR, RVR (tubes and STUBE connectors only), RADI, RADJ, VAI, VAJ, VXI, VXJ, EFFP, TORQ, and QTMK.† It also lists the net pressure force due to spinning.

Calling Sequence:

```
CALL ROTTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: PORTTAB

Description: This routine tabulates path port information, including port (path end) coordinates (CXI, CYI, CZI or CXJ, CYJ, CZJ) and current depth (BFI or BFJ) relative to the liquid vapor surface, which is also listed.

---

* AC_{eff} is the effective AC taking into account extra constant and linearized terms that are normally hidden from the user for some paths.

† The last three parameters for PUMP connectors (EFFP, TORQ, and QTMK) will reflect the PUMP inputs, and not the rotation options of the same name which they supersede. The same is true for the last two parameters (TORQ and QTMK) for TABULAR connectors if those options were specified in table form.

The fill fraction (“FILL”) of the associated endpoint lump is listed. This is equal to 1-AL for a homogeneous tank, or the liquid volume ratio for an active pair of twinned tanks, in which case “(TW)” is listed as an advisory.

Body force vector components are also listed. This port location and depth information is useful when viewed with LMXTAB output.

Note that the update of the endpoint lump location (CX, CY, CZ) to reflect a point on the current liquid/vapor plane is normally the responsibility of the user. However, if the endpoint lump represents an active segment in a flat front model (Section 3.27.5.4), this location of the liquid/vapor position is calculated automatically within that line segment, and the information displayed by PORTTAB will no longer correspond to the lump end location. If this is the case, “(FF)” is listed next to the liquid/vapor point coordinates as an advisory.

Calling Sequence:

```
CALL PORTTAB ('SMN')
```

where:

`SMN.....` a fluid submodel name (use the word ’ALL’ to print all fluid submodels, which is the default if the argument is omitted). Character delineators (’) are required.

Subroutine Name: LOSSTAB

Description: This routine tabulates inputs and outputs for LOSS-type paths.

Calling Sequence:

```
CALL LOSSTAB ('SMN')
```

where:

`SMN .....` a fluid submodel name (use the word ’ALL’ to print all fluid submodels, which is the default if the argument is omitted). Character delineators (’) are required.

Subroutine Name: ORIFTAB

Description: This routine tabulates inputs and status for controlled ORIFICE devices, using nonzero MODA (“orifices as valves,” Section 3.5.12.2).

Calling Sequence:

```
CALL ORIFTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: DPTAB and DP2TAB

Description: This routine tabulates path forces (FK, FC, AC, etc.) in units of pressure difference. DP2TAB is for twinned paths. Note HCBF is the pressure force caused by body forces and rotations.

Calling Sequence:

```
CALL DPTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: MACINTAB

Description: This routine tabulates a flow-wise listing of **input** parameters for both lumps and paths within a duct macro. End lumps that are not part of the macro are listed for convenience, with parentheses bracketing their IDs to distinguish them from lumps owned by the macros.

Note that the volume of junctions, if input, is listed inside of parentheses for reference. The volume of paths, calculated as TLEN*AF, is always listed in parentheses for reference: for comparison with adjacent lump volumes. (If AFI and AFJ are used, the above "path volume" is approximate.)

The macro ID listed in MACINTAB might be automatically generated by Sinaps or FloCAD: it might not be recognizable as a Pipe ID. In other words, if the ID does not correspond to any user Pipe, it was automatically generated. (SINDA/FLUINT cannot distinguish whether a GUI or a text input file was used.) Automatically generated duct macros can be identified using coloring options in both Sinaps and FloCAD, and their creation can be controlled in both programs.

Calling Sequence:

```
CALL MACINTAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: MACROTAB

Description: This routine tabulates a flow-wise listing of **output** parameters for both lumps* and paths within a duct macro. End lumps that are not part of the macro are listed for convenience, with parentheses bracketing their IDs to distinguish them from lumps owned by the macros.

The "dP Tot" value is the pressure of the upstream lump minus the downstream lump, and is normally a positive value if frictional resistance dominates. This is opposite in sign convention (in order to be consistent with the same value in other tabulation routines) from "dP Accel," the spatial acceleration term. The "dP Accel" term is positive if the flow is being pushed in the positive flow rate direction by this term (accelerated), and negative if it is being decelerated by this term in the positive flow direction.

The macro ID listed in MACROTAB might be automatically generated by Sinaps or FloCAD: it might not be recognizable as a Pipe ID. In other words, if the ID does not correspond to any user Pipe, it was automatically generated. (SINDA/FLUINT cannot distinguish whether a GUI or a text input file was used.) Automatically generated duct macros can be identified using coloring options in both Sinaps and FloCAD, and their creation can be controlled in both programs.

Calling Sequence:

```
CALL MACROTAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

* The tabulated heat rate for a lump may be different from QDOT if this lump is the outlet of a heat exchanger, as described in Section 7.11.10.3.

Subroutine Name: FTTAB

Description: This routine tabulates parameters relating to ftie performance and status.

Calling Sequence:

```
CALL FTTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: IFCTAB

Description: This routine tabulates parameters relating to iface performance and status.

Calling Sequence:

```
CALL IFCTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: HTLTAB

Description: For each tank or junction, HTLTAB tabulates heat transfer coefficients (ULI, UVI, CULI, CUVI).

Calling Sequence:

```
CALL HTLTAB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: HTPTAB

Description: For each tube and STUBE connector, HTPTAB tabulates heat transfer coefficients (ULPU, ULPD, UVPU, UVPD, etc.).

Calling Sequence:

```
CALL HTPTAB ('SMN')
```

where:

SMN..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: GASTAB

Description: For each tank and junction, and for each dissolved substance, GASTAB reports the values of PH, the mass transfer coefficients (GL, CGL, and the total G*A term including the GD and GU terms of attached paths), FRD, FRNCV, and HEN.

Calling Sequence:

```
CALL GASTAB ('SMN')
```

where:

SMN ..... a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: GASTB2

Description: For each tank and junction, and for each dissolved substance, GASTB2 reports the values of IDGC, PH, ZR, ZJ, FRH, ASL, and CASL.

Calling Sequence:

```
CALL GASTB2 ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: GASATB

Description: For each tube and STUBE connector, GASATB reports the associated lumps, the surface area (ASPU, ASPD, CASPU, CASPD), and the apportionment to each lump (FDAS, FUAS).

Calling Sequence:

```
CALL GASATB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: GASGTB

Description: For each tube and STUBE connector, and for each dissolved substance, GASGTB reports the associated lumps, the apportionment (FDAS, FUAS), and the mass transfer coefficients (GU, GD, CGU, CGD).

Calling Sequence:

```
CALL GASGTB ('SMN')
```

where:

SMN . . . . . a fluid submodel name (use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted). Character delineators (') are required.

Subroutine Name: LMPRNT

Description: This routine prints out a variety of lump thermodynamic state variables. A print flag controls the printing of TL, PL, XL, HL and DL for a submodel.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL LMPRNT ('SMN', 'PRFLG')
```

where:

SMN ..... a fluid submodel name; (use the word 'ALL' to print all fluid submodels, which is the default if both arguments are omitted). Character delineators (') are required.

PRFLG..... A character or combination of characters that define the variables to be printed. Valid entries are:
'T' to print TL
'P' to print PL
'X' to print XL
'H' to print HL
'D' to print DL
Or the word 'ALL', which is the default if both arguments are omitted. Character delineators (') are required.

Subroutine Name: QDPRNT

Description: This subroutine prints out the value of QDOT for lumps.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL QDPRNT ('SMN')
```

where:

SMN ..... is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: TKPRNT

Description: This subroutine prints out the values of mass, volume, rate of change of volume (growth/shrink rate), and compliance factors for tanks.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL TKPRNT ('SMN')
```

where:

SMN . . . . . is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: FRPRNT

Description: This subroutine prints out flow rates for paths along with path type and connector device names.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL FRPRNT ('SMN')
```

where:

SMN. . . . . is a fluid submodel name or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Names: TBPRNT, STPRNT

Description: These subroutines print out the values of DH, AF and TLEN for tubes or STUBE connectors.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL TBPRNT ('SMN')
CALL STPRNT ('SMN')
```

where:

SMN . . . . . is a fluid submodel name or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: PUMPMAP

Description: This subroutine prints out approximately a page of information about each PUMP device within a submodel, including the current operating point, inlet and outlet states (including ISTP effects) and an echo of head-flow and power-efficiency inputs in a compact form. Positively sloped regions of the head-flow map, corresponding to negative DGDH, are excluded from the echoed map since these points and unstable and will not normally persist.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL PUMPMAP ('SMN')
```

where:

SMN . . . . . is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: TURBMAP

Description: This subroutine prints out approximately a page of information about each TURBINE device within a submodel, including the current operating point, inlet and outlet states (including ISTP effects) and an echo of pressure-flow and power-efficiency inputs in a compact form.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL TURBMAP ('SMN')
```

where:

SMN . . . . . is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: COMPRMAP

Description: This subroutine prints out approximately a page of information about each COMPRESS device within a submodel, including the current operating point, inlet and outlet states (including ISTP effects) and an echo of pressure-flow and power-efficiency inputs in a compact form. Positively sloped regions of the pressure-flow map, corresponding to negative DGDH, are excluded from the echoed map since these points and unstable and will not normally persist.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL COMPRMAP ('SMN')
```

where:

SMN . . . . . is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: COMPPDMAP

Description: This subroutine prints out approximately a page of information about each COMPPD device within a submodel, including the current operating point, inlet and outlet states (including ISTP effects) and an echo of pressure-flow and power-efficiency inputs in a compact form.

Restrictions: A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL COMPPDMAP ('SMN')
```

where:

SMN ..... is a fluid submodel name, or use the word 'ALL' to print all fluid submodels, which is the default if the argument is omitted. Character delineators (') are required.

Subroutine Name: FLOMAP

Description: This subroutine prints out the connectivity network for a fluid submodel. It prints out the mass flow rate and energy rate associated with a path (positive *into* each lump). It also prints out the temperature and quality of the adjoining lump, along with the difference in pressures (adjoining minus current). After the path information is printed for each lump FLOMAP prints out the total mass and energy storage rate for the lump. FLOMAP may be used to map a single model or all fluid submodels. If the QMAP file name is set in OPTIONS DATA, FLOMAP will write to this file. Output can optionally be routed to the OUTPUT file, although this copy can be suppressed if a QMAP file is named.

Restrictions and Guidance: Because of the volume of output, FLOMAP calls should only appear in OPERATIONS or PROCEDURE and should never appear in any VARIABLES or FLOGIC block. A fluid submodel name must be specified and should agree with a submodel name appearing on a BUILDF card.

Calling Sequence:

```
CALL FLOMAP (SMN, NPRNT)
```

where:

SMN ..... a fluid submodel name in quotes or 'ALL' for all fluid submodels (the default if this argument is omitted)

NPRNT..... OUTPUT Print / No Print flag:
0: print to processor OUTPUT file (default if argument is omitted)
1: suppress printing to OUTPUT file (still writes to QMAP file if named in OPTIONS DATA)

Example:

```
CALL FLOMAP('ALL',0)
```

Subroutine Name: LMPMAP

Description: This subroutine performs the same function as FLOMAP, but maps only one requested lump to help reduce outputs.

Calling Sequence:

```
CALL LMPMAP (SMN, LUMP, NPRNT)
```

where:

SMN . . . . . a fluid submodel name in quotes
LUMP. . . . . ID of lump to be mapped
NPRNT . . . . . OUTPUT Print / No Print flag:
0: print to processor OUTPUT file (default if argument is missing)
1: suppress printing to OUTPUT file (still writes to QMAP file if named in OPTIONS DATA)

Example:

```
CALL LMPMAP('WETNWILD',131,0)
```

Subroutine Name: PRPMAP

Description: This output routine is available to aid in the development of a fluid description input in an FPROP DATA block. This routine should be called once from OPERATIONS. It produces tabulated properties for any fluid over the valid range of temperatures and pressures. The property information is written to the OUTPUT file.

Calling Sequence:

```
CALL PRPMAP (P, T, smn.FI or FIn )
```

where:

P. . . . . Pressure. For 8000 and 9000 fluids, this is the pressure at which tables will be produced. For other fluids, this is the *maximum* pressure for which data will be produced if it is smaller than PGMAX.
T. . . . . Maximum temperature at which data will be produced if it is smaller than TGMAX (or TLMAX for 9000 fluids).
smn . . . . . submodel containing desired fluid
n. . . . . fluid ID number

Restrictions and Guidance: The smn.FI or FIn format choice is the same as that of other property routines, but that *P and T are assumed to be in relative (not absolute), user units*, whereas they must be in absolute units for other property calls. All output is in the unit system convention chosen by the user-input values of UID, PATMOS, and ABSZRO.

Examples:

```
CALL PRPMAP(100.0, 500.0, FI9718)
CALL PRPMAP(1.0D30, 1.0E30, SCHMIT.FI)
```

Subroutine Name: MAPHEN

Description: This output routine is available to aid in the development of a soluble fluid description input in a MIXTURE DATA block (Section 3.22). This routine should be called once from OPERATIONS. It produces tabulated Henry's constants for any binary solvent/solute system over the valid range of temperatures and pressures. The effects of other solvents are neglected -- use the HEN constant (Section 3.23.3) to find the actual effective constant including the effects of other solvents. The property information is written to the OUTPUT file.

Calling Sequence:

```
CALL MAPHEN (idg, ids, Pmax, Tmax)
```

where:

`idg` ..... the 4 digit identifier of the solute gas: 8nnn.
`ids` ..... the identifier of the solvent (a 6000, 7000, 9000 series or library fluid).
`Pmax` ..... Maximum *partial* pressure. This is the maximum pressure for which data will be produced if it is smaller than the allowable range limit for the fluid.
`Tmax` ..... Maximum temperature. This is the maximum temperature for which data will be produced if it is smaller than the allowable range limit for the fluid.

Restrictions and Guidance: P and T are assumed to be in relative (not absolute), user units, whereas they must be in absolute units for other property calls. All output is in the unit system convention chosen by the user-input values of UID in CONTROL DATA, GLOBAL: either Pa or psia (pressure over unitless *mass* fraction). These units may differ from the units used in MIXTURE DATA.

Example:

```
CALL MAPHEN(8029, 717, 1.0D6, 500.0)
```

Subroutine Name: UDFERR

Description: This simple routine is provided to allow the user to write warnings to the OUTPUT file. The primary purpose is to support 6000 series user-defined fluids. The calling sequence is:

```
CALL UDFERR (ROUT, ID, LINE)
```

where:

ROUT.....Routine name where error occurred, character variable
ID .....Fluid ID, integer
LINE.....Status or message line, character variable.

This routine prints the following message to the output file:

*** WARNING *** ILLEGAL OPERATION IN PROPERTY ROUTINE *(ROUT)* *(LINE)* ; FID: *(ID)*, TYPE: ADVANCED TWO-PHASE

Subroutine Name: YSMPWK

Description: This output routine is available to aid in the customizing of the YSMP (matrix inversion package) workspace allocation.* YSMPWK simply reports on the actual usage (as experienced thus far in the run) of workspace by each fluid submodel. It is called automatically by the processor in the event of an abort due to insufficient allocation, and is available for direct user calls as well.

The exact amount workspace required cannot be known beforehand, and insufficient allocation will result in an aborted run. The workspace allocation for each fluid submodel is normally calculated automatically by the preprocessor, which employs a conservative estimate. The user may override this estimate by specifying the allocation with the "NWORK=I" keyword within the HEADER FLOW DATA subblock. *Unless directed to do so by the processor abort message, the user should avoid employing this option.* Refer to the caution below.

Calling Sequence:

```
CALL YSMPWK
```

Guidance: Since the utilization of workspace is cumulative, *place the call to YSMPWK at the end of OPERATIONS*. Calling this routine from other locations may result in lower than actual reported usages.

Caution: Reducing the allocation of workspace will reduce the memory requirements of the processor and may result in execution speed increases on some machines. However, such reductions increase the likelihood of a processor abort. The user is forewarned that actual utilization is *very* sensitive to slight modeling changes, to differences in cases run, and to the solution routine employed.

Refer to YSMPWS for analogous reporting of thermal submodels.

* On f90/f95 versions (denoted with the last character L and D on PCs and U on Suns), workspace is allocated automatically and no user involvement is required: users of those versions can ignore this routine.

7.11.9 FLUINT Simulation Routines

This section describes the following simulation routines:

COMPRS . . . . . Compressors and other turbomachinery
NCGBUB . . . . . Stationary Noncondensible Gas Bubble
TANK_LINKER. . . . . IFACE-like connection between submodels
WETWICK. . . . . Wet wick conductance correlation
PLAWICK . . . . . Wet wick planar conductivity calculation
CYLWICK . . . . . Wet wick cylindrical conductivity calculation
WCHOKE. . . . . Choked flow (sonic limit) warning aid
COMPLQ . . . . . Add liquid compressibility to tanks; waterhammer simulation aid
WAVLIM . . . . . Maximum time step for waterhammer and acoustic waves
YTKALG . . . . . Gardel's correlation for wye-tee (diverge/merge) flow losses
YTKALI . . . . . Idelchik's correlation for wye-tee (diverge/merge) flow losses
YTKONV . . . . . Wye-tee K-factor conversion utility
BALTPL . . . . . Two-phase loop energy balancing utility
EQRATE. . . . . Chemical reaction rate utility
TankCalc60. . . . . Geometric calculations for a partially filled vessel

Most of these routines perform adjunct solutions, helping to calculate FLUINT network parameters such as COMP, VDOT, GK, HK, etc. for tanks, NULL connectors, etc. input by the user.

7.11.9.1 Compressors and Other Turbomachinery

All paths are inherently adiabatic and therefore represent isenthalpic processes: the energy addition associated with pumping is neglected. While this is normally acceptable for liquid pumps, fans, and circulators, it is nonsensical for compressors and turbines. Thus, a simulation of such turbomachinery requires the attributes of both paths and lumps.

Subroutine Name: COMPRS

Description: This routine allows the user to simulate generalized compressors. The required inputs include a total input power* and an isentropic efficiency η.† The same routine may be used to simulate turbines, although its usage in that context is more limited since the usual analytic approach is to calculate the output power, rather than to define it as an input.

NOTE: This routine is largely obsolete with the advent of the TURBINE, COMPRESS, and COMPPD connector devices (Section 3.5).

* Recall that QL is the lump input parameter, which COMPRS uses as net shaft input power. QDOT is equal to QL plus the effects of any ties and fties. The latter is assumed to not affect compressor operation, only outlet temperatures.

† See Sample Problem G for alternate methodologies when the power is not known but the speed and the volumetric efficiency *are* known, perhaps as performance maps.

To use this routine:

1) Build a NULL connector to act as the compressor. (A fake initial GK value must be provided in FLOW DATA to satisfy the input requirements of NULL connectors.)

2) *Apply QL on the defined downstream lump as the power input that drives the compressor*; it may be changed as needed.

3a) Call COMPRS in FLOGIC 0. When called from this location, the resulting NULL flow rate driving factor HK may be damped.

3b) Or, call COMPRS in FLOGIC 1. When called from this location, the resulting NULL flow rate driving factor HK will be undamped.

Using a tank as the downstream lump can cause the flow rates to differ transiently from the equilibrium value since the tank cannot react instantly to the actions of the compressor. Internal damping to prevent flow rate jumps or oscillations (if called from FLOGIC 0) and use of a junction may cause temperature excursion warnings since the flow rate may be occasionally prevented from rising or falling to its desired level instantly.

Calling Sequence:

```
CALL COMPRS (FSMN, NULL, EFF)
```

where:

`FSMN`. . . . . fluid submodel name, in single quotes
`NULL`. . . . . the ID of the NULL connector representing compressor
`EFF` . . . . . Isentropic efficiency η; less than 1.0 for a compressor, or greater than one ($1/\eta$) for a turbine.

Zero QL causes the NULL to act like a shut valve to be consistent with the governing equations. Negative QL is equivalent to a turbine. In this case, EFF should be greater than or equal to 1.0: it should be set to the inverse of the turbine η.

Restrictions: COMPRS should not be used with liquids or two-phase flow. It can be used with library fluids, and 7000 and 8000 series user fluids. To use with 6000 fluids, the entropy property routine VS must be defined in the FPROP block. If two-phase flow appears at the inlet, a warning will be produced and the simulation will attempt to continue. However, if the heat input is insufficient to result in vapor at the outlet, property routine errors and/or an abort may result.

7.11.9.2 NCGBUB: Noncondensible Gas Bubble Simulation

The following routine is intended to simulate a noncondensible gas (NCG) bubble trapped within a subcooled liquid control volume. The gas is assumed to be a perfect gas in thermal equilibrium with the liquid in the control volume, and is assumed to not dissolve in it. The gas bubble may grow or shrink according to its partial pressure, displacing liquid in the process. The gas itself is not modeled by FLUINT, merely the effects of the gas on the rest of the control volume, which is modeled by a FLUINT tank.

As the subcooling diminishes, the gas volume will grow and the liquid volume will shrink. The minimum gas volume is 0.1% of the total. Dissolving of gas and subsequent gas evolution is neglected, as is the mass and energy associated with the working fluid vapor in the void and the related evaporation and condensation. The gas bubble is assumed to be trapped within the control volume by gravity, swirling, or wicks.

NCGBUB is convenient in certain cases, but is largely obsolete because more complete and powerful modeling is available in the form of the multiple-constituent flow capabilities (Section 3.19) and interfaces (Section 3.8) and twinned tanks (Section 3.25).

Subroutine Name: NCGBUB

Description: Simulates a trapped (stationary) NCG bubble within a subcooled control volume, whose liquid portion is modeled by a tank.

To use this routine:

1. Build one tank to represent the liquid in the control volume.
2. Before calling NCGBUB or transient routines, make sure that STEADY has been called or that the initial conditions are correct (i.e., that the liquid is subcooled, and that the volume corresponds to the total minus that which would be occupied by gas).
3. Call NCGBUB in FLOGIC 1, according to the formats specified below.

In STEADY, the tank may be held by HLDLMP. At the completion of STEADY, the volume of the tank will have been adjusted according to its temperature and pressure as needed to provide consistent initial conditions for later solution routines.

Restrictions: Must be called in FLOGIC 1. Tank must be subcooled. Zero gas mass is not allowed. If no gas is present, set VDOT=COMP=0. and do not call NCGBUB. Otherwise, the user should not manipulate tank parameters (especially VDOT, COMP, and QL) *after* the call to NCGBUB.

Calling Sequence:

```
CALL NCGBUB(MODEL, NLIQ, EMARR, VTOT)
```

where:

MODEL . . . . . Fluid submodel name containing the tank, in single quotes
NLIQ. . . . . User ID of the liquid tank
EMARR . . . . . Amount of gas present, expressed in terms of the mass times the gas constant: mR from the perfect gas relation PV=mRT.
VTOT. . . . . Total volume of the control volume (i.e., the liquid tank volume plus the fictitious gas void volume).

In metric units, EMARR has units of J/K. The universal gas constant is 8314 J/kmol-K. EMARR is therefore the mass of NCG in kg, times the universal constant, divided by the molecular weight (kg/kmol). Or, input EMARR as the number of kmol times the universal constant.

In English units, EMARR has units of psia-ft^3/R. The universal gas constant is 1545 ft-lb_f/lbmol-R. EMARR is therefore the mass of NCG in lb_m, divided by 144.0, times the universal constant, divided by the molecular weight (lb_m/lbmol). Or, input EMARR as the number of lbmol times the universal constant, divided by 144.0.

Example:

```
CALL NCGBUB('RUBY', 100, 1.0E-12*8314.,1000.E-6)
```

In the above example, assuming metric units, one liter of control volume contains 1.0E-12 kmol of noncondensible gas. Remember the volume of tank 10 will always be less than one liter. To initialize the tank volume before calling NCGBUB or STEADY, the following lines of logic might apply:

```
 TL10 = TL10 - ABSZRO
 VTEST= VTOT - EMARR*TL10
+     /(PL10-VPS(TL10,RUBY.FI)-PATMOS)
 TL10 = TL10 + ABSZRO
 CALL CHGVOL('RUBY',10,VTEST)
```

To model gas generation, transport, evolution, dissolving, etc., EMARR may be varied, but changes should be very gradual--preferably dependent on time, temperature, or volume.

7.11.9.3 IFACE-like Utility for Linking Tanks

A restriction on ifaces (Section 3.8) is that they must attach to two tanks within the same fluid submodel. Another limitation is that they cannot include the effects of friction. These two issues are addressed by the TANK_LINKER simulation routine, which creates an iface-like linkage between the two tanks by manipulating their COMP and VDOT terms.

SPRING and FLAT ifaces are the most common. Like all ifaces, they both feature an optional mass term, but neither offers a frictional resistance term. SPRING ifaces can be used to model bellows accumulators, and FLAT ifaces can be used to keep the pressures in the two tanks equal by shifting the volume boundary. FLAT ifaces can represent a liquid/vapor interface or a weak elastomeric bladder accumulator, or even a virtual subdivision of the control volume.

TANK_LINKER provides a user-callable utility that helps fulfill these needs: it offers a way to link any two FLUINT tanks, whether they are in the same submodel or not, by a pressure-volume relationship during a transient. This relationship is similar to that of either a FLAT or SPRING iface. TANK_LINKER features an optional frictional term. It also includes a mass term, but unlike an iface, this inertial term is *not* optional.

Ifaces operate directly within the solution matrix, and their effects are included directly in the implicit solution. They are therefore smooth and stable, and able to take larger time steps.

Conversely, TANK_LINKER is an “adjunct solution,” as are PID controller simulations and co-solved ODEs. This means that simulation effects are predicted as best they can be at the beginning of a time step, and any differences between predicted and actual network response are folded into the next solution as error terms. Smaller time steps can be expected when using TANK_LINKER than when using ifaces. These smaller time steps are required to keep the error terms small and the response stable.

To help avoid the need for these small time steps, the user should consider adding more mass or friction than is physically justified ... perhaps orders of magnitude more. A significant amount of mass and friction can be added before the characteristic time scale of the TANK_LINKER response becomes so high that the effects are noticeable. Otherwise, if friction and mass terms are realistic but very small, the program may focus on extremely small time scale events that are only relevant if the dynamic response of the tank interface itself was the main purpose for the analysis. Of course, the user should only accept augmented mass and friction values *after* verifying that the results have not substantially changed.

Subroutine Name: TANK_LINKER

Description: If a bellows accumulator contains fluid from one submodel on one side of the bellows, and fluid from another submodel on the other side of the bellows, then TANK_LINKER can be used to logically link the pressures and volumes of otherwise independent tanks. This routine is also applicable to elastomeric accumulators, pistons, and servo actuated valves that require a flow solution on both sides of the valve diaphragm.

In fact, there need be no physical object between the two fluid volumes: TANK_LINKER could be used to model a stationary (trapped) bubble in one tank, and the liquid that surrounds that bubble in another tank ... similar to a FLAT iface. (Note that neither the heat nor mass transfer between the two tanks is modeled by TANK_LINKER.)

There are costs and limitations associated with trying to combine species into one fluid submodel as a working fluid mixture, yet in many cases the two fluids never actually mix (e.g., they are only encountered on opposite sides of a piston, bladder, or bellows). By tracking fewer species in each submodel, TANK_LINKER can increase modeling efficiency by simplifying each mixture (or perhaps eliminating the need to use mixtures altogether). If the two fluids cannot be mixed in a single submodel, TANK_LINKER might *enable* such modeling. For example, one side of the link might have one condensible species, while the other side contains a different condensible species. Since these two species cannot be combined into a single fluid submodel (such that an iface could be used), TANK_LINKER provides a means to create such a model using separate fluid submodels.

The two tanks that will share a common boundary will be referred to as Tank A and Tank B, using the same scheme employed by ifaces (Section 3.8). Tank A can actually be the primary tank in a pair of twinned tanks (Section 3.25), as can Tank B.

The user specifies the total volume (VTOT) of both tanks, such that if one tank grows the other will shrink. The user also specifies the upper and lower limits of volume on Tank A (VHI, VLO respectively).

If a bellows accumulator or strong elastomeric bladder exists, a spring-force relationship can be set using the inputs DPDV and VZRO, mimicking a SPRING iface. To mimic a FLAT iface (perhaps as needed to model a horizontal piston), specify DPDV=0.0 (in which case VZRO is ignored).

The mass of the bellows, bladder, or piston is specified as the parameter MASS, which cannot be zero and should not be underestimated either (i.e., include extra mass if needed for longer time steps, including at the very least “added mass” from the fluid masses in Tanks A and B). Note that this input parameter is not the same as the iface EMA parameter, which is the ratio of mass to effective (cross sectional) area squared. Instead, when using TANK_LINKER the effective area across which the pressure forces act is input separately as AEFF.

One reason that this area needs to be specified separately is that an optional frictional term exists: F_VEL, the ratio of linear force to the velocity it opposes.This value can be zero (frictionless).

Given the mass and all the forces acting on the boundary between the two tanks (e.g., bellows), TANK_LINKER creates, updates, and monitors a second-order ODE for the equation tracking the linear motion of this “point” mass. The user provides an ID for this ODE (via the parameter ID_ODE) and can watch it if desired,* but otherwise this equation is under the control of TANK_LINKER. TANK_LINKER creates and controls an ODE for each tank pair (each independent TANK_LINKER call, which means each call with a unique ID_ODE).

* The variable X in the second order differential equation does not have units of length, but rather is dimensionless: it is the volume of Tank A (the total volume if Tank A is twinned) divided by VTOT. Thus, the nondimensional velocity dX/dt would need to be multiplied by VTOT/AEFF to yield an actual velocity in either m/s or ft/hr.

This ODE provides position and velocity information to TANK_LINKER that it uses to update the VDOT terms of Tank A and Tank B. Ideally, the VDOT of one tank would be directly proportional to this velocity, and the VDOT of the other tank would be the negative. However, error terms exist that require adjustments to each VDOT independently. One of the most significant sources of error is the fact that each tank is provided a COMP value corresponding to the compressibility of the adjacent tank. If the pressure of both tanks goes up during the next time step, both tanks would grow slightly, and the total growth over VTOT would be one error term that would need to be folded back into the next set of VDOT predictions. The drift of the volume of Tank A from the ODE predictions is another error term. TANK_LINKER attempts to keep these drifts from becoming too large without constricting the time step unnecessarily. Usually the errors are less than 1%. But if those error terms are unacceptable, the user can reduce the time step themselves (e.g., DTMAXF).

Restrictions: TANK_LINKER must be called from within FLOGIC 2, which is invoked at the end of each time step. It is intended to be used in transients, and only performs initializations if called from within a steady state solution (noting that FLOGIC 2 is invoked once at the end of a steady state solution).

Co-Solved ODE: TANK_LINKER co-solves a second order ODE representing the equation of linear motion one side of Tank A. (See Section 7.9 and especially Section 7.9.2.) Instead of distance as the solution variable "X" of the ODE, TANK_LINKER uses the volume of tank A* divided by VTOT to normalize it. (This is why VTOT should not be changed within a run.)

When it first starts a new ODE, TANK_LINKER sets the DXSIZE of *all* ODEs to be no more that 0.01 (1% change in normalized volume), whereas the default is 0.1 (10% change in X).

The user may call DEQTAB or DIFFEQ2G independently to output or fetch the values of the ODE, whose ID is provided by the variable ID_ODE.

Volume Limits: If the limits of travel, VLO or VHI are encountered, then the tanks become disconnected as would happen if using an iface: the VDOT and COMP are zeroed. Similarly, the ODE is reset to the limits (VLO/VTOT or VHI/VTOT), with velocity and acceleration set to zero at the limits. If a pressure force later encourages Tank A to move away from its limits, it will do so automatically.

Since this is an adjunct solution, VLO and VHI will not be met exactly: the volume of Tank A can be expected to overshoot those limits a little (up to a percent or two). If the model cannot tolerate exceeding these limits, use a smaller range of input values to accommodate such overshoot.

Steady-States: In steady-states, the volumes will be adjusted automatically based on the pressure differential at the end of the steady state (in FLOGIC 2). This may cause the volumes to hit the limits of VLO or VHI at the start of a transient, so a CHGVOL call may be required to make adjustments before calling a transient. Otherwise, use HLDLMP or HLDTANKS to freeze the two tanks during a STEADY call, and then use RELLMP or RELTANKS afterwards to release them.

---

* The can be a distinction between the X solved by the ODE, and the current value of VOL(LA)/VTOT [or (VOL(LA)+VOL(LA+1))/VTOT if tank A is twinned]. Similarly, dX/dt will not precisely match VDOT(LA)/VTOT. Such error terms are tracked and gradually corrected by TANK_LINKER.

If no steady-state solution is invoked prior to a transient, consider adding a relatively large compliance (COMP) to any all-liquid tanks in order to survive the first time step, until TANK_LINKER can be called at the end of the time step.

Restriction: Conflicts with other means of setting tank compliance, such as the COMPLQ routine (Section 7.11.9.6), can occur as described below.

The COMP and VDOT of both tanks may be set as a result of this call. (If a tank is twinned, only the values of COMP and VDOT of the primary tank are changed, though they affect both twins.) Do not override these values before the next solution. This restriction includes avoiding calls to COMPLQ. If calls to COMPLQ *are* made, save the COMP of both tanks before the COMPLQ call and restore them to the original values afterwards, such as:

```
COMP_TANKA  = ONE_MOD.COMP32    $ SAVE COMP VALUES (from TANK_LINKER)
COMP_TANKB  = OTHER.COMP44
CALL COMPLQ('ONE_MOD',0.0)     $ OVERWRITES ALL COMP VALUES
CALL COMPLQ('OTHER',0.0)
ONE_MOD.COMP32= COMP_TANKA     $ RESTORE COMP VALUES
OTHER.COMP44 = COMP_TANKB
```

Adjunct Solutions and Time Steps: Unless the specific dynamic response of device inserted between the tanks (e.g., bellows, piston, bladder) is the focus of the model, then it is often best to overestimate both the mass and friction, perhaps considerably, to avoid small time steps. This is especially true if one side contains a COMPLIQ fluid (thermodynamically compressible liquid, not to be confused with the COMPLQ routine above) and XL=0.0 in such a tanks. In that case, the time scale can be very small as the boundary mass oscillates on top of a column of liquid. Also, the mass itself isn't just the mass of the piston or bellows, but it should "borrow" some (at least half) of the fluid's mass too ("added" or "virtual" mass).

Restrictions: Do connect the same tank to more than one other tank using TANK_LINKER: use unique pairs with each call. Additional ifaces on the two tanks connected by TANK_LINKER are not illegal but such usage is strongly discouraged.* The program does not abort nor produce warnings based on these transgressions.

Restriction: Some gas or compressible liquid (6000 series COMPLIQ fluid) must be present in at least one of the tanks. Do not use TANK_LINKER if both tanks contain only incompressible liquids. If this condition happens during a transient, TANK_LINKER will abort.

Calling Sequence:

```
      CALL TANK_LINKER(MODELA, NA, MODELB, NB, ID_ODE
     .    ,VTOT, VLO, VHI
     .    ,VZERO, DPDV, MASS, AEFF, F_VEL)
```

* The FLAT iface generated and controlled by a twinned tank is an exception to this rule: either Tank A or Tank B may be twinned, or both may be twinned.

where:

MODELA .... Fluid submodel name containing tank A, in single quotes

NA..... User ID for tank A, perhaps the primary of a pair of twinned tanks

MODELB .... Fluid submodel name containing tank B, in single quotes. If this input is illegal (e.g., ' '), MODELB will be assumed to be the same as MODELA

NB..... User ID for tank B, perhaps the primary of a pair of twinned tanks

ID_ODE .... ID of 2nd order ODE that TANK_LINKER will create and control (there will be one such ODE per pair of TANK_LINKER-controlled tanks). See also Section 7.9. Use a unique ID for each pair of TANK_LINKER-controlled tanks.

VTOT ..... Total volume of control volume, ft^3 or m^3
VTOT should not change during a run.

VLO ..... Minimum volume of tank A (0 < VLO <= VHI < VTOT). This input is similar to that of an iface: see Section 3.8.2.4.
Do not use extremely small values of VLO. The actual volume of Tank A may go slightly below VLO, especially if the last time step taken is large.

VHI ..... Maximum volume of tank A (0 < VLO <= VHI < VTOT). This input is similar to that of an iface: see Section 3.8.2.4.
Do not use extremely large values: do not let VTOT-VHI (the lower limit on tank B) get too small. The actual volume of Tank A may go slightly above VHI, especially if the last time step taken is large.

VZRO ..... Tank A volume at which bellows forces is zero (may be any number, including negative). This input is similar to that of a SPRING iface: see Section 3.8.3.2 and especially Figure 3-19. Ignored if DPDV=0.0

DPDV ..... Bellows volumetric spring constant: the change in delta pressure (PL(NA)-PL(NB)) divided by the change in tank A volume. This input is similar to that of a SPRING iface: see Section 3.8.3.2 and especially Figure 3-19.
DPDV be zero for to mimic a FLAT iface, in which case VZRO is ignored.
Units are either psi/ft^3 or Pa/m^3

MASS ..... Mass of bellows, piston, etc. plus any added (virtual) fluid mass, lb_m or kg

AEFF ..... Effective area upon which forces act, ft^2 or m^2

F_VEL..... Friction coefficient: force per velocity.
May be zero. Otherwise if nonzero, F_VEL is a positive value.
Units are lb_f per ft/hr (lb_f-hr/ft), or N per m/s (N-s/m)

Note that EMA for an iface is equal to MASS/$AEFF^2$. These two parameters in separate inputs for TANK_LINKER, but are combined for ifaces.

Examples:

```
C Frictionless SPRING-like:
       CALL TANK_LINKER('MOD1',10,'MOD2',22,1022,
     .    VACC, 0.01*VACC, 0.99*VACC,
     .    0.5*VTOT, 1.5e7, 10.0, ASECT, 0.0)
```

```
C FLAT-like with friction, both lumps in same submodel:
       CALL TANK_LINKER('DRAINO',1,' ',2,555,
     .    VBOTTLE, 0.1*VBOTTLE, 0.9*VBOTTLE,
     .    0.0, 0.0, 0.5*DRAINO.DL1*DRAINO.VOL1,
     .    0.25*pi*DBOTTLE^2, 100.0)
```

7.11.9.4 Auxiliary Routines for Conduction in Wet Wicks

As an aid in preparing ftie conductances (GF) for USER fties (Section 3.7.1), routines are provided for modeling back-conduction terms wetted wicks. These terms are important for capillary pumped loops (CPLs) and loop heat pipes (LHPs).

One routine, WETWICK, helps predict the "wet" *conductivity* (W/m-K or BTU/hr-ft-F) of a porous structure given the "dry" conductivity of the wick material, its porosity, and taking into account the conductivity any liquid contained within the wick.

Two other routines, CYLWICK and PLAWICK, calculate an effective *conductance* (W/K or BUT/hr-F) of a wetted wick taking into account the heat exchange effect with the liquid flowing through it. CYLWICK assumes liquid flows outwork radially through a cylindrical wick, and PLAWICK assumes liquid flows through a planar wick.

This heat exchange correction is normally very small compared to using the formula based on "solid" conduction (i.e., $2\pi k_w L/\ln(r_o/r_i)$ or $k_w A/t$, where k_w is the wet conductivity perhaps produced from WETWICK), except in the limits of high flow rates and low wet conductivity, which are precisely the cases in which back-conduction itself becomes negligible. Furthermore, this correction is negligible compared to the large uncertainties in the values produced by WETWICK: the actual value of wet wick conductivity is best measured experimentally, or at least applied as an uncertainty in a data correlation task.

Caution: Mechanisms exist in CPLs and LHPs for "greater than theoretical" back-conduction, where "theoretical" means the predictions of the routines in this subsection. These phenomena have been characterized but are not completely understood. Users are cautioned to apply significant margin to hot case effective wick conductivities, and to discuss these effects and their potential magnitude with their CPL/LHP supplier.

Subroutine Name: WETWICK

Description: This routine estimates the conductivity of a wetted porous structure using one of three methods.

The parallel method (METH=1) simply assumes that the wick material and liquid are in parallel. This normally represents an upper limit on the returned conductivity for low-conductivity liquids. If *por* is the porosity, C_{liq} is the conductivity of the liquid, and C_{dry} is the conductivity of the dry wick:

$$C_{wet} = por \cdot C_{liq} + (1 - por) \cdot C_{dry}$$

The series method (METH=2) assumes that the wick and liquid are in series. This normally represents a lower limit on the returned conductivity for low-conductivity liquids:

$$C_{wet} = \left(\frac{por}{C_{liq}} + \frac{(1-por)}{C_{dry}}\right)^{-1}$$

The sintered method (METH=3) is based on a correlation from Dunn & Reay* and is a "best guess" estimate for materials such as sintered nickel:

$$C_{wet} = C_{dry} \cdot \frac{2+\gamma-2\cdot por\cdot(1-\gamma)}{2+\gamma+por\cdot(1-\gamma)}$$

$$\gamma = C_{liq}/C_{dry}$$

Calling Sequence:

```
CALL WETWICK (smn, lump, cdry, por, meth, cwet)
```

where:

`smn` ..... the fluid submodel containing the wick, in single quotes
`lump` ..... lump to use for liquid properties (otherwise vapor properties will be used if this lump has a quality of unity)
`cdry` ..... input "dry" conductivity of the wick material (W/m-K or BTU/hr-ft-F)
`por` ..... input porosity of the wick
`meth` ..... method to use to estimate conductivity:
1 ..... parallel (maximum CWET for low conductivity liquids)
2 ..... series (minimum CWET for low conductivity liquids)
3 ..... sintered estimate, from Dunn & Reay
`cwet` ..... the returned "wet" conductivity

Example:

```
HEADER FLOGIC 0, LHP
      CALL WETWICK('lhp', 100, 15.0, 0.5, 3, ctest)
```

Subroutine Name: PLAWICK

Description: This routine estimates the conductance of a wetted (planar) porous structure taking into account the heat exchange with fluid passing through it.

Without this correction, for a given total frontal area A and a given thickness t, the conductance would be:

$$GF = C_{wet} \cdot \frac{A}{t}$$

* Chapter 3.3.2, p 100.

Correcting for the reduced temperature gradient at the entrance of the wick yields:

$$GF = \frac{\dot{m} \cdot C_p}{e^{\alpha} - 1}$$

$$\alpha = (\dot{m} \cdot C_p \cdot t)/(C_{wet} \cdot A)$$

where C_p is the specific heat of the liquid, and $\dot{m}$ is the mass flow rate.

Calling Sequence:

```
CALL PLAWICK (smn, path, cwet, area, thick, gf)
```

where:

`smn`.....the fluid submodel containing the wick, in single quotes
`path`.....path to use for flow rate, and the defined upstream end of this path will be used for liquid properties (otherwise vapor properties will be used if this lump has a quality of unity). If the flow rate in this path is nonpositive, then the uncorrected (maximum) conductance will be returned.
`cwet`.....input "wet" *conductivity* of the wick material (W/m-K or BTU/hr-ft-F)
`area`.....input frontal area of the wick
`thick` .....input thickness of the wick
`gf` .....the returned effective *conductance* (W/K or BTU/hr-F)

Example:

```
HEADER FLOGIC 0, LHP
      CALL WETWICK('lhp', 100, 15.0, 0.5, 3, ctest)
      CALL PLAWICK('lhp', 150, ctest, length*width, 0.01, GF133)
```

Subroutine Name: CYLWICK

Description: This routine estimates the conductance of a wetted (cylindrical) porous structure taking into account the heat exchange with fluid passing through it. The liquid is assumed to flow radially from inner diameter to outer diameter

Without this correction, for a given total length L, and a given outer and inner diameter D_o and D_i, the conductance would be:

$$GF = C_{wet} \cdot \frac{2\pi \cdot L}{\ln\left(\frac{D_o}{D_i}\right)}$$

Correcting for the reduced temperature gradient at the entrance of the wick yields:

$$GF = \frac{\dot{m} \cdot C_p}{\left(\frac{D_o}{D_i}\right)^{\alpha} - 1}$$

$$\alpha = (\dot{m} \cdot C_p) / (C_{wet} \cdot 2\pi \cdot L)$$

where C_p is the specific heat of the liquid, and $\dot{m}$ is the mass flow rate.

Calling Sequence:

```
CALL CYLWICK (smn, path, cwet, Do, Di, length, gf)
```

where:

`smn`	the fluid submodel containing the wick, in single quotes
`path`	path to use for flow rate, and the defined upstream end of this path will be used for liquid properties (otherwise vapor properties will be used if this lump has a quality of unity). If the flow rate in this path is nonpositive, then the uncorrected (maximum) conductance will be returned.
`cwet`	input "wet" *conductivity* of the wick material (W/m-K or BTU/hr-ft-F)
`Do`.....	input outer diameter of the wick
`Di`.....	input inner diameter of the wick
`length`	input length of the wick
`gf`.....	the returned effective *conductance* (W/K or BTU/hr-F)

Example:

```
HEADER FLOGIC 0, LHP
      CALL WETWICK('lhp', 100, 15.0, 0.5, 3, ctest)
      CALL CYLWICK('lhp', 150, ctest, 0.02, 0.01, length, GF133)
```

7.11.9.5 Choked Flow Detection and Simulation Aids

Except for the routine WCHOKE, the routines listed in this section have been obsoleted. They are documented in case they are encountered in older models, as an aid to converting such models into more modern usage based on the MCH, HCH, and FRC parameters as described in Section 3.18.2.

In case they are not eliminated/converted before an older model is run, obsolete routines CHKCHL, CHKCHP, and CHOKER will issue a single caution yet will continue to operate, albeit under the new rules of usage. In the first call, they will set METH and HYST as MCH and HCH, respectively, noting that HCH is not completely equivalent to the prior meaning of HYST, and then they will check for choking as before. However, in subsequent calls if the path MCH is the same as the input METH value, they will assume that choking simulation has already been taken into account internally and will use the values of FRC etc. CHOKER, CHKCHL, CHKCHP will become undocumented in future versions, and their use is therefore discouraged. CHKCHP still has a useful purpose: issuing warnings if a path is in the choked flow mode. It has therefore been replaced with WCHOKE, which lacks the final argument (METH) that was used in the CHKCHP routine.

Subroutine Name: WCHOKE

Description: Intended to keep watch on all or part of a submodel for localized choking, WCHOKE prints out a warning to the output file when a sonic limit is encountered. No other action is taken; the user must then decide what, if any, model changes are required. WCHOKE may be used to keep an eye on single paths (or single pairs of twinned paths), or, by setting the path ID to zero, it may be used to watch all appropriate paths.

WCHOKE can be called from any logic block, although FLOGIC 2 is probably the most appropriate block for almost all analyses.

Restrictions: If all paths are to be checked, WCHOKE ignores connectors with no defined flow or throat areas. Flow area is an optional input for MFRSET, VFRSET, PUMP, TURBINE, COMPRESS, COMPPD, CAPIL, TABULAR, and NULL connectors, and WCHOKE will abort if such an individual device is targeted for which no positive AF is supplied. For twinned tubes and STUBE connectors, input the primary path only.

Calling Sequences:

```
CALL WCHOKE (fsmn, pathid)
```

where:

`fsmn` . . . . . fluid submodel name, single quotes
`pathid` . . . . . ID of path to check. Zero means check all paths in submodel. For twins, input ID of primary twin.

Examples:

```
CALL WCHOKE('GLUG',0)          $ CHECKS ALL PATHS
CALL WCHOKE('GLUG',10)         $ CHECKS PATH #10
```

Subroutine Name: CHKCHP

This routine has been obsoleted, and should be replaced by WCHOKE (above) if encountered, eliminating the final argument.

Calling Sequences:

```
CALL CHKCHP(fsmn, pathid, METH)
```

where:

fsmn . . . . . fluid submodel name, single quotes
pathid . . . . . ID of path to check. Zero means check all paths in submodel. For twins, input ID of primary twin.
METH. . . . . Will be set as the MCH value of pathid if pathid is nonzero (refers to a single path).

Subroutine Name: CHKCHL

This routine has been obsoleted, and should be deleted if encountered.

Calling Sequences:

```
CALL CHKCHL (fsmn, pathid, METH, ISCHK, FRC)
```

where:

`fsmn` . . . . . fluid submodel name, single quotes
`pathid` . . . . ID of path to check. For twins, input ID of primary twin. Zero is invalid and will result in an abort.
`METH` . . . . . applied as the MCH value of the path
`ISCHK`. . . . . returned integer flag:
0 = path is *not* choked
1 = path *is* choked
`FRC` . . . . . returned choked flow rate (calculated even if path is not choked)

Subroutine Name: CHOKER

This routine has been obsoleted, and should be deleted if encountered, perhaps placing the METH value as the path's MCH value and the HYST value as the path's HCH value.

Unlike the current methods, CHOKER did not apply to tubes (except within STEADY) and did not apply to twinned paths if the upstream tank was also twinned.

Calling Sequences:

```
CALL CHOKER(fsmn, pathid, METH, HYST)
```

where:

`fsmn`.....fluid submodel name, single quotes
`pathid`.....ID of path to check. Zero means check all relevant paths in submodel. For twins, input ID of primary twin.
`METH`.....applied as the path's MCH value
`hyst`.....applied as the path's HCH value

7.11.9.6 Modeling Water Hammer and Acoustic Waves

Using tanks, tubes, and an appropriately constrained time step, SINDA/FLUINT may be used to accurately simulate propagation of acoustic waves* in liquids and gases. Excellent comparisons have been made with analytic solutions, test results, and even other codes that specialize in the calculation of such phenomena. Figure 7-8, for example, shows a comparison between a method-of-characteristics (MOC) solution† and an equivalent FLUINT network solution.

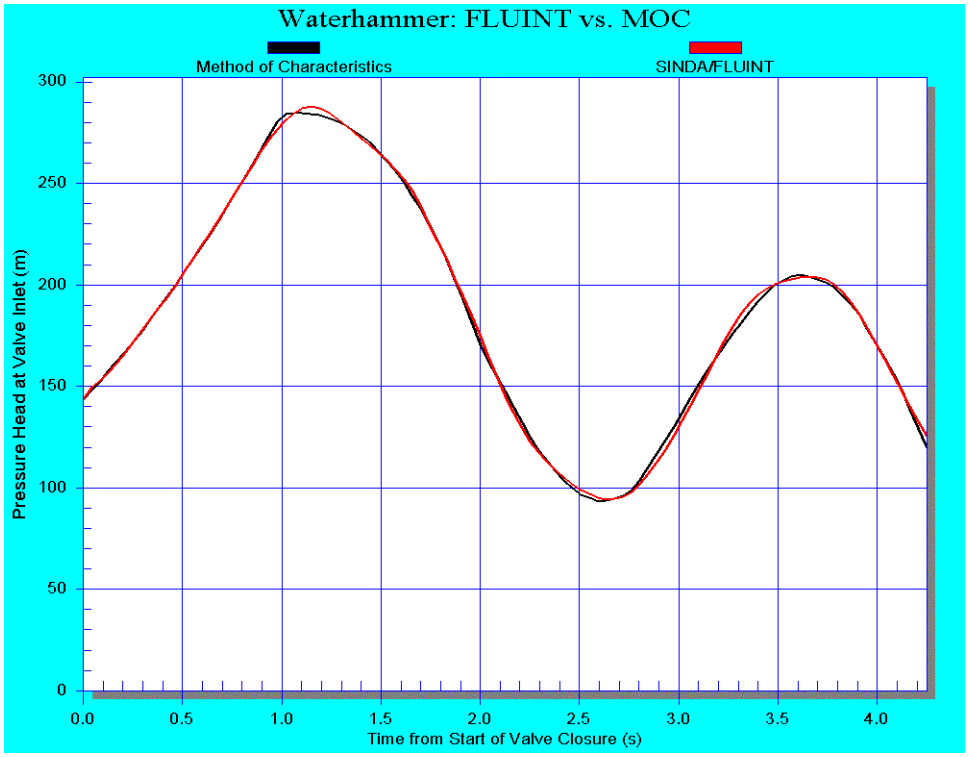


Figure 7-8 Comparison of FLUINT vs. Method of Characteristics for Waterhammer

This section describes the methods that should be applied along with two utility routines intended to facilitate such modeling. Refer also to Section 3.14.4.

Modeling fast hydrodynamic events such as water hammer and pressure waves implicitly requires the use of high temporal resolution fluid elements such as tanks and tubes. This does not mean that the entire model need be built from these elements. Rather, all significant volumes must be accounted for with tanks, and all significant lengths must be accounted for with tubes. Junctions and connectors may also be used as is appropriate given their instantaneous (massless, inertialess) nature.

* excluding shock waves

† Example 3-1 in Fluid Transients, Wylie and Streeter, 1982 edition.

Results are somewhat insensitive to the number of tanks and tubes (e.g., the spatial resolution). As long as no significant property (e.g., density) gradients exist in a line, relatively coarse resolution is often adequate. Higher resolution causes higher frequency responses, but rarely affects peak pressures. Sudden variations in spatial resolution within a duct, however, can lead to false reflections (see caution below).

Unlike spatial resolution, results can be very sensitive to the time step chosen (temporal resolution). FLUINT was specifically designed to avoid the necessity of taking time steps small enough to resolve such fast transient events. Because of its implicit solution, it will tend to smear out such waveforms and perhaps miss them altogether unless the time step is specifically constrained. Loss of such high frequency phenomena is no concern when the problem at hand focuses on lower frequency phenomena such as heat transfer transients. However, when the design concern is flow-induced loads, then accurate calculation of peak pressures requires that the time step be limited.

Comparisons with analytic solutions have yielded a *recommended* maximum time step to apply when accurate calculation of such waveforms and peak pressures are required: $[L/(2a)]_{min}$, where L is the length of a segment and a is the local speed of sound in that segment. Once a spatial resolution has been chosen, a maximum time step exists above which accuracy is lost. A routine ("WAVLIM") is available to calculate this limiting time step. *The user should note that even smaller time steps may be required in some models, and that either a safety factor should be applied or an assessment should be performed of the sensitivity of results to the chosen time step and spatial resolution.*

By default, liquids are assumed to be incompressible, with infinite sound speeds. The slight compressibility that most liquids exhibit can be accurately modeled with the compliance factor (Section 3.14.4). A routine ("COMPLQ") is available to facilitate this calculation. Alternatively, the liquid can be assumed to be thermodynamically compressible using the COMPLIQ feature of 6000 series FPROP DATA blocks (3.21.7), although extra compliances will be required in this case to include the compliance of the wall.

In addition to the compressibility of the fluid itself, additional compliances may be required to model wall stiffness, trapped gases, etc. Such compliances will reduce the sound speed.

Two-phase Flow Caution—While single-phase gas or liquid flows can be handled with confidence, caution must be exerted when both phases are present at the same time. In such two-phase flows, whether single- or multiple-constituent, if homogeneous (untwinned) tanks are used then actual speeds of sounds will be higher than those predicted by the program for pressure waves propagating *through* two-phase portions of a fluid network. This discrepancy is caused by the default assumption of thermal equilibrium between phases within tanks, which causes them to be excessively compressible during fast transients. Since peak pressures are very likely to be underpredicted when using perfectly mixed (singlet) tanks, ***the use of twinned tanks should be considered mandatory in such cases. However, in such cases the WAVLIM routine may overpredict the maximum step.***

This caution does not cover all acoustic phenomena in all situations involving two-phase flow. Rather, it applies if pressure waves travel *through* a two-phase zone. For example, excellent comparisons with test data have been made for a case where flashing ("column separation") occurred

locally at the end of a closed line during those portions of a transient event when the pressure dropped below saturation. The two-phase zone represented a boundary condition in this case, rather than a conduit for pressure waves, which in this case traveled only through single-phase zones.

When dissolved substances are present, their effect on the liquid speed of sound is neglected.

Partially Reflected Wave Caution—When an obstruction (e.g., partially open valve, reducer, sharp bend or tee) is encountered, the wave will split into forward and backward moving components, whereas the finite volume/difference methods used in SINDA/FLUINT do not track wavefronts explicitly. Simple K-factor LOSS-like elements have been shown to yield the same results in SINDA/FLUINT as those produced by MOC methods for partial reflections. However, validated methods are required for treating various other piping components. Therefore, use of the code in safety-critical applications involving such reflections is discouraged without validating modeling methods for the components contained within the piping system.

Caution Regarding Inconsistent Discretization—Normally, single-phase liquid water hammer results such as peak pressures are insensitive to whether a pipeline has been subdivided into 10 segments or 100 segments. However, strong variations in resolution *within* a line can lead to false pressure peaks. For example, if the upstream half of a line was modeled using a duct macro with 10 segments, and the downstream half used a different macro with 90 segments (a 9 fold increase in spatial resolution), an artificial reflection can arise where the two resolutions meet. The user should try to maintain a consistent resolution, whether fine or coarse, throughout the water hammer model. In other words, all tanks within any one line should have *about* the same volume. Specifically, tanks should not have adjacent tubes whose AF*TLEN does not correspond to the volume of that tank: the time constant of a tank should be about the same order as that of the tubes through it.

Consider a center-discretized or upstream-discretized duct macro ending in a junction, and downstream of that junction is a valve that opens and closes. If that junction is changed to small FLUINT tank to represent the fluid volume of the valve body, it no longer corresponds to the inertia of the line just upstream. When the valve closes, high-frequency pressures of exaggerated amplitude can result since that small tank is attempting to deal with the inertia of the upstream tube, whose length and diameter bear no relationship to the volume of the small tank. If the valve is large enough, or if it opens and closes in a finite amount of time (specifically, slower than the period of these oscillations), no issue will arise. Otherwise, it would be best to leave the end point as a junction.

Subroutine Name: COMPLQ

Description: This routine sets the compliance of all liquid-containing tanks in a fluid submodel to be equal to the compressibility of the otherwise incompressible liquid. (For COMPLIQ fluids, which already include liquid compressibility, no additional compliance is required nor added if COMPLQ is called accidentally.) In other words, if the liquid phase is assumed to be incompressible, the slight compressibility exhibited by most liquids can be modeled as a slight deflection of the control volume boundary. This routine is intended as an aid in modeling water hammer, or for easily adding realistic compliances into a system simulation.

COMPLQ should be called from within FLOGIC 0. If WAVLIM is also used, it should be called *after* COMPLQ.

Restrictions: This routine overwrites the current value of COMP for all tanks in the model, regardless of their thermodynamic state, if the tank contains any incompressible liquid. Therefore, it should not be used in combination with routines that manipulate the COMP value.

The liquid phase is assumed incompressible (zero compliance) if the fluid is a 9000 series user fluid, or if it is a 6000 series user fluid in which neither VDLC nor VSOSF have been supplied. No warnings are produced in these cases.

Calling Sequences:

```
CALL COMPLQ (fsmn, cmpadd)
```

where:

`fsmn`.....fluid submodel name, single quotes
`cmpadd`.....additional compliance to be added to all tanks

Normally, cmpadd will be zero although it may be used to specify container rigidity, included gases, etc. The compliance of all tanks will be set to cmpadd. An additional compliance (corresponding to the liquid compressibility) will be added for all tanks that are currently contain any incompressible liquid, in proportion to their current liquid volume fraction (1.0-AL).

Example:

```
CALL COMPLQ('FLOSS',0.0)
```

Subroutine Name: WAVLIM

Description: Calculates the maximum recommended time step to be used if the capture of fast hydrodynamic transient events is required. This limit is calculated as one half the ratio of the smallest tube length in the submodel to the largest sound speed in all tanks in the submodel: $TLEN_{min}/(2*a_{max})$. This is a recommendation only; even smaller time steps are often necessary.

For liquid-filled tanks, the sound speed is calculated on the basis of user-supplied compliances. For tanks containing any vapor, the sound speed is calculated on the basis of the fluid sound speed plus the effects of any additional compliances.

WAVLIM should be called from within FLOGIC 0. If COMPLQ is also used, it should be called *before* WAVLIM.

Restrictions: An abort will result if both tanks and tubes are not present, or if all tanks are noncompliant and contain liquid.

Restrictions: ***WAVLIM is unable to include the effects of ifaces and twinned tanks*** in its estimation. If ifaces are present, the user should experiment with small time steps until no change in the results are noted.

Calling Sequences:

```
CALL WAVLIM (fsmn, dTmax)
```

where:

`fsmn` ..... fluid submodel name, single quotes
`dTmax` ..... returned maximum time step for fast hydrodynamic transients

Examples:

```
CALL WAVLIM('SURFS',SURFS.DTMAXF)

CALL WAVLIM('UP',DTEST)
UP.DTMAXF = 0.2*DTEST
```

7.11.9.7 Modeling Wyes (Ys), Tees (Ts), and Manifolds

Often, head losses at tees and wyes (3-branch divisions or combinations of flow paths) are negligible compared to losses in the lines themselves. In systems with few other losses, however, accurate modeling of tee effects can make a significant difference in the resulting predictions of flow distributions.

Manifolds, if they cannot be assumed to be perfect (i.e., modeled with a single lump, and perhaps even using duplication factors for the branch lines to further reduce the model), or if they cannot be assumed to be so long and thin (i.e., very *im*perfect) as to have negligible tee effects, should be modeled using the methods outlined below.

Duct macros (Section 3.9.2) automatically include the acceleration effects of axial variations in density and flow area, and they attempt to include the effects of side flow branches. However, these macros rely on a one-dimensional momentum equation, and many effects of tees and wyes are multi-dimensional. For example, flow leaving a macro path is assumed to do so radially, taking no momentum with it (at least not in vector representing the direction of the macro). In the real case, some flow leaves at an angle to the main line even if the piping connection is perpendicular. Thus, duct macros will overestimate the accelerations and decelerations associated with side flow passages.

This section details routines and methods for modeling combining and splitting flows that empirically take into account the actual geometries involved. These methods should be used instead of duct macros when tee effects are important compared to normal line losses, but may also be used in addition to duct macros as a slightly less accurate but perhaps more convenient alternative.

FLUINT works with static and not total pressures. While a K-factor loss in total pressure is also a K-factor loss in static pressure, a K-factor does not completely describe the situation with respect to static pressures: additional terms are required, such as the AC factor in tubes or STUBEs. These terms can be folded into the K-factor, however, to create an *effective K-factor* that provides the complete change in static pressures using no other term.

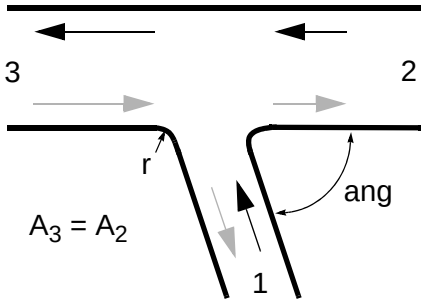
Therefore, the tee and wye routines are divided into two steps either of which can be used independently:

1. Correlations (routines YTKALG and YTKALI) that predict the irrecoverable loss (change in total pressure) K-factors. These K-factors are based on the dynamic head of the leg with the combined (largest) flow. The main line might be represented with a duct macro.

2. A utility (routine YTKONV) to convert the above K-factors into effective factors for the other two legs and that include acceleration effects as well: equivalent K-factors that include both irrecoverable losses (total pressure changes) as well as recoverable losses (static pressure changes). The main line might be represented with a duct macro if YTKONV is used.

A third category of wye-tee routines is available that control abort and warnings in the above routines: YTHUSH and YTWARN.

Subroutine Name: YTKALG



Description: This routine uses Gardel's* curve fit equations to estimate the *total* pressure K-factor losses for a wye or tee of the configuration shown at the right: flow to or from a side line (branch) and a main line at various angles, and with variable fillet radii. The flow areas of path 3 and path 2 must be the same. *K-factors that are calculated are based on the dynamic head of path #3, and should be adjusted if applied to paths #2 and #1.* This may require that the returned K-factor for path #1 be multiplied by the square of the ratio of the velocities, as noted below. Note that the calculated K-factors are sometimes negative.

Note that not all possible permutations of flow rate directions are possible. Rather, the flow must be moving in the directions shown in either the black arrows (merging) or grey arrows (diverging). Flow cannot be zero in the combined path (#3), but may be zero in paths #1 or #2. If, for example, only the flow rate in path #1 reverses, then the definition of #2 and #3 switches and an alternative call to YTKALG should be made with different arguments.

Since FLUINT works with static and not total pressures, the returned K-factors should not be used directly unless (1) they have been converted to effective (static pressure) K-factors using perhaps YTKONV, or (2) recoverable losses have been accounted for separately *and* the K-factors have been adjusted to reflect the dynamic head of the pipe to which they are applied. If pipes #2 and #3 are part of a single duct macro, the AC factor is updated automatically to reflect recoverable losses for those legs, but that macro does not apply any recoverable gains or losses to the side line (#1). If the tee performance is critical compared to line losses (e.g., a manifold is being designed), then YTKONV should be used and a duct macro should *not* be used for pipes #2 and #3 since those macros tend to overpredict acceleration terms in manifolds.

Calling Sequence:

```
CALL YTKALG (smn, p1, p2, p3, f1, f2, ang, rd)
```

where:

`smn` ..... the fluid submodel containing the tee, in single quotes
`p1`..... side flow path ID: path 1 in diagram
`p2`..... straight flow path ID: path 2 in diagram
`p3`..... combined flow path ID: path 3 in diagram
`f1`..... the returned K-factor, *based on the dynamic head of p3,* to apply to p1 perhaps after conversion with YTKONV, or at least after being multiplied by $(FR_3A_1/(FR_1A_3))^2$

* Gardel (1957) as referenced in Internal Fluid Flow by Ward-Smith (1980) with additional corrections from other sources as needed to match other data. Conflicting definitions exist, such as the definition of the nondimensional fillet radius, so as with other correlations use with caution and conservatism, and perhaps comparison with YTKALI.

`f2` . . . . . the returned K-factor, *based on the dynamic head of p3*, to apply to p2 perhaps after conversion with YTKONV, or at least after being multiplied by $(FR_3A_2/(FR_2A_3))^2$

`ang` . . . . . angle (in degrees) between p1 and p2, normally between 15.0 and 135.0

`rd` . . . . . ratio of the fillet radius to the diameter of p3 (r/D3 or r/D2, since D2=D3), normally between 0.0 and 0.2

The routine should be called from FLOGIC 0. See restrictions and assumptions below.

Example:

```
c The following example calculates the K-factors for paths 100 and 200
c which flow into path 300 (FR100 + FR200 is nearly equal to FR300).
c The “total K-factors” based on p3 are then converted into
c “static K-factors” based on p1 and p2
c
HEADER FLOGIC 0, MIGHTY
      CALL YTKALG(’mighty’, 100, 200, 300, ftest, gtest, 90.0, 0.0)
      CALL YTKONV(’mighty’, 100, 200, 300, ftest, gtest, fk100, fk200)
```

Restriction: The three named paths must:

1. share a common lump (the direction of positive flow rate for each path, however, is irrelevant)
2. have DUPI and DUPJ factors of unity (at least those adjacent to the common lump)
3. be either tubes, STUBE connectors, or LOSS-type connectors (LOSS, LOSS2, CTLVLV, CHKVLV, UPRVLV, and DPRVLV, although K-factors are not normally specified for UPRVLV and DPRVLV connectors). If tubes or STUBE connectors are used, then flow areas should not change: AFI/AFJ should be negative (not used), or should be equal.

Restriction: Zero flow rate in path p3 is illegal. If path p3 flows into the tee (diverging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and both should be flowing out of the tee, although one of these paths may have zero flow rate. Otherwise, if the flow in path p3 flows out of the tee (merging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and should be flowing into the tee, although one of these paths may have zero flow rate. If all flow rates reverse at once, no problems should occur, but if the flow rate in just one path reverses the program may abort (see YTWARN and YTHUSH below). In this case, logic should be added (perhaps with multiple conditional calls to YTKALG) to handle this case.

In the following example, the flow in p1 occasionally reverses. When this happens, p3 becomes the straight path and p2 becomes the combined flow. Not only must the K-factors must be moved to a new location, but the angle must change since p2 and p3 have swapped definitions. (In the following example, there is no other K-factor associated with the main line, so it is set to a small value.)

```
c Flow reversal in the side line (p1) causes p2 and p3 to swap:
c
HEADER FLOGIC 0, MIGHTY
      IF(FR100 .GE. 0.0)THEN
       CALL YTKALG('mighty', 100, 200, 300, ftest, gtest, 60.0, 0.01)
       CALL YTKONV('mighty', 100, 200, 300, ftest, gtest, fk100, fk200)
       FK300 = 1.0E-2
      ELSE
       CALL YTKALG('mighty', 100, 300, 200, ftest, gtest, 120.0, 0.01)
       CALL YTKONV('mighty', 100, 300, 200, ftest, gtest, fk100, fk300)
       FK200 = 1.0E-2
      ENDIF
```

Caution: Normally, the tee will be represented by a junction, but a tank is also acceptable in most cases. However, if the volumetric flow rates are not balanced (see last paragraph) within about 10%, a caution is issued but execution will continue. While the underlying correlations were not intended for local compression effects (i.e., if the lump is a tank), the purpose of the caution is more to advise that the paths may not be named in the right order in the subroutine call, or that flow rates have reversed and logic must be added to handle this case (see above), or that other paths might exist on the tee that violate the underlying assumptions of the correlation.

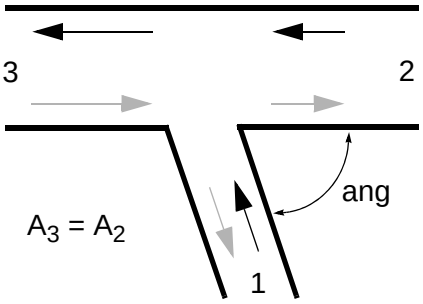
Caution: YTKALG predicts *two* K-factors for *three* flow legs, reflecting the fact that the underlying correlation was based on three pressure taps: the pressure of the junction itself was not measured (although this measurement would have permitted calculation K-factors for each leg). YTKONV applies all static pressure adjustments to paths #1 and #2, and none to path #3. Such "all or nothing" approaches correctly account for overall wye/tee performance, but leave an uncertainty in the prediction of the junction pressure itself. While these methods have been validated for certain manifolds with substantial (K-factor greater than 25) losses in the lateral legs, if the wye/tee losses have a measurable effect on the side branch flow rate, then predictions should be used with extreme caution.

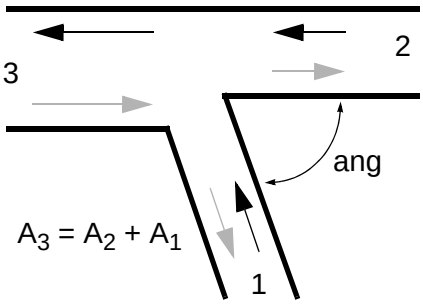
Guidance: The routine will not return a K-factor value whose absolute value is smaller than 10^{-4} since some path types (e.g., LOSS-type connectors) cannot tolerate zero K-factors.

Assumptions: Fully developed entrance flows, fully turbulent (high Reynolds number), circular cross sections, and largely incompressible flow is assumed. The flow area of p1 (side line) is assumed to be less than or equal to the main line.

Guidance: To suspend aborts and perhaps warnings resulting from unsupported flow rate directions, refer to the YTWARN and YTHUSH routines documented below.

Subroutine Name: YTKALI





Description: This routine uses Idelchik's* curve fit equations and tables to estimate the *total* pressure K-factor losses for a wye or tee of the configuration shown at the right: flow to or from a side line (branch) and a main line at various angles. The flow areas of path 3 and path 2 must be the same. *K-factors that are calculated are based on the dynamic head of path #3, and should be adjusted if applied to paths #2 and #1.* This may require that one or perhaps both K-factors be multiplied by the square of the ratio of the flow areas, as noted below. Note that the calculated K-factors are sometimes negative.

Unlike YTKALG, the fillet radius is not included: a sharp-edged tee is assumed. Also unlike YTKALG, YTKALI can handle the additional case shown below at the right, where the flow area of paths #1 and #2 sum to the flow area of path #3.

Note that not all possible permutations of flow rate directions are possible. Rather, the flow must be moving in the directions shown in either the black arrows (merging) or grey arrows (diverging). Flow cannot be zero in the combined path (#3), but may be zero in paths #1 or #2. If, for example, only the flow rate in path #1 reverses, then the definition of #2 and #3 switches for the first case (A_3=A_2) but renders the second case (A_3=A_2+A_1) invalid. Even if A_3=A_2, if flow rates reverse then an alternative call to YTKALI should be made with different arguments

Since FLUINT works with static and not total pressures, the returned K-factors should not be used directly unless (1) they have been converted to effective (static pressure) K-factors using perhaps YTKONV, or (2) recoverable losses have been accounted for separately *and* the K-factors have been adjusted to reflect the dynamic head of the pipe to which they are applied. If pipes #2 and #3 are part of a single duct macro, the AC factor is updated automatically to reflect recoverable losses for those legs, but that macro does not apply any recoverable gains or losses to the side line (#1). If the tee performance is critical compared to line losses (e.g., a manifold is being designed), then YTKONV should be used and a duct macro should *not* be used for pipes #2 and #3 since those macros tend to overpredict acceleration terms in manifolds.

Calling Sequence:

```
CALL YTKALI (smn, p1, p2, p3, f1, f2, ang)
```

* I.E. Idelchik, Handbook of Hydraulic Resistance, 3rd Edition (1996). This reference is fraught with typographical errors, and while attempts have been made to detect and correct them, use with caution and conservatism, and perhaps comparison with YTKALG where possible.

where:

`smn` ..... the fluid submodel containing the tee, in single quotes
`p1`..... side flow path ID: path 1 in diagram
`p2`..... straight flow path ID: path 2 in diagram
`p3`..... combined flow path ID: path 3 in diagram
`f1`..... the returned K-factor, *based on the dynamic head of p3*, to apply to p1 perhaps after conversion with YTKONV, or at least after being multiplied by $(FR_3A_1/(FR_1A_3))^2$
`f2`..... the returned K-factor, *based on the dynamic head of p3*, to apply to p2 perhaps after conversion with YTKONV, or at least after being multiplied by $(FR_3A_2/(FR_2A_3))^2$
`ang` ..... angle (in degrees) between p1 and p2, normally between 15.0 and 135.0

The routine should be called from FLOGIC 0. See restrictions and assumptions below.

Example:

```
c The following example calculates the K-factors for paths 100 and 200
c which flow into path 300 (FR100 + FR200 is nearly equal to FR300).
c The “total K-factors” based on p3 are then converted into
c “static K-factors” based on p1 and p2
c
HEADER FLOGIC 0, MIGHTY
      CALL YTKALI(’mighty’, 100, 200, 300, ftest, gtest, 90.0)
      CALL YTKONV(’mighty’, 100, 200, 300, ftest, gtest, fk100, fk200)
```

Restriction: The three named paths must:

1. share a common lump (the direction of positive flow rate for each path, however, is irrelevant)
2. have DUPI and DUPJ factors of unity (at least those adjacent to the common lump)
3. be either tubes, STUBE connectors, or LOSS-type connectors (LOSS, LOSS2, CTLVLV, CHKVLV, UPRVLV, and DPRVLV, although K-factors are not normally specified for UPRVLV and DPRVLV connectors). If tubes or STUBE connectors are used, then flow areas should not change: AFI/AFJ should be negative (not used), or should be equal.

Restriction: Zero flow rate in path p3 is illegal. If path p3 flows into the tee (diverging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and both should be flowing out of the tee, although one of these paths may have zero flow rate. Otherwise, if the flow in path p3 flows out of the tee (merging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and should be flowing into the tee, although one of these paths may have zero flow rate. If all flow rates reverse at once, no problems should occur, but if the flow rate in just one path reverses the program may abort (see YTWARN and YTHUSH below). In this case, logic should be added (perhaps with multiple conditional calls to YTKALI) to handle this case.

In the following example, the flow in p1 occasionally reverses, and A_3=A_2. When this happens, p3 becomes the straight path and p2 becomes the combined flow. Not only must the K-factors must be moved to a new location, but the angle must change since p2 and p3 have swapped definitions. (In the following example, there is no other K-factor associated with the main line, so it is set to a small value.)

```
c Flow reversal in the side line (p1) causes p2 and p3 to swap:
c
HEADER FLOGIC 0, MIGHTY
      IF(FR100 .GE. 0.0)THEN
       CALL YTKALI('mighty', 100, 200, 300, ftest, gtest, 60.0)
       CALL YTKONV('mighty', 100, 200, 300, ftest, gtest, fk100, fk200)
       FK300 = 1.0E-2
      ELSE
       CALL YTKALI('mighty', 100, 300, 200, ftest, gtest, 120.0)
       CALL YTKONV('mighty', 100, 300, 200, ftest, gtest, fk100, fk300)
       FK200 = 1.0E-2
      ENDIF
```

Caution: Normally, the tee will be represented by a junction, but a tank is also acceptable in most cases. However, if the volumetric flow rates are not balanced (see last paragraph) within about 10%, a caution is issued but execution will continue. While the underlying correlations were not intended for local compression effects (i.e., if the lump is a tank), the purpose of the caution is more to advise that the paths may not be named in the right order in the subroutine call, or that flow rates have reversed and logic must be added to handle this case (see above), or that other paths might exist on the tee that violate the underlying assumptions of the correlation.

Caution: YTKALI predicts *two* K-factors for *three* flow legs, reflecting the fact that the underlying correlation was based on three pressure taps: the pressure of the junction itself was not measured (although this measurement would have permitted calculation K-factors for each leg). YTKONV applies all static pressure adjustments to paths #1 and #2, and none to path #3. Such "all or nothing" approaches correctly account for overall wye/tee performance, but leave an uncertainty in the prediction of the junction pressure itself. While these methods have been validated for certain manifolds with substantial (K-factor greater than 25) losses in the lateral legs, if the wye/tee losses have a measurable effect on the side branch flow rate, then predictions should be used with extreme caution.

Guidance: The routine will not return a K-factor value whose absolute value is smaller than 10^{-4} since some path types (e.g., LOSS-type connectors) cannot tolerate zero K-factors.

Assumptions: Fully developed entrance flows, fully turbulent (high Reynolds number), circular cross sections, and largely incompressible flow is assumed. The flow area of p1 (side line) is assumed to be less than or equal to the main line if A_3=A_2.

Guidance: To suspend aborts and perhaps warnings resulting from unsupported flow rate directions, refer to the YTWARN and YTHUSH routines documented below.

Subroutine Name: YTKONV

Description: This routine converts irrecoverable (total pressure) K-factors based on the combined path (#3) to effective (static pressure) K-factors based on the straight and side paths (#1 and #2) that include recoverable losses (velocity changes due to accelerations and flow area changes).

YTKONV is normally intended to be called after YTKALI or YTKALG, but can be used directly by the user if needed. Refer to the diagrams documenting those routines for the definitions of paths 1, 2, and 3.

The products of this routine can be directly applied to the K-factor of a LOSS-type connector (LOSS, LOSS2, CHKVLV, CTLVLV) or to a tube or STUBE connector. If paths #2 and #3 (the main line) are part of the same duct macro, YTKONV detects this usage and returns a K-factor for path #2 that excludes acceleration terms since they are already included in the duct macro.

Calling Sequence:

```
CALL YTKONV (smn, p1, p2, p3, f1, f2, e1, e2)
```

where:

`smn`	the fluid submodel containing the tee, in single quotes
`p1`.....	side flow path ID: path 1 in diagram (see YTKALG or YTKALI)
`p2`.....	straight flow path ID: path 2 in diagram (see YTKALG or YTKALI)
`p3`.....	combined flow path ID: path 3 in diagram (see YTKALG or YTKALI)
`f1`.....	the input total pressure K-factor for p1, based on the dynamic head of p3
`f2`.....	the input total pressure K-factor for p2, based on the dynamic head of p3
`e1`.....	the resulting converted or “effective static pressure K-factor” to apply to p1, based on the dynamic head of that path
`e2`.....	the resulting converted or “effective static pressure K-factor” to apply to p2, based on the dynamic head of that path

The routine should be called from FLOGIC 0, usually after the related call to YTKALG and YTKALI. See restrictions and assumptions below.

Example:

```
HEADER FLOGIC 0, MIGHTY
      CALL YTKALG('mighty', 100, 200, 300, ftest, gtest, 90.0, 0.0)
      CALL YTKONV('mighty', 100, 200, 300, ftest, gtest, fk100, fk200)
```

Restriction: The three named paths must:

1. share a common lump (the direction of positive flow rate for each path, however, is irrelevant)
2. have DUPI and DUPJ factors of unity (at least those adjacent to the common lump)

3. be either tubes, STUBE connectors, or LOSS-type connectors (LOSS, LOSS2, CTLVLV, CHKVLV, UPRVLV, and DPRVLV, although K-factors are not normally specified for UPRVLV and DPRVLV connectors). If tubes or STUBE connectors are used, then flow areas should not change: AFI/AFJ should be negative (not used), or should be equal.

Caution: Zero flow rate in path p3 is not illegal, but will result in a very small K-factor and perhaps a caution (see below). If path p3 flows into the tee (diverging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and both should be flowing out of the tee, although one of these paths may have zero flow rate. Otherwise, if the flow in path p3 flows out of the tee (merging flow), then the volumetric flow rates in paths p1 and p2 should roughly sum to p3 and should be flowing into the tee, although one of these paths may have zero flow rate. If all flow rates reverse at once, no problems should occur, but if the flow rate in just one path reverses the program may issue a caution. In this case, conditional logic should be considered to handle this case.

Caution: Normally, the tee will be represented by a junction, but a tank is also acceptable in most cases. However, if the volumetric flow rates are not balanced (see last paragraph) within about 10%, a caution is issued but execution will continue. While the underlying correlations were not intended for local compression effects (i.e., if the lump is a tank), the purpose of the caution is more to advise that the paths may not be named in the right order in the subroutine call, or that flow rates have reversed and logic must be added to handle this case (see above), or that other paths might exist on the tee that violate the underlying assumptions of the correlation.

Caution: YTKALI and YTKALG each predict *two* K-factors for *three* flow legs, reflecting the fact that the underlying correlation was based on three pressure taps: the pressure of the junction itself was not measured (although this measurement would have permitted calculation K-factors for each leg). YTKONV applies all static pressure adjustments to paths #1 and #2, and none to path #3. Such “all or nothing” approaches correctly account for overall wye/tee performance, but leave an uncertainty in the prediction of the junction pressure itself. While these methods have been validated for certain manifolds with substantial (K-factor greater than 25) losses in the lateral legs, if the wye/tee losses have a measurable effect on the side branch flow rate, then predictions should be used with extreme caution.

Guidance: The routine will not return an effective K-factor value whose absolute value is smaller than 10^{-4} since some path types (e.g., LOSS-type connectors) cannot tolerate zero K-factors. It will also not return an effective K-factor whose absolute value is greater than 10^{3}.

Guidance: To suspend aborts and perhaps warnings resulting from unsupported flow rate directions, refer to the YTWARN and YTHUSH routines documented below.

Guidance: If the paths are part of other wyes or tees, then the K-factors for each wye or tee should be calculated separately and summed.

Guidance: If paths #2 and #3 (the main line) are part of the same duct macro, YTKONV detects this usage and returns a K-factor for path #2 that excludes acceleration terms since they are already included in the duct macro. However, since duct macros tend to overpredict these acceleration terms, best results are obtained if duct macros are *not* used for the main line.

Subroutines YTHUSH and YTWARN

The ability to suspend warnings and certain aborts is available for the wye-tee (merge/split) simulation routines YTKALI, YTKALG, and YTKONV.

These features are implemented via two routines with no arguments; YTWARN and YTHUSH. If YTWARN is called, then aborts *but not warnings* associated with illegal flow rate directions are suspended. This does not affect aborts due to illegal networks nor illegal flow areas. If YTHUSH is called then aborts *along any warnings* associated with illegal flow rate directions are suspended.

The user is cautioned that the resulting predictions are not to be trusted with either of these modes enabled: they exist only to survive initial conditions and/or steady-state perturbations without aborting. If the YTHUSH mode has been enabled, it should be disabled with a call to YTWARN before accepting any resulting answers (e.g., in a final call to STEADY perhaps). YTHUSH and YTWARN may be used to toggle the cautions on and off as needed, but if *either* of them is *ever* called, then aborts will no longer be invoked from the YT family of routines for the duration of the run.

These routines can be called anywhere, but OPERATIONS is a logical placement for most applications.

Calling sequences:

```
CALL YTHUSH          $ Suspend aborts and warnings
CALL YTWARN          $ Suspend aborts not warnings
```

As a final note, YTKALG is recommended over YTKALI for flow rate cases which are outside of the valid ranges of the correlations. YTKALI has discontinuities that are absent in YTKALG. Furthermore, unpublished comparisons of YTKALG with computational fluid dynamics (CFD) results (unpublished and performed by a third party) suggest that its correlations can be extended past the official range, at least for a few studied cases.

7.11.9.8 Two-phase Loop Balancer

In STDSTL or in transient solution routines, the mass and energy of FLUINT tanks are tracked and pressures are calculated: no plena are needed to specify a reference pressure. In other words, the pressure in a “closed system” (consisting of junctions and at least one tank, but lacking plena and HLDLMP lumps) is self-determining.

Unfortunately STDSTL is rarely the most efficient way to achieve a steady state for fluid sub-models because it solves the full transient equations. It is therefore sensitive to poor initial conditions, treating them as a severe transient requiring small pseudo time steps to resolve. STEADY is usually preferred. Unlike STDSTL, STEADY relaxes a set of time-independent equations: tank masses and volumes are ignored since all tanks are treated as junctions. Junctions’ pressures, however, can only be defined as *differences* relative to some reference state: a plenum or held (HLDLMP) lump. For this reason, a network lacking such a reference state is illegal in STEADY.

In many systems, the mass and volume are known but the operating pressures are not known and must be determined as part of the solution. For these systems, the above distinction between STDSTL (predicting pressures slowly by conserving total mass) and STEADY (fast but requiring that a pressure be specified, with total mass being an output) becomes an issue: neither approach is universally suitable. Either a long STDSTL solution must be endured, or the reference pressure(s) in a STEADY solution must be adjusted to balance energy flows and/or achieve the desired total fluid mass.

Examples of such systems with definite fluid “charge” and auto-determining pressures include vapor compression and other power cycles, loop heat pipes (LHPs), detailed hydrodynamic models of heat pipes* and counterflow thermosyphons, and loop thermosyphons (LTSs). Sample Problem G shows three distinct ways of working with power cycles. For the remainder of the list (passive or pump-free two-phase loops), the BALTPL utility can be used to balance energy flows in STEADY solutions.

Subroutine Name: BALTPL

Description: BALTPL adjusts the pressure in a nearly isobaric system (e.g, a transport loop versus a power cycle), nudging it toward a point at which energy flows in and out of the system are balanced. BALTPL is meant to be called repeatedly from FLOGIC 0.

The primary purpose of BALTPL is to help LHP and LTS modeling in STEADY. However, with suitable cautions it can be used for other applications and even other solution routines such as transients.

* Such detailed thermohydraulic models are not needed by *users* of heat pipes. Rather, thermal-only methods such as HEATPIPE and HEATPIPE2 are normally used.

BALTPL works by estimating a system pressure that will cause heat flows into and out of the loop to balance, using current tie conductances and energy imbalances (current UA and QL values plus current net power in plena and held lumps). The calculation takes into account path and tie duplication factors. All of the pressures within the loop are then shifted by the same amount, holding quality constant.

Calling Sequence:

```
CALL BALTPL (modnam, lumref, speed)
```

where:

`modnam` .... fluid submodel name, in single quotes

`lumref` .... User ID of the lump within that submodel to use as a reference point. This point ideally be a plenum or held tank or junction (i.e., HLDLMP applied), and that lump should also ideally represent a two-phase state (0 < XL < 1). If zero is input, the code will search for a reference point, in which case only one can exist otherwise an error message will result.

`speed`..... Accelerating/damping factor. Must be real and positive. Use 1.0 as a starting point. A factor greater than 1.0 accelerates the adjustments; a factor less than 1.0 slows them. If steady state convergence takes too many iterations, increase *speed*. If nonconvergence is experienced, consider reducing *speed*.

Examples:

```
if(nsol .eq. 0) call baltpl('lts1', 0, 1.0)     $ STEADY ONLY

if(nsol .le. 1)then
      call baltpl('lhp', 102, 10.0)             $ STEADY STATE
else
      call baltpl('lhp', 102, 1.0)              $ TRANSIENT
endif
```

Guidance:LHP Modeling. For LHP modeling, the compensation chamber (CC, also known as a hydraulic accumulator) should be the reference point, perhaps using a HLDLMP tank or pair of twinned tanks that are released for subsequent STDSTL or transient solutions. Except in the reflux mode (gravity assist), higher speeds (speed values perhaps as high as 10 or more) are acceptable. A change in the quality of the CC will change loop charge, but as long as this device stays in the two-phase range (and back-conduction or other conductances are not modeled as a function of CC void fraction), no change in loop operating point will result. This independence means that the mass in an LHP can usually be adjusted after a STEADY call and before any subsequent transient solutions.

Guidance:LTS Modeling. For LTS modeling, the evaporator outlet or condenser inlet are good reference points as long as they aren't superheated. A flow-through reference point (versus a plenum or held lump on the side of the loop--connected to the loop using a zero-flow STUBE, for example) is recommended. Reduced *speed* values are sometimes required to assist in convergence since the iterations can be jittery in buoyancy-driven flows already, and BALTPL represents a adjunct (not

implicit) adjustment to the solution. Also, in LTSs (unlike LHPs) the mass in the loop and its performance will depend on the quality in the flow-through reference point: separate iterations or adjustments must be used to achieve a desired loop charge (total fluid mass).

Guidance:Multiple Independent Loops. Normally, one fluid submodel (with one reference point) should be used per loop. Such usage results in significant computational efficiencies. However, it is also possible to place all otherwise independent fluid loops within a single fluid submodel as long as *lumref* is nonzero and as long as the fluid loops are never interconnected (not even with closed valves or zero-flow MFs). BALTPL will then adjust the pressures in the loops independently.

Restriction and Guidance:Usage in Transients. BALTPL should be avoided in transients: released (RELLMP) tanks should be used instead to govern the loop pressures. Unfortunately, in some systems it may be computationally expensive to use enough tanks to properly represent distributed mass: small time steps may result that are unacceptable. Therefore, BALTPL does not prohibit its own usage in STDSTL and transients.

However, if BALTPL *is* used in transients, junctions should be used instead of tanks for non-reference lumps, and the user must take extra measures to assure that flow-through reference lumps are not absorbing nor releasing excessive energy. The BALTPL adjustments will be made per time step, but are otherwise time-independent. Adjustments may lead or lag the rest of the system excessively, resulting in significant energy errors. The last column of the LMPTAB output lists energy imbalances in units of power: the user should repeat runs using different speed values and reduced time steps (via the control constant DTMAXF) until the acceptably small energy errors are achieved in the reference lump.

7.11.9.9 Chemical Equilibrium and Reaction Rate Utility

A utility routine, EQRATE, is available to help drive the reaction rates (XMDOT) toward their equilibrium values, where equilibrium concentrations (mass fractions) are specified by the user, perhaps as calculated from a call to a chemical equilibrium code as a function of current temperature and pressure. EQRATE may also be used to meet a specified reaction or combustion efficiency in terms of the residual amount of limiting reactant (e.g., uncombusted fuel). As with any XMDOT manipulation, the scope of this routine is limited to tanks in transients (or in STDSTL).

A support routine, EQRATESET, is available for modifying control parameters that will apply to all subsequent EQRATE calls.

The use of these two routines will be described using narrative examples following the formal definition of their calling parameters.

Subroutine name: EQRATE

Calling Sequence:

```
      CALL EQRATE (modnam, idt, XS(IC), XN(IC), NS,
     +          IDS(IC), AS, RATE)
```

where:

`modnam` .... fluid submodel name, in single quotes, containing tank
`idt` ..... user tank ID
`XS(ic)` .... SINDA (integer count) style array reference, for array containing desired species mass fractions
`XN(ic)` .... optional SINDA (integer count) style array reference, for array containing stoichiometric numbers for a reaction to be obeyed by the resulting XMDOT values.
`NS`..... number of reacting species (the number of species listed in XS, XN, and IDS). May be a subset of all available species within this submodel.
`IDS(ic)` ... list of fluid IDs corresponding to the species in XS and XN, or zero if AS is used instead
`AS`..... alphabetic string (in single quotes, or character string) of letter IDs of species corresponding to the species in XS and XN, or an empty string (‘ ‘) if IDS is used instead
`RATE` ..... control parameter for the resulting reaction rate. Use unity (1.0) or see discussion for more details on this parameter.

Example:

```
CALL EQRATE('SJA', 1960, A100, A300, 3, 0, 'OHW', 1.0)
```

Subroutine name: EQRATESET

Calling Sequence:

```
CALL EQRATESET (pname, value)
```

where:

`pname` . . . . . parameter name in single quotes: either ‘XLOW’ or ‘DTMIN’
`value` . . . . . input real value to assign the named parameter

Examples:

```
CALL EQRATESET('DTMIN', 1.0e-3)
CALL EQRATESET('XLOW', 1.0E-4)
```

Using EQRATE and EQRATESET

Each call to EQRATE applies to a single reaction within a single tank. Each time EQRATE is called, the XMDOT values for the named species are updated.

EQRATE should only be called from within FLOGIC 0. If it is called from within STEADY, or if NS is less than or equal to 1, no action is taken and no warning is produced.

XS() is a SINDA array* containing the mass fractions of NS species representing desired equilibrium mass fractions: the main purpose of EQRATE is to determine the values of XMDOT that will drive the tank towards this state in the least number of time steps. Mass fraction (XS) values of zero are not allowed, and values greater than the lower limit XLOW (documented below) are recommended.

The order of species within the XS() array may be specified by *either* the fluid ID number, via the integer SINDA array IDS(), or by the letter identifier as represented by the character string AS (in single quotes). If the AS method is used, the IDS() argument should be zero. If IDS(1) is between 1 and 9999, the AS argument is ignored. Examples of both methods are provided below.

RATE influences the time constant for the corrections, and are explained toward the end of this section. Additional control factors may be changed via an optional call to EQRATESET, as documented later in this subsection.

XN() is a SINDA array containing optional stoichiometric numbers used to assign a chemical equation to the process. The usage of XN() is explained at the end of this section. If the argument XN() is input as zero (real or integer) in the call to EQRATE, this signals it is not currently being used.

* “XS()” and “XS” will be used nearly interchangeably in this section. The parentheses are added to emphasize that an array is being used, but they are occasionally omitted if they might cause confusion with other punctuation or notation.

Note that XS(), XN(), and IDS() are SINDA arrays: they are expected to be input in IC (integer count) form rather than DV (data value) form. The actual arrays can be larger than the NS value specified, in which case only the first NS values are used.

Any species present in the submodel but not listed in IDS() or AS will be left alone (zero XMDOT); NS may be less than or equal to the number of species in a submodel. If NS is less than the number of species present, note that *XS is interpreted as the fraction of reacting species.* Specifically, XS describes the total current mass fraction compared to all other reacting species, excluding those not reacting (i.e., those not listed in IDS or AS). In other words, $\Sigma_{i=1,NS}(XS_i)=1.0$ is normally assumed, and if the XS values do not sum to unity within a small tolerance, the EQRATE routine will err off.* This also means that, in a gaseous mixture, XS will not be equal to the corresponding XG at equilibrium unless all species in that mixture are reacting (i.e., listed in IDS or AS).

Since the primary purpose of calling EQRATE is to set the XMDOT values for the species listed, any other adjustments to XMDOTs should be made *after* EQRATE is called. This restriction does not refer to the XMDOT of species that are not listed in the current EQRATE call: the XMDOT values for those species are not altered by EQRATE, so their XMDOT values can still be set by the user.†

Equilibrium Concentration Example—Consider a gaseous mixture of O_2, H_2, and H_2O (as species ‘O’, ‘H’, and ‘W,’ respectively). If NS=3, then all of these fluids are reacting. In this case, if AS = ‘OHW’ and XS is specified in ARRAY data as real SINDA array #100:

```
100 = 0.02, 0.01, 0.97
```

this means that, at equilibrium and with no condensate forming, the mixture will be driven to contain XGO=0.02, XGH=0.01, and XGW=0.97. The EQRATE call for tank 1960 in submodel *stanjan* might appear as:

```
CALL EQRATE('stanjan', 60, A100, 0.0, 3, 0, 'OHW', 1.0)
```

Note that the mass fractions specified in XS() do not need to represent any chemical equation: molar proportions might be considered in arriving at these mass fractions, but they are not required in the calculation of the resulting XMDOT (species mass production or extinction rate). In other words, mass and energy will be conserved, but *without information regarding the chemical reaction, element masses will not be conserved*. The XN() option, explained below, allows the user to specify the underlying chemical reaction involved, and constrains EQRATE to return appropriate XMDOT values that do conserve element masses.

---

* If the XN() chemical equation option is used, the user may chose to ignore the final mass fraction of selected species by specifying them as negative XS values, as explained later. In this case, $\Sigma_{i=1,NS \text{ for } XSi>0}(XS_i) < 1.0$, meaning that the specified fraction of target species may be less than unity.

† Multiple EQRATE calls can be made for the same tank, if there are different unrelated equilibrium reactions involved. However, in that case no overlap should exist between the two calls (i.e., no species should be common to both lists) or else later calls will erase the effects of prior calls.

A more complete listing of the input file for the above example is provided below, including the alternative use (commented out) of the IDS() array instead of the AS string:

```
HEADER FLOW DATA, STANJAN, FIDO = 8732, FIDH = 8704, FIDW = 6070
 ...
HEADER ARRAY DATA, STANJAN
C these are often calculated by an equilibrium solver like STANJAN
      100 = 0.02, 0.01, 0.97
c list of fluid identifiers for IDS() option:
      200 = 8732, 8704, 6070
 ...
HEADER FLOGIC 0, STANJAN
 ...
c using a list of species letters (AS method):
      call EQRATE('STANJAN', 1960, A100, 0.0, 3, 0, 'OHW', 1.0)
c OR using a list of FPROP DATA identifiers (IDS method):
c     call EQRATE('STANJAN', 1960, A100, 0.0, 3, NA200, ' ', 1.0)
c
c the species can be listed in any order, as long as XS is consistent
c with either AS or IDS.
```

Equilibrium Concentration, with Nonreacting Species—To continue the above example, assume that N_2 is introduced as nonreacting species 'N' such that XGN=0.5 is preserved (perhaps as an initial condition in a closed tank, or via steady ingress of nitrogen). In this case, NS=3 and XS() array values should still sum to 1.0, not 0.5. Given the above inputs to EQRATE (with no change to array 100), the eventual gas fractions would be XGO=0.005, XGH=0.01, XGW=0.485 assuming nitrogen (as the uninvolved species) took up the remainder at XGN=0.5.

Condensible/Volatile Species—If a single species currently exists in two phases, the XS value for that species refers to the total mass fraction in both phases, compared to the masses of all other reacting species. In the above example, assume that water (species 'W') condenses and that nitrogen is absent. Assume also that the temperature and pressure are such that the tank contains 90% liquid water by mass (meaning XL=0.1). If, once again, XS() is give by the same array #100 as listed above, then the final equilibrium mixture would satisfy the following relationships:

```
0.02 = XGO*XL                    so XGO = 0.2
0.01 = XGH*XL                    so XGH = 0.1
0.97 = XGW*XL + XFW*(1.0-XL)     so XGW = 0.7 (XFW=1)
```

If a chemical equilibrium routine or program is used to produce XS that tracks each condensible phase separately (as separate "species") then the mass fractions of each such phase should be added together to yield the mass fraction value that is passed into EQRATE as that species' cell in the XS() array.

Control Parameters—Because the time step cannot be precisely known ahead of time, and because conditions within the tank can change over the next time step (e.g., the temperature, pressure, fraction of nonreacting species), *the XMDOT rates that are predicted by EQRATE are necessarily approximate*, and will tend to drive the mixture towards equilibrium (or the desired reaction efficiency) over the course of several time steps: *equilibrium will not occur in one time step*.

The parameters RATE and DTMIN are used to determine the speed at which the reaction will proceed. The RATE parameter is an argument to the routine EQRATE since it can be adjusted per tank, or per reaction. The DTMIN parameter is global in application, and is set for all EQRATE calls via a call to EQRATESET (see below), with a default of 3.6E-6 seconds (1.0E-9 hours).

EQRATE calculates XMDOT rates taking into account both the current state of the tank as well as the rate of ingress and egress. The RATE parameter has the greatest effect when the resulting XMDOT values are dominated by the current state of the tank (meaning large volume or long residence times), and it has the least effect when XMDOT values are dominated by the rate of species flowing in to or out of the tank (meaning small volume or short residence times). The recommended starting value of RATE is unity (1.0), which will cause the reaction to reach equilibrium over several time steps by attempting to diminish the error (difference between current and desired mass fractions) by 50% each time step, assuming the time step is constant. A higher value of RATE will reach equilibrium in less model time, but it may be unstable or cause small time steps, especially if the input value exceeds two (2.0). A smaller value of RATE will diminish the effects of calling EQRATE, increasing the time constant for achieving equilibrium by performing smaller adjustments per time step.

The DTMIN parameter is used to set the smallest time constant for corrections. This value can be set at zero if no such limit is to be imposed, but even with RATE=1.0 the time step can become small if reactions are fast, so a reasonable lower limit (such as the default value of DTMIN represents) should be provided to keep XMDOT values from growing too large. The following call to EQRATESET (which can be made at the start of OPERATIONS) increases the DTMIN time constant to 1.0E-3 for all subsequent calls to EQRATE:

```
CALL EQRATESET('DTMIN',1.0E-3)
```

An additional control parameter is used to set the smallest allowable mass fraction of a species, XLOW, which defaults to 1.0E-3 (0.1% by mass). For example, the following lowers this fraction to 1.0E-4:

```
CALL EQRATESET('XLOW',1.0E-4)
```

If a species is approaching or is less than XLOW, reaction rates will be slowed as needed.

Unless the XN() option (described next) is employed, *then no value in the XS array should be set below XLOW.* However, if this situation were to arise, then the resulting simulation would be *inefficient*, but not *illegal*.

XN() Option: Prescribing Chemical Equations—As was mentioned above, if the XS() array is input but the XN() option is not, then the XMDOT rates set by EQRATE will drive the species mass fractions toward the specified XS() values, but these XMDOT rates will not conserve the mass of each element since FLUINT does not know what elements are contained within each species: the chemical equation has not been supplied. The XN() option provides a means of specifying such an equation, and this applies a significant constraint on the resulting XMDOT values.

To continue the above example, the following chemical equation is applicable:

$$H_2 + ½O_2 \rightarrow H_2O$$

or equivalently:

$$2H_2 + O_2 \rightarrow 2H_2O$$

The stoichiometric numbers in the first example are 1.0, 0.5, 1.0, and in the second example they are 2.0, 1.0, 2.0. These numbers can be supplied as the XN array, according to the order of species listed in either IDS() or AS.

To use the XN array, either the products or reagents must arbitrarily be assigned a negative value so the code can tell one side of the equation from the other. Otherwise, only the relative size of the stoichiometric numbers is important, not their absolute value. For example, assuming the order of the species is given (per the continuing example) as AS = 'OHW', the following values for the XN array (real SINDA array #300) are all equally valid for describing the above chemical reaction:

```
300 = 0.5, 1.0, -1.0
300 = -0.5, -1.0, 1.0
300 = 1.0, 2.0, -2.0
300 = -1.0, -2.0, 2.0
```

The resulting EQRATE call would appear as:

```
CALL EQRATE('STANJAN', 60, A100, A300, 3, 0, 'OHW', 1.0)
```

When XN() is applied, the EQRATE routine must try to push the contents of the tank toward the prescribed values in XS(), but it is constrained to do so while preserving the correct molar ratios.* In other words, a specific relationship between the resulting XMDOT values will be applied if stoichiometric numbers are supplied. The XMDOT rates will follow the prescribed chemical reaction forward or backward as needed to best reach the prescribed values of XS().

However, the imposition of this additional constraint often means that the exact prescribed values in XS() can no longer be met. Instead, the code will attempt to achieve a stable best fit to those values. In the above example, instead of achieving XGO=0.02, XGH=0.01, and XGW=0.97 (based

* Internal calls to VMOLWPT are made for this purpose.

on the XS values), if XN() were also supplied, then the final state might instead be XGO=<trace>, XGH=0.3, and XGW=0.7. In other words, if O_2 is in short supply, as much water as possible will be produced, but that might leave excess H_2 (more than had been prescribed).

In the above case, the XGO will be driven to hover near the value of XLOW, the minimum mass fraction (defaulted to 0.001 if not reset by a call to EQRATESET), causing a limit on the value of the resulting time step and reducing the reaction rates. *The XLOW parameter is always applicable, but it is more likely to be encountered when using the XN() option.*

XN() Option: Reaction or Combustion Efficiency—If the XN() array is provided, then the user has the additional option of telling EQRATE which species' concentrations (i.e., XS values) are important to achieve, and which species' concentrations can be any value that is needed to satisfy the other inputs based on current conditions. If the XS value of any given species is negative (zero is still not allowed), this signals the routine that there is no prescribed value for that species, and that any value is as good as any other: the concentration of that species ceases to be a constraint. In this case, the sum of positive XS values will be less than unity.

The mass fraction of at least one species must be prescribed (i.e., must be positive) in the XS array.

For example, consider that fuel (H_2, to continue the above example) were limiting the reaction: a lean mixture was being combusted. If the XN() array were supplied, then the user might choose the resulting residual fraction of H_2, and let the O_2 and H_2O fractions arrive at whatever values are consistent with that specified H_2 fraction, the prescribed chemical reaction, and the flow in and out of the tank (if any). For instance, if the residual fuel fraction were chosen to be 0.5% (by mass), then continuing to use the same species order AS='OHW' that was used in the above examples, the XS() array would be entered as:

```
100 = -1.0, 0.005, -1.0
```

Note that negative values of XS are only legal if the XN() array is provided, and that zero values of XS are never legal.

Recall that array cell values can be specified using registers or register-based expressions. Thus, the above example could be expanded to include a register HLEFT:

```
100 = -1.0, HLEFT, -1.0
```

If fluid were flowing into the tank in question, HLEFT could be calculated in logic before the EQRATE call (with perhaps an intermediate call to UPREG to update the array values):

```
HEADER FLOGIC 0, STAN
...
        HLEFT = ...     $ function of efficiency, inflows
        CALL UPREG      $ propagate change to XS() array
        CALL EQRATE( ... )
```

Thus, the XN() option, combined with negative values in the XS() array, provides a means of specifying a reaction or combustion efficiency: the fraction of completeness of a chemical reaction.

EQRATE Usage Summary

For ease of reference, this redundant subsection summarizes the uses of EQRATE described narratively above. EQRATE may be used to

1. Move the tank state towards prescribed equilibrium concentrations, perhaps as provided by an externally called chemical equilibrium solver that has taken into account chemical reactions:

   Use a full* XS() array, but omit the XN() array (provide zero as the XN argument in the call to EQRATE).

2. Move the tank state as close as possible to prescribed equilibrium concentrations, but constrain the resulting XMDOTs to follow a chemical reaction (as given by stoichiometric numbers):

   Use a full or partial XS() array, and include a full XN() array.

3. Make the tank state react according to an efficiency (fractional utilization) of a limiting reagent (usually fuel for combustion) in a chemical reaction (as given by stoichiometric numbers):

   Use a partial XS() array (perhaps containing only one positive entry), and include a full XN() array.

For instantaneous changes to a tank's state (or to a plenum providing boundary conditions), see CHGLMP_MIX (Section 7.11.1.2).

* "Full" means all positive values, whereas "partial" means some negative values are used in the XS() array.

7.11.9.10 Importing TankCalc Data For Partially Filled Vessels

Most pressurized vessels are cylindrical, though the end cap shapes can be more variable. When using the DEPTH parameter on HTP ties (Section 3.6.3.3), or when monitoring path end locations (Section 3.13.4), or when updating the CX/CY/CZ elevation of the liquid/vapor surface (Section 3.13), geometric calculations are required: find the height of the liquid surface as a function of the volume. This can be a complicated calculation when the liquid/vapor surface is within an end cap, or when the tank is tilted.

A program called TankCalc is available (http://www.arachnoid.com/TankCalc/) at no cost that performs these geometric analysis tasks. Descriptions of the math used within TankCalc are available online, so no attempt will be made to replicate that description.*

A utility (TankCalc60) is available to import TankCalc calculations into SINDA/FLUINT.

In addition to locating the liquid/vapor surface position, TankCalc also provides the liquid/vapor surface area, which is convenient when a pair of twinned tanks (Section 3.25, especially Section 3.25.9) is used to model the fluid within, since this output can be used directly as the tank AST parameter. If this twinned tank is connected to the wall using twinned ties, TankCalc also provides data that can be used to update the characteristic lengths (DCH) of each phase.

Details on the utility TankCalc60 are provided below. This section provides a usage overview.

1. Use TankCalc 6.0 or later. In the **Input** tab, choose the units based on the FLUINT model: *meters* for length and *meters*^{3} for volume if you use UID=SI (as shown in Figure 7-9). Otherwise, choose *feet* and *feet*^{3} for a model using UID=ENG.

   Other TankCalc inputs are documented along with that program, but it should be noted that the size and orientation (horizontal/vertical, and tilt) will normally be fixed per SINDA/FLUINT run, unless extra measures are taken to import multiple data sets (as separate SINDA arrays) into a single run.

   The default sensor position (0,0,0) represents the bottommost point of the tank, and is correct for SINDA/FLUINT usage, since the liquid (surface) height above the sensor is the parameter of interest.

2. On the **Compute** tab (Figure 7-10), press the **Data** button to see the results. Only use the **Height > Volume** mode (even though SINDA/FLUINT will be working in the reverse direction). Choose an adequate number decimal points to represent the data, and an adequate “Step Size” to represent the shape of the tank. The Step Size is the resolution of the liquid volume: the size of each analytic slice. These two values are chosen from the **Input** tab near the bottom.

   Work with the HTML **Table type** (see top of Figure 7-10) until you are satisfied with the data produced, then switch to the CSV mode (bottom of Figure 7-10) in preparation for the next step.

* Gratitude is owed to TankCalc’s author, Paul Lutus, for extending its functionality to better support thermal/fluid modeling uses such as are documented in this section.

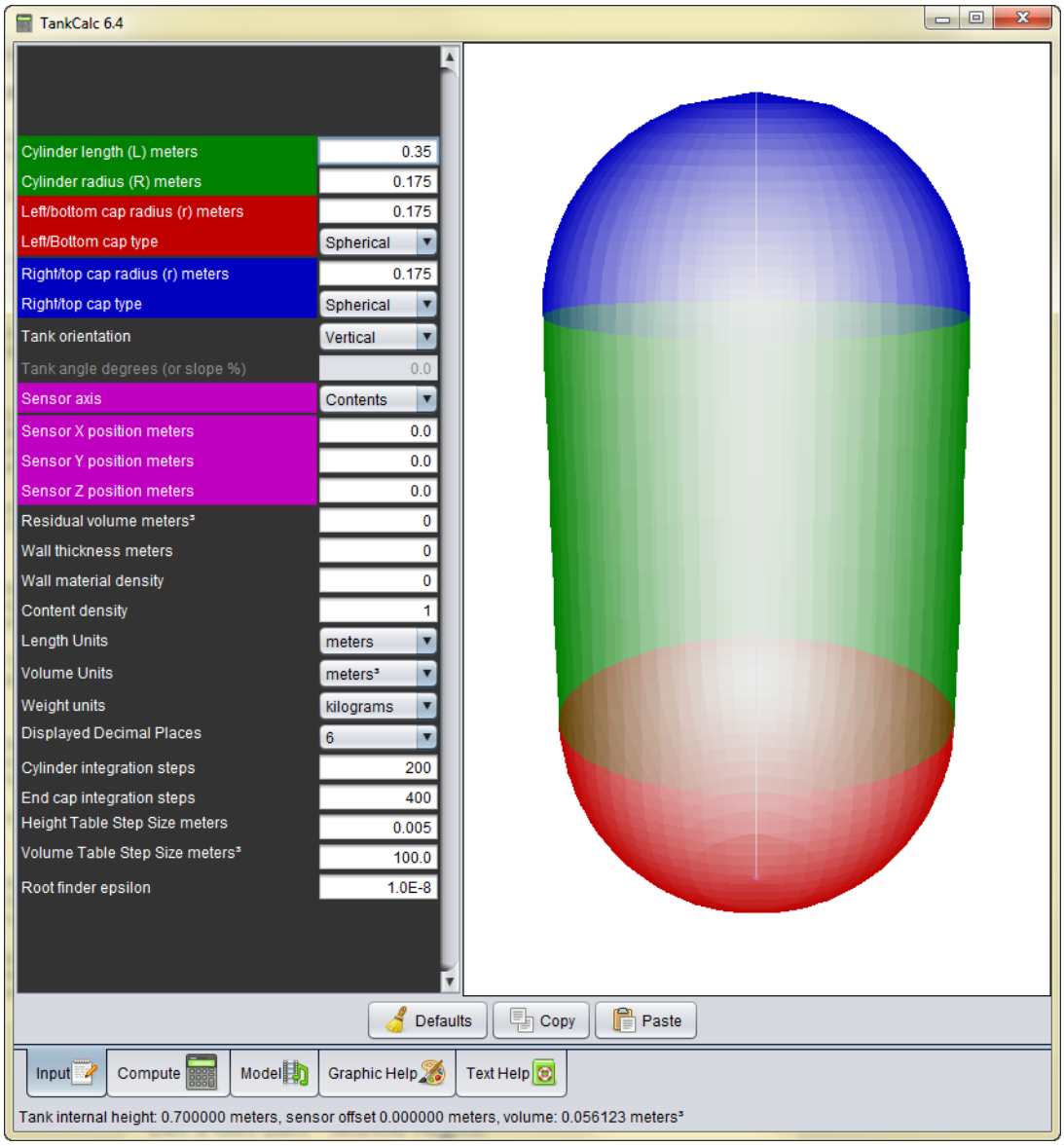


Figure 7-9 TankCalc Input Screen

3. In the CSV mode, press **Copy** to place the results in the Windows clipboard. Press <Ctl-v> in an empty text editor window (e.g., Notepad, Wordpad, Notepad++) to place the contents in this file. Edit the file to insert a SINDA array reference ("1000=" in Figure 7-11) and comment out other lines. ***Important:*** Shift the data over to the right by at least one space or <tab> to keep column 1 clear for SINDA commands.

   Note that in Figure 7-11, lines 17-117 have been hidden using Notepad++. Notepad++ is a freely available tool that makes it easy to edit columns and to shift the data to the right.

   Once finished with editing, save this file, and use the INSERT command (Section 2.5) to add it into the appropriate HEADER ARRAY block (Section 2.11).

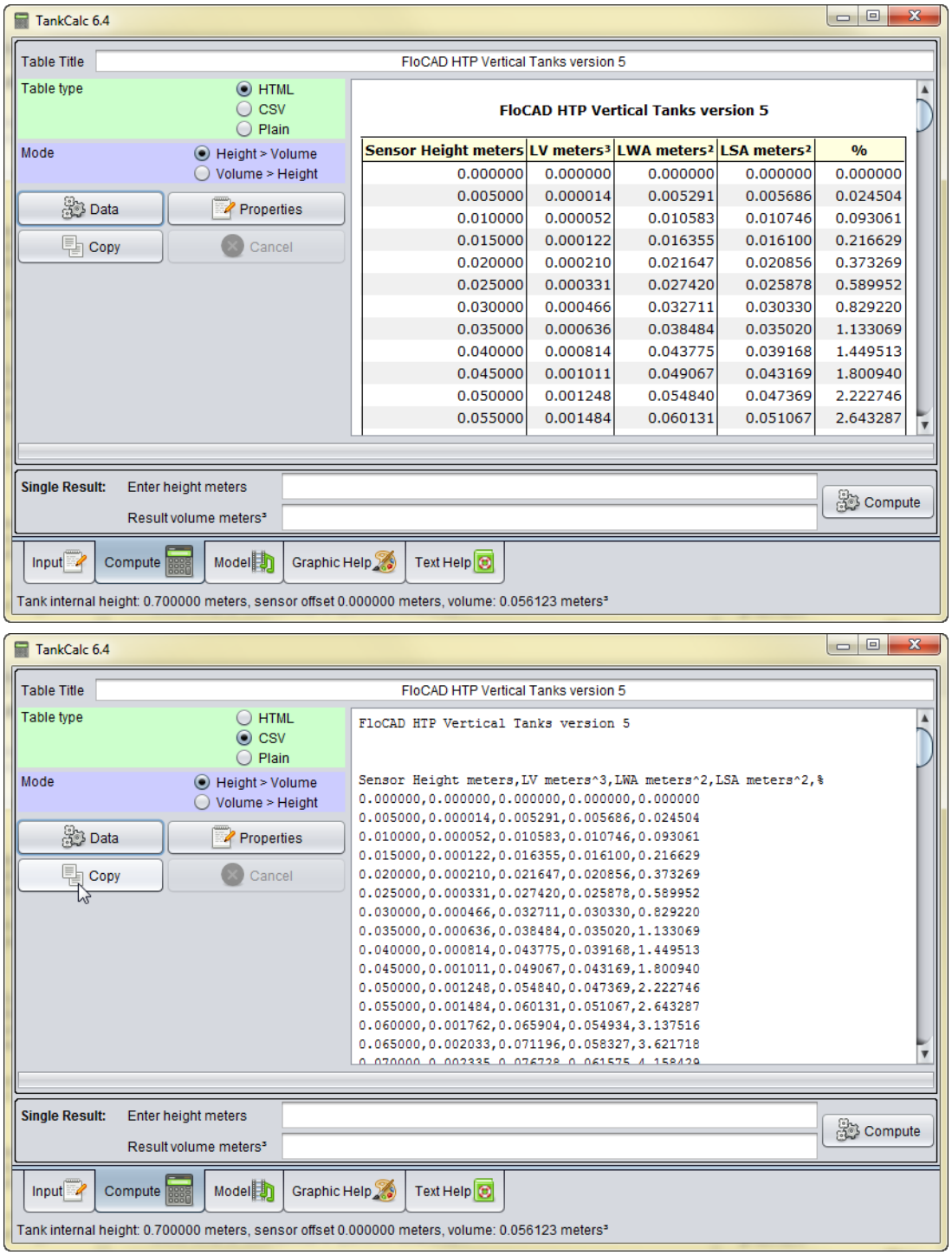


Figure 7-10 TankCalc Compute Screens (two views of the same data)

4. Create a call to TankCalc60 (documented below) in FLOGIC 0 that uses this array to calculate the relevant parameters as a function of current liquid volume. See Figure 7-12 as an example of Network Element Logic applied to a twinned tank's FLOGIC 0 block.

5. Make sure that tank elevations, the tank AST parameter, and HTP tie DEPTH and DCH parameters etc. are arranged to make use of these values as further documented below.

TankCalc5.inc

```
c FloCAD HTP Vertical Tanks version 5
c from TankCalc V6.0
c     Sensor Height meters,LV meters^3,LWA meters^2,LSA meters^2,%
   1000 =
      0.000000,0.000000,0.000000,0.000000,0.000000
      0.005000,0.000014,0.005291,0.005686,0.024504
      0.010000,0.000052,0.010583,0.010746,0.093061
      0.015000,0.000122,0.016355,0.016100,0.216629
      0.020000,0.000210,0.021647,0.020856,0.373269
      0.025000,0.000331,0.027420,0.025878,0.589952
      0.030000,0.000466,0.032711,0.030330,0.829220
      0.035000,0.000636,0.038484,0.035020,1.133069
      0.040000,0.000814,0.043775,0.039168,1.449513
      0.045000,0.001011,0.049067,0.043169,1.800940
      0.050000,0.001248,0.054840,0.047369,2.222746
      0.055000,0.001484,0.060131,0.051067,2.643287
      0.615000,0.052809,0.675884,0.070694,94.058778
      0.620000,0.053172,0.681657,0.067634,94.706850
      0.625000,0.053492,0.686948,0.064677,95.275170
      0.630000,0.053823,0.692721,0.061286,95.865562
      0.635000,0.054111,0.698013,0.058025,96.378282
      0.640000,0.054383,0.703304,0.054619,96.862484
      0.645000,0.054660,0.709077,0.050736,97.356713
      0.650000,0.054896,0.714369,0.047026,97.777254
      0.655000,0.055133,0.720141,0.042812,98.199060
      0.660000,0.055330,0.725433,0.038797,98.550487
      0.665000,0.055523,0.731206,0.034251,98.893921
      0.670000,0.055679,0.736497,0.029931,99.170780
      0.675000,0.055813,0.741789,0.025467,99.410048
      0.680000,0.055935,0.747561,0.020430,99.626731
      0.685000,0.056023,0.752853,0.015661,99.783371
      0.690000,0.056092,0.758626,0.010292,99.906939
      0.695000,0.056131,0.763917,0.005219,99.975496
      0.700000,0.056144,0.769449,0.000000,100.000000
```

Figure 7-11 TankCalc Output as SINDA Array

Translated Code Edit

Comment:

Submodel: FLOW

Flogic 0* | Flogic 1 | Flogic 2 | OUTPUT | Before Build | After Build | Post Solution | Operations

```
            if(vol#twin .eq. 0.0)then
                  vtest = (1.0-al#this)*vol#this
            else
                  vtest = vol#this
            endif

c using TankCalc array
            call TankCalc60(a1000, vtest, liq_height ,Asurface, AHTliq, AHTvap, DCHliq, DCHvap)

c to model coordinates
            cy#this =  liq_height
            cy#twin = cy#this
```

Figure 7-12 FloCAD Network Element Logic Example of TankCalc60 Usage

Calling Sequence:

```
      CALL TankCalc60( TCarray, VOLliq, HGTliq, ASlv,
     +        AHliq, AHvap, DCHliq, DCHvap)
```

where:

`TCarray` ... SINDA array containing TankCalc Version 6.0 or later CSV-formatted results (floating point input: SINDA "IC" array reference). If a version later than 6.0 produces different or additional data, edit this data to match the content shown in Figure 7-10. Illegal contents will result in an abort.

`VOLliq` .... Current liquid volume within the vessel (floating point input). For a homogeneous tank, this will be (1.0-AL)*VOL. For a twinned tank, allowing that bubbles or other voids will raise the liquid/vapor surface, the void fraction reduction term "1.0-AL" can be omitted and the volume of the liquid (primary) twin can be used directly as VOLliq. See Figure 7-12.

`HGTliq` .... Returned height of the liquid surface above the bottommost point (floating point output). The units are the same as that used for TankCalc, which should be consistent with the model's UID selection.

`ASlv` ..... Returned liquid/vapor surface area (floating point output). This may be used directly for the AST of a twinned tank representing the contents of the vessel.

`AHliq`..... Returned wall heat transfer area for the liquid: "wetted area" (floating point output).

`AHvap`..... Returned wall heat transfer area for the ullage (floating point output).

`DCHliq` .... Estimated characteristic length for a Nusselt number within the liquid volume (floating point output). See below for methods employed and usage.

`DCHvap` .... Estimated characteristic length for a Nusselt number within the vapor volume (floating point output). See below.

DCHliq and DCHvap values are intended for use as the DCH values of twinned ties in the ITAC=1 mode with XNUL>0.0.* In other modes, the ties share a common DCH value. The formula used for DCHliq is as follows:

```
DCHliq = (2.0 + 4.0*VOLliq/Vtotal)*VOLliq/(ASlv+AHliq)
```

A corresponding formula is used for DCHvap based on its volume and area. When a phase's volume takes up nearly the complete vessel, the above formula becomes DCH ~ 6V/A. For a cube, this dimension is the length. For a sphere, it is the diameter.

For the other phase that has been nearly completely displaced (i.e., when very little of a phase remains), the above formula becomes DCH ~ 2V/A. As the volume becomes planar (e.g., a thin disk), this characteristic dimension is the thickness of the plane.

* Use Network Element Logic in FloCAD or Sinaps, such as: DCH#this=DCHliq and DCH#twin=DCHvap. Otherwise, the DCH of the secondary tie cannot be specified via expression within FLOW DATA. Instead, it must be updated within logic.

Guidance: The DCH of an HTP tie is used in conjunction with the XNUL (Nusselt number) parameter for estimating heat transfer from wall to fluid in the single-phase mode. XNUL is by default negative, signaling that either NCSPHERE or NCCYLIN natural convection utilities are used, and that the absolute value of XNUL is taken as a multiplier rather than an input Nusselt number. In this case (XNUL<0), DCH is often chosen as a fixed value (e.g., the diameter of the sphere or the length cylinder, irrespective of the contents). However, it is also possible to consider applying NCSPHERE to each phase separately using the absolute value of XNUL as a multiplier, but using ITAC=1 to specify separate DCH (and user-controlled FQ) values.

For twinned tanks, ULT and UVT also represent a Nusselt number multiplier when negative. However, it should be noted that the characteristic lengths used in *that* case are somewhat different from the DCH values shown above. For ULT and UVT, the characteristic lengths are the volume of each phase divided by the liquid/vapor surface area. This correspond to VOLliq/ASlv (using the above nomenclature, or VOL#this/AST#this, in a lump reference) for the liquid, and (Vtotal-VOLliq)/ASlv for the vapor (or VOL#twin/AST#this, in a lump reference).

Guidance: Use the returned HGTliq to update the elevation of a single or twinned tank representing the vessel, noting that this distance must be added to the coordinate location of the bottom of the tank. For example, if the Z direction is “up” for gravity, then the CZ of the tank becomes HGTliq+*CZ_bottom* where *CZ_bottom* is the elevation of the bottommost point.

Note that FloCAD HTP ties can be used to automatically update the DEPTH parameter as a function of a symbol/register used to define the liquid height.

Also note that if port end locations are used, updating the liquid/vapor surface location will automatically update port calculations (Section 3.13.4).

Guidance: The wet and dry wall areas within the vessel (AHliq and AHvap) are mostly returned for informational purposes. They may be useful in setting twinned tie FQ factors,* or calculating other characteristic lengths.

* In twinned ties, a single AHT is split according to the FQ ratio, so do not use either AHliq or AHvap as the AHT value itself for a pair of twinned ties representing the entire wall. They could, however, be used to update the AHT of other ties.

7.11.10 Heat Exchanger Design and Simulation: System-Level

Detailed (discretized) models of heat exchanger passages can accommodate important effects such as two-phase flows, density changes due to pressure drops, complex flow passages, and fluidic transients. However, such models are appropriate only when the geometric details of the heat exchanger are known, when the heat exchanger itself is the focus of the analysis (and cannot be replaced by a simplified representation), and when the heat transfer and performance can be characterized: correlations are available.

The utilities described in this section are applicable when sizing heat exchangers, or when representing them using simplified component-level ("black box") methods. These methods are only marginally appropriate when one or more sides of the heat exchanger contains two-phase flow, or when transient responses are required, or when strong fluid property gradients exist (due perhaps to large pressure drops, to temperature-sensitive transport properties, etc.).

This section focuses on system-level representation of a heat exchanger: a lump representing the inlet (for either the hot or cold stream), a path representing the pressure loss, and a lump representing the outlet. Detailed/discretized models corresponding to compact heat exchanger methods are described separately in Section 7.11.11.

This section also focuses on FLUINT-based (fluid submodel) methods. Section 7.7.5 provides alternatives for SINDA-based (thermal submodel) models using one-way conductors to represent flow. Of course, it is possible to use a thermal submodel on one side of the heat exchanger, and a fluid submodel on the other side; either manual section may contain data or methods relevant to a particular modeling task.

HTUS or HTNS ties (Section 3.6.4) may also be appropriate choices for a LMTD (log-mean temperature difference) style solution when the wall is isothermal across the segment (i.e., C_{max} is infinite).

7.11.10.1 Calculating Heat Capacity Rates (Cmin, Cmax)

For single-phase flow with constant specific heat, the *heat capacity rate* is the product of mass flow rate times the specific heat: $C = FR*C_p$ which has units of W/K or BTU/hr-R (and is not to be confused with the nodal capacitance "C," which has units of J/K or BTU/R). For two-phase flow of a pure substance, C is infinite. For two-phase flow with at least some noncondensible or nonvolatile fluid, the effective C might be very large, but will be finite. For zero flow rates, the C is zero.

For a two-fluid heat exchanger, one side will normally have a larger heat capacity rate than the other: C_{max}, and the other side having the smaller C_{min}. Both of these values as well as the ratio ($C_r = C_{min}/C_{max}$) are important for heat exchanger sizing and simulation. If C_{max} is infinite, or if C_{min} is zero, then C_r is zero. A heat exchanger with $C_{min}=C_{max}$ ($C_r=1$) is valid as long as the heat capacity rates are not both zero or both infinite.

Unless the fluid properties are trivial, calculating the heat capacity ratio for one or both sides of the heat exchanger can be difficult. The CMINMAX routine is available to assist, as described next.

Subroutine Name: CMINMAX

Description: This subroutine is used to calculate the heat capacity rate for one or both sides of a two-fluid heat exchanger. The resulting values (C_{min}, C_{max}) can be used for calculating heat exchanger effectiveness, the overall heat transfer coefficient (UA_{tot}), and the NTU (number of heat transfer units) using perhaps the utilities listed in Section 7.11.10.2. The effectiveness is useful for top-level simulation of heat exchangers (Section 7.11.10.3), perhaps for the purpose of sizing or size verification. The UA_{tot} and NTU values are useful for sizing heat exchangers.

CMINMAX calculates the C (=FR*C_p) values for one or two FLUINT paths, which may exist in the same or different fluid submodels. These paths are presumed to represent the heat exchanger pressure drop, with the inlets of the heat exchanger at one end of each path, and the outlets at the other ends. The current flow rates of these paths are employed (multiplied by the DUPI factor, which must equal the DUPJ factor as a restriction). The specific heat is evaluated using a thermodynamic state (e.g., temperature and pressure) that is the average of the current up and downstream lump state.

The following restrictions apply, with violations causing a program abort:

1. The duplication factors (DUPI, DUPJ) can be nonunity, but they must be equal and should not be zero (otherwise C=0 will be returned).
2. The paths *can* be twinned (provide the ID of the primary twin if so), but the endpoint lumps *cannot* also be active twinned tanks.
3. This routine can be invoked from any logic block. If current values are required (because of changing flow rates and fluid states), FLOGIC 0 or 2 are reasonable choices for locating the call.

Calling Sequence:

```
CALL CMINMAX ('FSMN1', ID1, 'FSMN2', ID2, CMIN, CMAX)
```

where:

`FSMN1`	fluid submodel name, in single quotes, containing the first path (ID1)
`ID1`	path identifier for the first path
`FSMN2`	fluid submodel name, in single quotes, containing the second path (ID2). If this is the same submodel as FSMN1, a fake or empty argument (e.g., '') may be used.
`ID2`	path identifier for the second path. If there is no second path (i.e., if only one path's C value is required), then use ID2=ID1 (in which case CMIN will equal CMAX).
`CMIN`	returned minimum heat capacity rate, C_{min} (may be returned as zero)
`CMAX`	returned maximum heat capacity rate, C_{max} (may be returned as 1.0e30 if infinite)

Examples:

```
CALL CMINMAX('hotside',304,'coldside',134,ctest,dtest)
CALL CMINMAX('samesub',1495,'',1588,Cmin,Cmax)  $ same submodel
CALL CMINMAX('justone',555,'',555,Cone,Cone)  $ one path
```

7.11.10.2 Effectiveness and NTU (or UA_{tot}) Relationships

Heat exchanger effectiveness is defined as the actual transferred power compared to the maximum possible. For a single-phase heat exchanger with constant specific heats, the maximum power is equal to the C_{min} times the difference in *inlet* temperatures ($Q_{max} = C_{min}*[T_{h,i}-T_{c,i}]$), independent of the type of heat exchanger under consideration.

The total or overall heat transfer coefficient, UA_{tot}, is a measure of the complete heat exchange connectivity between the hot side and the cold side of a two-fluid heat exchanger, and is not to be confused with the UA of a tie. Indeed, the UA of a tie contains *only* the film coefficient on *one* side of the heat exchanger, whereas the UA_{tot} includes the effects of *both* film coefficients in series (perhaps averaged along the length), *plus* any other thermal resistances in the heat exchanger such as temperature drops through the wall and structure. UA_{tot} will therefore be a smaller value than the smallest tie UA.

The NTU, or number of heat transfer units, is related to the overall heat transfer coefficient divided by the minimum heat capacity rate: $NTU = UA_{tot}/C_{min}$.

If the ratio $C_r=C_{min}/C_{max}$ is known, then for a given type of heat exchanger, NTU can be calculated given effectiveness, and vice versa.

Subroutine Name: NTU2EFF

Description: This routine calculates heat exchanger effectiveness given the type of heat exchanger, the NTU, and the ratio of heat capacity rates for both sides of a two-fluid device. If the number of heat transfer units (NTU) is known, or if the total or overall heat transfer coefficient (UA_{tot}) for a heat exchanger is known and the C_{min} has been calculated (perhaps using CMINMAX) such that NTU can be calculated ($NTU = UA_{tot}/C_{min}$), then this routine can be used to calculate effectiveness. The effectiveness can then be used in system-level simulations (Section 7.11.10.3).

For crossflow heat exchangers, the user needs a second argument (integer KODE) to define which, if any, of the fluid streams is mixed. The streams are identified by whether they are the C_{min} stream or the C_{max} stream.* "Mixed" means that the fluid stream is not separated via baffles, tubes, etc. and can move laterally (perpendicular to the net flow velocity). For flow over a tube bank, for example, the tube side is unmixed but the external side is mixed. If instead the tubes were embedded in common fin plates such that the external fluid cannot mix laterally (in the tube-wise direction) as it flows between each tube, then both streams are considered unmixed.

For shell and tube heat exchangers, the KODE parameter defines the integer number of *shell* passes. The tube passes will then be assumed to be an even multiplier of the number of shell passes (e.g., 2*KODE, 4*KODE, 6*KODE, etc.). In other words, the tubes are assumed to enter and exit the same end of the shell, and single tube passes (i.e., the tubes exiting on the opposite end of the shell) are not supported. If multiple shell passes are employed (KODE>1), note that the input NTU is the *total* NTU for the whole heat exchanger: the NTU for *each* shell pass is equal to the total divided by the number of shell passes: $NTU_1 = NTU/KODE$.

Calling Sequence:

```
CALL NTU2EFF (NTU, CR, 'type', KODE, EFF)
```

where:

NTU . . . . . (floating point input) Number of heat transfer units, UA_{tot}/C_{min}. Must not be negative.
For shell and tube heat exchangers, this is the *total* NTU (not the NTU per shell pass, which is equal to NTU/KODE).

CR . . . . . Ratio of C_{min}/C_{max}, must be between 0.0 and 1.0, inclusive. These values can be calculated using a preceding call to CMINMAX (Section 7.11.10.1).

* The CMINMAX routine does not identify which path corresponds to the C_{min}, and which is the C_{max} stream. If it is not certain which is which, a call can be made to CMINMAX with ID1=ID2 such that either of the last two arguments in CMINMAX is the resulting C. In other words, CMINMAX can be used in a single-path mode if needed to determine which path is the C_{max} path, and which is the C_{min} path.

`type` ..... Heat exchanger type (in single quotes):
'PA' or 'PARALLEL' or 'PARALLELFLOW'
'CO' or 'COUNTER' or 'COUNTERFLOW'
'CR' or 'CROSS' or 'CROSSFLOW'
'SH' or 'ST' or 'SHELL' or 'SHELLNTUBE'

`KODE` ..... integer input (ignored for parallel or counterflow).
For shell and tube heat exchangers, this is the number of *shell* passes (between 1 and 1000). Single tube passes are unavailable: the number of tube passes will be 2*KCODE, 4*KCODE, etc.
For cross flow heat exchangers, this is a signal between 1 and 4:
1 ... both streams mixed
2 ... both streams unmixed
3 ... C_{min} stream unmixed, C_{max} stream mixed
4 ... C_{min} stream mixed, C_{max} stream unmixed

`EFF` ..... returned (real) value of effectiveness.

Examples:

```
CALL NTU2EFF( 3.0, Cmin/Cmax, 'counter', 0, etest)
CALL NTU2EFF( XNTU, Cratio, 'CROSSFLOW', 2, effect)
CALL NTU2EFF( 5.1, 0.5, 'ST', npasses, Eff)
```

Subroutine Name: EFF2NTU

Description: This routine calculates the number of heat transfer units (NTU) given the type of heat exchanger, the effectiveness, and the ratio of heat capacity rates for both sides of a two-fluid device. If the required effectiveness is known, and if the heat capacity rates (C_{min}, C_{max}) are known (perhaps using a prior call to CMINMAX), then the NTU and therefore the total or overall heat transfer coefficient (UA_{tot}) for a heat exchanger can be calculated (NTU = UA_{tot}/C_{min}). In other words, this routine can be used to calculate the size of the heat exchanger that achieves a prescribed effectiveness, noting that the pressure drop will usually be roughly proportional to the NTU or UA_{tot} values for a fixed cross section.

For crossflow heat exchangers, the user needs a second argument (integer KODE) to define which, if any, of the fluid streams is mixed. The streams are identified by whether they are the C_{min} stream or the C_{max} stream.* "Mixed" means that the fluid stream is not separated via baffles, tubes, etc. and can move laterally (perpendicular to the net flow velocity). For flow over a tube bank, for example, the tube side is unmixed but the external side is mixed. If instead the tubes were embedded in common fin plates such that the external fluid cannot mix laterally (in the tube-wise direction) as it flows between each tube, then both streams are considered unmixed.

* The CMINMAX routine does not identify which path corresponds to the C_{min}, and which is the C_{max} stream. If it is not certain which is which, a call can be made to CMINMAX with ID1=ID2 such that either of the last two arguments in CMINMAX is the resulting C. In other words, CMINMAX can be used in a single-path mode if needed to determine which path is the C_{max} path, and which is the C_{min} path.

For shell and tube heat exchangers, the KODE parameter defines the integer number of *shell* passes. The tube passes will then be assumed to be an even multiplier of the number of shell passes (e.g., 2*KODE, 4*KODE, 6*KODE, etc.). In other words, the tubes are assumed to enter and exit the same end of the shell, and single tube passes (i.e., the tubes exiting on the opposite end of the shell) are not supported. If multiple shell passes are employed (KODE>1), note that the returned NTU is the *total* NTU for the whole heat exchanger: the NTU for *each* shell pass is equal to the total divided by the number of shell passes: NTU_1 = NTU/KODE.

Calling Sequence:

```
CALL EFF2NTU(EFF, CR, 'type', KODE, NTU)
```

where:

`EFF` . . . . . Input heat exchanger effectiveness (floating point), must be between 0.0 and 1.0, inclusive

`CR` . . . . . Ratio of C_{min}/C_{max}, must be between 0.0 and 1.0, inclusive. These values can be calculated using a preceding call to CMINMAX (Section 7.11.10.1).

`type` . . . . . Heat exchanger type (in single quotes):
'PA' or 'PARALLEL' or 'PARALLELFLOW'
'CO' or 'COUNTER' or 'COUNTERFLOW'
'CR' or 'CROSS' or 'CROSSFLOW'
'SH' or 'ST' or 'SHELL' or 'SHELLNTUBE'

`KODE` . . . . . integer input (ignored for parallel or counterflow).
For shell and tube heat exchangers, this is the number of *shell* passes (between 1 and 1000). Single tube passes are unavailable: the number of tube passes will be 2*KCODE, 4*KCODE, etc.
For cross flow heat exchangers, this is a signal between 1 and 4:
1 ... both streams mixed
2 ... both streams unmixed
3 ... C_{min} stream unmixed, C_{max} stream mixed
4 ... C_{min} stream mixed, C_{max} stream unmixed

`NTU` . . . . . (floating point output) Number of heat transfer units, UA_{tot}/C_{min}. This value is often returned as 1.0e30 (infinite) if the requested effectiveness cannot be achieved.
For shell and tube heat exchangers, this is the *total* NTU (not the NTU per shell pass, which is equal to NTU/KODE).

Examples:

```
CALL EFF2NTU( 0.8, Cmin/Cmax, 'pa', 0, xtest)
CALL EFF2NTU( Eff, Cratio, 'CROSS', 4, XNTU)
CALL EFF2NTU( Eneeded, 0.74, 'SHELL', npass, Znumber)
```

7.11.10.3 Heat Exchanger Simulating or Sizing Utility

Once the effectiveness of a heat exchanger is known, whether by calculation or by design specification, this value can be used to simulate the heat exchanger's thermal performance* without regard to the fluid type (e.g., single-phase or two-phase) or the heat exchanger type (e.g., parallel or counterflow). Of course, the effectiveness need not be constant: it can be updated or modified as needed between iterations and time steps. The effectiveness might be calculated using a prior call to NTU2EFF (Section 7.11.10.2), for example.

Alternatively, the user might know the power that must be exchanged (e.g., "remove 1000W from the air/water stream") or the temperature (e.g., "cool the oil to 120°F"), and this requirement is being used to size the heat exchanger. In this case, the effectiveness can be calculated, and then used to size the heat exchanger perhaps using a subsequent call to EFF2NTU (Section 7.11.10.2).

This section documents HXMASTER, a general-purpose utility for sizing or simulating heat exchangers using a system-level (inlet/outlet) approach.

Subroutine Name: HXMASTER

Description: This routine calculates and optionally simulates the performance of two-fluid heat exchanger given one of the following specifications:

1. effectiveness
2. power transferred, or
3. one of the temperatures of the outlet streams.

HXMASTER is designed to be called in FLOGIC 0 (in either submodel's FLOGIC block if each side of the heat exchanger is located in a different fluid submodel). Given the current flow rates, inlet conditions, and pressure drops in two paths representing both sides of a two-fluid heat exchanger, HXMASTER can be used to:

1. Find transferred power or outlet temperatures given effectiveness (simulation mode)
2. Find transferred power and effectiveness given one outlet temperature (sizing mode)
3. Find effectiveness and outlet temperatures given transferred power (sizing mode)

HXMASTER assumes that the heat exchanger in question is completely defined by two flow paths, which may or may not exist in the same fluid submodel. The flow paths define the inlet and outlet of each flow stream (based on the current and not defined flow rate), and the lumps at their inlets represent the inlet temperatures and enthalpies to the heat exchanger (with phase suction taken into account as needed). The paths themselves and their parameters (e.g., dimensions) should be

* The pressure drop is handled independently based on the type of path chosen, and the size and other parameters of that path.

chosen independently to represent the pressure losses in each stream. Twinned paths may be used, but the end-point lumps must not be active twinned tanks. Path DUP factors can be nonunity, but they should be equal (DUPI=DUPJ for each path).

What HXMASTER considers to be a “heat exchanger” might actually be a subset of a larger heat exchanger, using multiple HXMASTER calls as needed (at least in the simulation mode) for each subsection. For example, consider an automotive air-conditioning condenser with cool air on one side and hot two-phase R134a on the other. The R134a enters superheated (single-phase), becomes two-phase, and ends subcooled (single-phase). Unless the global effectiveness is available, a single segment (inlet lump, path, outlet lump) is unlikely to be able to capture the strong gradients in heat capacity rates, local UA (NTU per segment), etc. In other words, each segment will exhibit a unique effectiveness, and requiring up to three HXMASTER calls to represent the whole condenser. If the condenser is subdivided finely enough, if suitable correlations exist for pressure drop and heat transfer, and if the heat transfer paths through the structure have been included, then HXMASTER is no longer needed for simulation purposes: a detailed model has been created.

Given that a heat exchanger is to be represented as “two inlets, two outlets, and two paths between them” there are still a wide variety of modeling choices to be made after HXMASTER has calculated the current operating point:

1. Specify a heat rate (transferred power), and let the exit temperatures adjust
2. Specify the exit temperatures, and let the heat rate adjust
3. Use a hybrid method (one exit defined by temperature, and the other by heat rate)

The first choice (heat rates on both exits) perfectly conserves energy. However, this heat rate will be held constant over the next time step (or relaxation step for a steady state solution): the temperatures of the lumps will not arrive at their expected values unless the paths’ flow rates both stay constant and unless the inlet lumps’ states also both stay constant. Oscillations or other instabilities can result, especially if the network is still changing rapidly. Nonetheless, if tanks are used to represent the outlets in transient solutions, this first choice is the best. Furthermore, while oscillations and instabilities are *possible* for steady states and for transients using junctions as exits, these problems are not common.

For the second choice (specified exit temperatures), if the exit lumps are junctions (or tanks during a STEADY solution), then special techniques can be invoked to hold the path output enthalpy* constant over the next interval (see “Guidance” below). Otherwise, heat rates are applied to the exit lump if it is a tank in a transient.

These special methods for outlet junctions (or STEADY tanks) are is used because they result in very stable solutions. However, changes in the paths’ flow rates and in the inlet states during transients can cause the actual transferred power rate to differ from the value that was calculated by HXMASTER at the start of the time step. Some lag may be experienced, in other words.

* Versions previous to V5.8 applied HTRLMP to make the output junction a “heater junction,” but this was disruptive if other ties or paths were attached to that outlet junction. Similarly, applying a CONSTQ FTIE between exit lumps is no longer required. The temperature of exit plena are no longer adjusted, either, since they may be used for other purposes: the user may set them using CHGLMP if desired.

HXMASTER may also be used in a “query mode” (MODE_S=0, meaning no simulations enacted), in which the user provides either the current power or one of the current outlet temperatures as input, and desires the current effectiveness as output. The effectiveness may then be converted into a UA_{tot} or NTU value using EFF2NTU. This mode would apply if the heat exchange rate or thermal resistance paths had been supplied separately, perhaps by a more detailed model, and the user wished to characterize the more complicated model using effectiveness-NTU methods.

Calling Sequence:

```
      CALL HXMASTER('FSMN1', ID1, 'FSMN2', ID2,
     +          EFF, TOUT1, TOUT2, POW12,
     +          MODE_I, MODE_S, IERR)
```

where:

FSMN1..... fluid submodel name, in single quotes, containing the first path (ID1)

ID1 ..... path identifier for the first path

FSMN2..... fluid submodel name, in single quotes, containing the second path (ID2). If this is the same submodel as FSMN1, a fake or empty argument (e.g., ’’) may be used.

ID2 ..... path identifier for the second path. ID2 cannot be the same as ID1.

EFF ..... input or returned effectiveness. If input (MODE_I=0), the value should be between 0.0 and 1.0 inclusive. Negative values and values greater than unity are tolerated, even though they are nonphysical.

TOUT1..... input or returned outlet temperature for path ID1, in user temperature units. If input (MODE_I=1), the value should be within the range of valid fluid properties, and will be internally limited if it causes Q_{max} of the heat exchanger to be exceeded.

TOUT2..... input or returned outlet temperature for path ID2, in user temperature units. If input (MODE_I=2), the value should be within the range of valid fluid properties, and will be internally limited if it causes Q_{max} of the heat exchanger to be exceeded.

POW12..... input or returned transferred power, positive from 1 to 2. Negative means the inlet for path 2 should be hotter than the inlet for path 1, but this is not enforced if POW12 is input (MODE_I=3). Units are either W (if UID=SI) or BTU/hr (if UID=ENG). The actual power will be internally limited if it causes Q_{max} of the heat exchanger to be exceeded.

MODE_I .... integer input mode selector:

0 ... EFF is input (TOUT1, TOUT2, and POW12 are output)
1 ... TOUT1 is input (EFF, TOUT2, and POW12 are output)
2 ... TOUT2 is input (EFF, TOUT1, and POW12 are output)
3 ... POW12 is input (EFF, TOUT1, and TOUT2 are output)

MODE_S . . . . . integer simulation mode selector:

0 ... do nothing (just return calculated values).

In this mode, either the user is querying (e.g., “What is the current effectiveness of this heat exchanger?”) or subsequent steps must be undertaken by the user to apply the resulting temperatures, powers, etc. to the model.

1 ... set exit temperatures if possible, and heat rate if not:

If either outlet is a junction or STEADY tank, the enthalpy flowing into this lump will be constant per iteration or time step. If there are no other ties or FTIEs or paths on that outlet lump, its temperature will be TOUT1 or TOUT2. If the lump is a tank, the heat rate into that tank will be augmented instead. If the exit lump is a plena or HLDLMP, the user may choose to adjust its temperature to TOUT1 or TOUT2 using CHGLMP.

2 ... set exit heat rate:

Add to the QDOT of the exit lumps according to POW12.

IERR . . . . . returned integer error flag: 0 if no error, 1 if property limits were encountered

Guidance: Constant enthalpy methods. For MODE_I = 1 or 2 and MODE_S=1, if the outlet lump is a junction (or tank in STEADY), the program internally sets the path’s outlet enthalpy rather than appending to the QDOT term. This increases solution stability, but it means that the addition to the QDOT term will not appear. In other words, if a call to STEADY is followed by a call to TRANSIENT and if a tank is at the outlet of the heat exchanger, its temperature might not change, but the QDOT value may jump up or down as the path’s enthalpy is released to vary and the heat exchanger power moves to the QDOT term.

In output routines such as LMPTAB, LMPMAP, L2TAB, and MACROTAB, the heat rate term is reported as the total effective rate, including heating or cooling for any lumps that are at the outlet of heat exchangers.

Example: Simulating a crossflow heat exchanger

```
c calculate the effectiveness in two steps: find path capacities,
c then use them plus assumption of cross flow to calculate effectiveness

      call cminmax('coldAir', 100, 'hotWater', 200, Cmin, Cmax)
      call ntu2eff(Xnum, max(0.,min(1.,Cmin/Cmax)), 'CROSS', 0, Effect)


c now use this effectiveness, and impose on the original paths' outlets

      call hxmaster('coldAir', 100, 'hotWater', 200, Effect,
     +  atest,btest,ptest, 0, 1, itest)
      if(itest .ne. 0) ... $ user error handling not shown
```

Example: Sizing a shell and tube heat exchanger (2 pass)

```
c find the effectiveness required to heat the oil to 200 degrees
c (this might be located in FLOGIC 0, driving the solution to the
c point required: sizing and simulating at the same time):

      call hxmaster('Steam', 345, 'ColdOil', 678, Effect,
     +  atest, 200.0, ptest, 2, 1, itest)
      if(itest .ne. 0) ... $ user error handling not shown

      call cminmax('Steam', 345, 'ColdOil', 678, Cmin, Cmax)
      Cratio = max(0.,min(1.,Cmin/Cmax))
      call eff2ntu(Effect, Cratio, 'SHELLNTUBE', 2, Xnumber)

c print total UA required per pass:

      write(nuser2,*)' UAtot per pass = ', (Xnumber/2.)*Cmin
```

7.11.11 Heat Exchanger Simulation: Detailed Approach

Subdividing heat exchangers into many axial segments is the most general-purpose approach. Such a model can handle single-phase and two-phase flow, transients, and strong property variations (including density drops and velocity increases in gases, specific heat variations in structures and fluids, etc.). However, discretized models can be slow to solve since resolution must be high enough (usually between 10 and 30) to capture all significant gradients, and so as not to underestimate overall heat transfer, which is a danger if the resolution is too low.

This section focuses on detailed modeling for the purposes of design verification, transient effects etc. Refer to Section 7.11.10 for descriptions of top-level (inlet/outlet) approaches, including sizing techniques.

Section 3.9.2 describes LINE and HX *duct macros*, which can be used to represent a flow stream in a two-fluid heat exchanger. The default heat transfer and pressure drop correlations correspond to smooth-walled and near-circular passages, and should be modified to account for the additional heat transfer enhancement structures that are typical of most heat exchangers.

This section describes how to override or modify the default correlation set to better correspond to the methods detailed in Kays and London Compact Heat Exchangers (CHX).* Note that single-phase turbulent flow is often implicitly assumed, and that special treatment may be required for laminar or two-phase flow, as explained below.

7.11.11.1 Compact Heat Exchangers: Colburn J Factor

SINDA/FLUINT normally uses the Nusselt number to characterize convective heat transfer. Compact heat exchanger (CHX) methods instead use the Colburn J factor (j_h):

$$j_h = St*Pr^{1/3}$$

where St is the Stanton number (defined in terms of FLUINT parameters, St = UB*AF/[FR*C_p]), and Pr is the fluid Prandtl number (Pr = μ*C_p/k) where μ is the fluid dynamic viscosity and k is the thermal conductivity.

Using FLUINT path parameter names, the Reynolds number is:

$$REY = FR*DH/(\mu*AF)$$

For CHX methods, the flow area AF should be specified as the minimum free-flow or core area, which equal to the constriction factor, σ, times the frontal area:

$$AF = A_{core} = \sigma*A_{front}$$

* W.M. Kays and A.L. London, Compact Heat Exchangers, 3rd Ed., 1984 (reprinted in 1998 with corrections), Krieger Publishing Co.

The hydraulic diameter DH is four times the hydraulic radius r_h:

$$DH = 4*r_h$$

The Colburn J factor is often a straight line on a log-log plot versus Reynolds number, meaning that it can be expressed as:

$$j_h = C_j * Re^{P_j}$$

If two points are taken from a graph (j_1 @ Re_1 and j_2 @ Re_2), then:

$$P_j = \ln(j_1/j_2) / \ln(Re_1/Re_2)$$
$$C_j = j_1 / Re_1^{P_j}$$

If the above power law is applicable, then the application to HTN, HTNC, and HTNC ties (as described in Section 3.6.6) is trivial:

$$CBD = C_j$$
$$UER = 1.0+P_j$$
$$UEC = UEH = 1.0/3.0$$

(The user is urged to consider nonzero UEV viscosity corrections as well, as described in Kays and London and other references.)

The above methods are intended for turbulent single-phase flows, but will be automatically extended by FLUINT to laminar and two-phase flows as first approximations. For laminar heat transfer (usually REY<1000 when the above methods are used), consider increasing the tie parameters XNUL or XNLM. Also, note that the built-in two-phase flow correlations are intended for near-circular ducts, and may require scaling (using perhaps UAM) to better fit available data.

If the relationship between the J factor and the Reynolds number cannot be represented using the above power law, then perhaps an array (interpolated using D1DEG1) is more appropriate. In this case, an HTU tie whose conductance is adjusted in FLOGIC 0 may be more appropriate. To use an HTU tie, set the AHT to be the heat transfer area per tie ("A_{heat}" in CHX parlance, divided by the number of ties). In FLOGIC 0, look up the current J factor (using perhaps a call to D1DEG1 with the path REY as the independent variable), then calculate the tie UB:

$$UB = j_h*C_p*FR/(AF*Pr^{2/3})$$

The specific heat C_p can be calculated using VCPV for gas or VCPF for liquid if it is not known or is variable. Similarly, the viscosity for the Prandtl number can be calculated using VVISCV or VVISCF.

The above discussion applies to specifying film coefficients on *one* side of the heat exchanger. The overall conductance UA_{tot} includes film coefficients on both sides plus the resistance of fins, container wall, and any bonding (e.g., epoxy or soldering). The remainder of this subsection addresses the inclusion of other thermal resistance terms.

A detailed finite element and/or finite difference model of the wall could be used, perhaps including the thermal capacitance in transients. Otherwise, a simplified representation can be made by including at least one node as a root temperature, and perhaps two nodes (one for each flow stream's root temperature) if the temperature drop across the structural components is significant (warranting a finite conductance between those two wall nodes).

When fins are used on at least one side of a two-fluid heat exchanger, and if the wall node represents the root temperature of that fin, then the fins need not be modeled explicitly. Rather, their effect can be included as a correction based on fin efficiencies.

Fin efficiencies are a function of the local convective environment, noting that the film coefficient "h" is "UB" for a tie and therefore can be found from preliminary runs. Textbook formulas are available for fin efficiencies. For example, for a rectangular fin of conductivity k, length L, and thickness t, in an environment with a convention coefficient of h:

$$m = \mathrm{sqrt}(2*h/(k*t))$$
$$\eta_f = \tanh(m*L)/(m*L)$$

This fin efficiency is based on the heat transfer area of the fin itself (A_{fin}), but the overall heat transfer area (A_{heat}) for the heat exchanger is different (often larger since it includes bare walls between fins, etc.). A corrected overall efficiency should be instead used:

$$\eta_o = 1 - (A_{fin}/A_{heat})(1 - \eta_f)$$

This correction factor can be applied as the UAM for a tie, or folded into the CDB value (CDB = $\eta_o * C_j$), or calculated as separate laminar (XNLM) and turbulent (XNTM) multipliers.

7.11.11.2 Compact Heat Exchangers: Fanning Friction Factor

SINDA/FLUINT normally uses the Darcy friction factor (f_d). Compact heat exchanger (CHX) methods instead use the Fanning factor (f_f), which is one fourth of the Darcy factor: $f_f = f_d/4$. For example, the Darcy friction factor for laminar flow in a circular tube is 64/Re, whereas the Fanning factor is 16/Re.

The definitions of Reynolds number, flow area, and hydraulic diameter are all the same as those listed above (Section 7.11.11.2). For reference:

$$REY = FR*DH/(\mu*AF)$$
$$AF = A_{core} = \sigma*A_{front}$$
$$DH = 4*r_h$$

The friction factor is often a straight line on a log-log plot versus Reynolds number, meaning that it can be expressed as:

$$f_f = C_f * Re^{P_f}$$

If two points are taken from a graph (f_1 @ Re_1 and f_2 @ Re_2), then:

$P_f = \ln(f_1/f_2) / \ln(Re_1/Re_2)$

$C_f = f_1 / Re_1^{P_f}$

Applying the above coefficients to tube or STUBE connector friction terms is not quite as straightforward as is applying the Colburn J factor coefficients to convection ties.

In theory, it is possible to take a "coefficient replacement" approach as was used in the prior subsection for ties:

IPDC = 0
FPOW = 1.0 + P_f
FC = -2*C_f*TLEN#this / (DL#up * AF#this^(2+P_f) * DH#this^(1-P_f) *μ^P_f)

where μ is the viscosity, perhaps replaced by a register containing that value.

The above approach is fast and stable, but does not treat laminar and two-phase flows (though this is not a concern for most applications). Setting IPDC=0 also eliminates the update of the REY parameter. Also, if f_f cannot be written as a simple function of C_f and P_f (i.e., if instead an array or function is needed), then more complex methods are required.

One possibility is to turn off the FC calculation and use an effective K-factor instead. To preserve the REY calculation, the FC is not set to zero so much as *effectively* zeroed by multiplying it with a small number:

FCLM = 1.0e-6
FCTM = 1.0e-6
FK = 4 * f_f * TLEN#this/DH#this

While f_f could be updated in logic as an arbitrarily complex function, it is instructive to instead demonstrate how it can be replaced within the above expression:

FK = 4 * C_f*max(1.,REY#this)^P_f *TLEN#this/DH#this

The "max()" function is introduced to avoid numerical problems should REY start at zero or become too small. However, other than this illustrative detail, *the above method is not recommended* because it does not deal with the loss of laminar and two-phase methods, and it is in fact less stable numerically than the FC approach.

A better approach is to go ahead and let FLUINT calculate its own friction factor, then use a *delta K-factor* to account for the different between that factor and the CHX-predicted f_f. This "corrected friction" approach at least provides approximate methods (and therefore numerical continuity) for laminar and two-phase flows. Given "f_d" as the FLUINT-predicted Darcy friction factor based on default correlations, the theory behind the "delta K-factor" method is to specify:

FK = (4*f_f - f_d) * TLEN#this/DH#this

Whether f_f is calculated as "C_f*max(1.,REY#this)^P_f" or as a function in logic or interpolated from arrays, the trick now becomes how to calculate f_d. For single-phase flows, FLUINT by default uses the function FRICT: f_d = FRICT(REYn,WRFn) for path n. However, since functions cannot currently be invoked within FLOW DATA expressions, that leaves two methods.

First, setting WRF=0.0 (smooth wall) means that f_d could be approximated as $0.316/Re^{1/4}$, resulting in:

```
WRF = 0.0
FK = (4*Cf*max(1.,REY#this)^Pf - 0.316/max(1.,REY#this)^0.25)*TLEN#this/DH#this
```

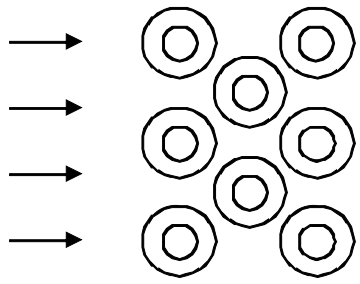
The second *and preferred method* is to exploit FloCAD or Sinaps "network logic*" placed in FLOGIC 0. Using registers Cof and Powf for C_f and P_f, respectively:

```
DTEST = FRICT(REY#this, WRF#this) $ can call a Fortran function in Network Logic
FTEST = Cof*max(1.0,REY#this)**Powf
FK#this = (4*FTEST - DTEST) * TLEN#this/DH#this
```

Finally, note that any entrance or exit losses (e.g., contraction to the core) should be included if they are not already included in the friction factor.

7.11.11.3 Compact Heat Exchanger Example

Surface CF-8.72(c) in Kays and London (p.265, Figure 10-85 in the 3rd Edition) is consists of parallel tubes with circular fins, with dimensions, parameters, and equivalent register names as defined in Figure 7-13.



Parameter	Value	Units	Register
D_h	4.425e-3	m	*dhyd*
σ	0.494	- (m^2/m^2)	*sig*
A_{heat}/Volume	446.	1/m (m^2/m^3)	*a2v*
A_{fin}/A_{heat}	0.876	- (m^2/m^2)	*f2t*

Figure 7-13 Example Problem for CHX Modeling

* Network logic uses expression-style syntax (e.g., "#this") but inside a block that will be turned into Fortran user logic. It provides a means of scripting complex operations for each network object (lump, path, etc.) without concern for identifiers. This means that these blocks can be simultaneously added to many elements at once. Refer to the FloCAD and Sinaps User's Manuals for more details.

Air flows over the outside in the finned section, and water flows inside the circular tubes. The length (the horizontal dimension in the diagram) and frontal area (the vertical dimension in the diagram times the distance into the page, or the length of the water lines) are left indefinite as registers *length* and *Afront*.

The water portion of the model is trivial: simple HX macros with the nodes representing the tube ID (and perhaps containing the mass of the structure). If the temperature drop across the tube is negligible (e.g., the tube is made from a high conductivity material such as copper), this same node may serve as a root node for the fins. Otherwise, a second node can serve as the tube OD and fin root node with the conductance between the ID and OD specified according to pipe properties.

If the fin efficiency is high enough, it can be assumed unity. Otherwise, it can be calculated based on textbook formula and using estimates for the air-side convection coefficient (either hand calculations or preliminary runs, or guessed and corrected later, etc.). This effectiveness can then be used to calculate the overall effectiveness per the methods detailed at the end of 7.11.11.1, using the value of the register *f2t*. These details are omitted since the focus of this example is on setting the path and tie parameters for the air-side heat exchange.

The length of the air-flow path (likely an STUBE connector) is TLEN=*length*, the hydraulic diameter is DH=*dhyd*, and its flow area is AF=*Acore=sig*Afront*. An HTN tie connects the air lump with the node representing the fin root and pipe OD.*

Two data points are available for the J factor and Fanning factor, at two different Reynolds numbers as listed below (as registers):

```
jh1 = 0.0175      $ Point one (at Re = 600)
ff1 = 0.06
re1 = 0.6e3

jh2 = 0.0058      $ Point two (at Re = 7000)
ff2 = 0.035
re2 = 7.e3
```

The coefficient and exponents for j_h and f_f can therefore be calculated (assuming linearity in the log-log plot):

```
powr = ln(jh1/jh2)/ln(re1/re2)                $ Pj
powf = ln(ff1/ff2)/ln(re1/re2)                $ Pf
coj = jh1/re1^powr                            $ Cj
cof = ff1/re1^powf                            $ Cf
```

Assuming that the viscosity of air is constant, it could be calculated in OPERATIONS (assuming lump 1 in fluid submodel “air” is representative):

```
viscA = vviscv(air.PL1-patmos, air.TL1-abszro, air.fi)
```

* Despite referring to this lump, path, tie, and node in the singular, this pattern can be repeated as needed to provide adequate spatial resolution, and these components may be contained within HX macros as well.

The viscosity could also be updated in FLOGIC 0 if it were temperature dependent, or if the average of lump and wall node temperature were used, etc.

The path and tie inputs might then appear as follows:

```
pa conn,1,1,2,    dev = stube
      dh = dhyd,  tlen = length,    af = Acore
      ipdc = 0,   fpow = 1+powf
c the following must fit on one line:
 fc = -2*cof*TLEN#this/(DL#up*AF#this^(2+powf)*DH#this^(1-powf)*viscA^powf)


t htn,1,1,wall.1,1,1.0
      uec = 1./3., ueh = 1./3., uer = 1+powr
      cdb = coj         $ perhaps corrected for overall fin eff.
```

Alternatively, the path could be specified as:

```
pa conn,1,1,2,    dev = stube
      dh = dhyd,  tlen = length,    af = Acore
```

with the following added to FLOGIC 0 via *Network Logic* in Sinaps or FloCAD:

```
      DTEST = FRICT(REY#this, WRF#this)
      FTEST = Cof*max(1.0,REY#this)**Powf
      FK#this = (4*FTEST - DTEST) * TLEN#this/DH#this
```

7.11.11.4 Offset Strip Fin Correlations

Correlations for offset strip fin (also known as lanced fin) heat exchangers are available.

These correlations were produced by Manglik and Bergles (1995), as referenced in Thermal Design* by H.S. Lee. They are applicable to both laminar and turbulent flow, and include a numerically smooth transition between the two regimes.

The reader is assumed to be familiar with the methods prevented in the prior subsections. This subsection serves as both a specific example of those methods as well as documentation of built-in subroutines for a popular heat exchanger configuration. Note that these calculations are meant to supplement a detailed thermal model of the fin structure, whether constructed in Thermal Desktop or TD Direct.

Figure 7-14 illustrates the configuration as well as documents the geometric arguments used in the subroutines listed below. This figure does not show the top and bottom plate that would enclose the fin stack, nor fin stacks for other fluids with which this stack exchanges energy.

* 2010 Edition, ISBN 978-0-470-49662-6

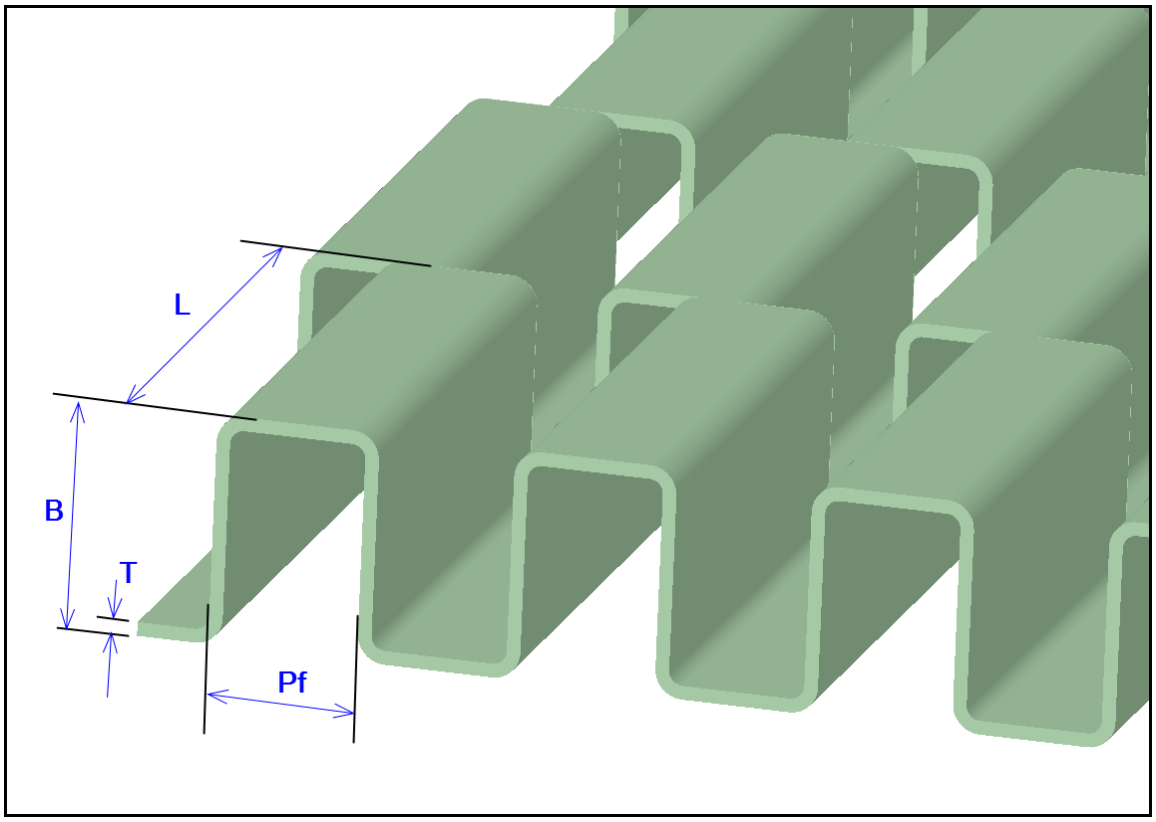


Figure 7-14 Offset Strip Fin with Defining Dimensions

Hydraulic Diameter:

```
CALL DH_OSF(Pitch, Hght, Thk, Sleng, DH_OUT)
```

where:

`Pitch`	Fin pitch (“Pf” in Figure 7-14), any consistent length unit
`Hght`	Segment height (“B” in Figure 7-14), any consistent length unit
`Thk`	Fin thickness (“T” in Figure 7-14), any consistent length unit
`Sleng`	Segment length (“L” in Figure 7-14), any consistent length unit
`DH_OUT`	Returned hydraulic diameter, in same length units as input

This is a one-time calculation, and so is best placed in OPERATIONS. It may also be more convenient to specify as a register or as a Thermal Desktop symbol, in which case the equation underlying this routine is presented below as an alternative:

```
DH_OUT = 4.0*(Pf-T)*(B-T)*L
         /(2.0*((Pf-T)*L + (B-T)*L + (B-T)*T) + (Pf-T)*T)
```

Colburn J Factor:

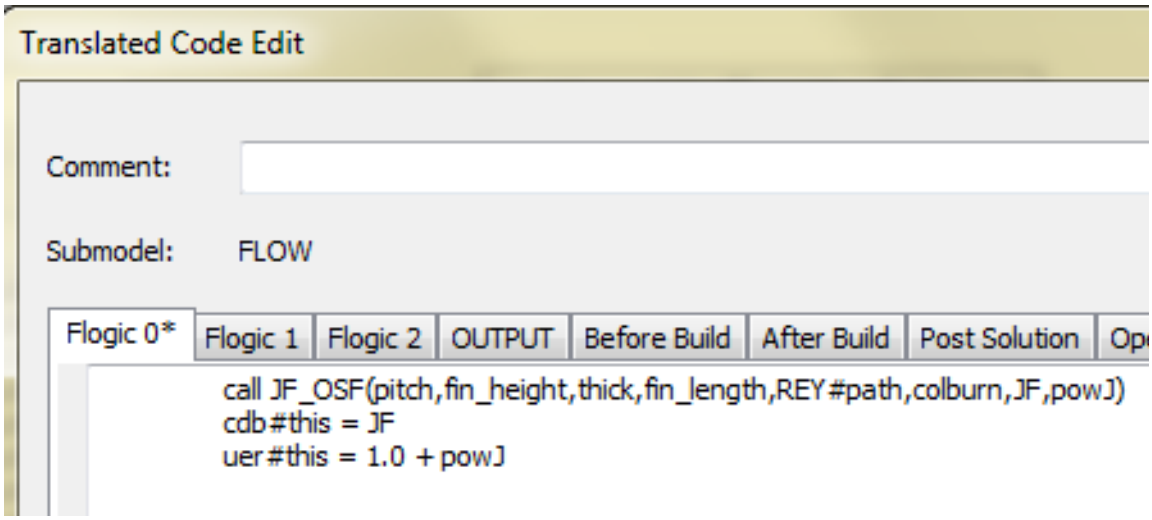
```
CALL JF_OSF(Pitch, Hght, Thk, Sleng, Rey, Jay, Jf, Jp)
```

where:

`Pitch` . . . . . Fin pitch (“Pf” in Figure 7-14), any consistent length unit
`Height` . . . . . Segment height (“B” in Figure 7-14), any consistent length unit
`Thk` . . . . . Fin thickness (“T” in Figure 7-14), any consistent length unit
`Sleng` . . . . . Segment length (“L” in Figure 7-14), any consistent length unit
`Rey` . . . . . Input current Reynolds number
`Jay` . . . . . Output current Colburn J value (usually for reference only)
`Jf` . . . . . Output current J factor in the equation: Jay = Jf*Rey**Jp
`Jp` . . . . . Output current J value exponent

This calculation should be updated for every tie to a wall within this fin stack. It relies on the REY value of the tie’s path,* and it returns the Jf factor and the exponent Jp for use in updating the tie’s CDB and UER values, respectively. (The Colburn J factor itself is also returned for reference, or for alternative uses.)

This utility is easiest to apply in each tie’s Network Element logic, as shown below (for FloCAD) using the methods described in previous subsections:



* While it is true that an HTNC tie has two paths, it should be adequate to use the Reynolds number for just the first path. This avoids the more cumbersome argument “0.5*(REY#path+REY#path2)” which would be needed to calculate an average Reynolds for centered ties.

Fanning F Factor:

```
CALL FF_OSF(Pitch, Hght, Thk, Sleng, Rey, Eff, Ff, Fp)
```

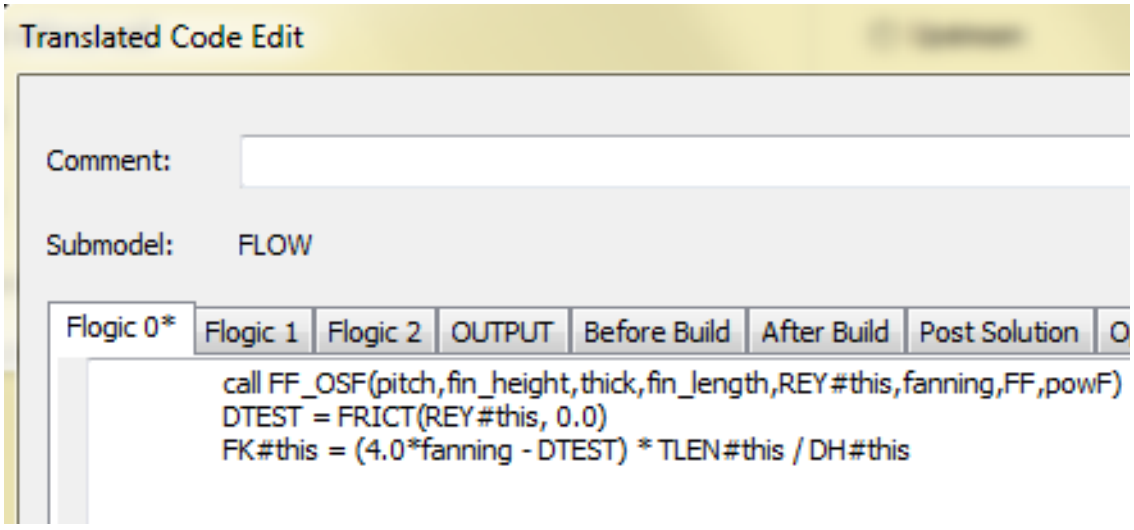
where:

`Pitch`..... Fin pitch ("Pf" in Figure 7-14), any consistent length unit
`Height` .... Segment height ("B" in Figure 7-14), any consistent length unit
`Thk` ..... Fin thickness ("T" in Figure 7-14), any consistent length unit
`Sleng`..... Segment length ("L" in Figure 7-14), any consistent length unit
`Rey` ..... Input current Reynolds number
`Eff` ..... Output current *Fanning* friction value (usually for reference only)
`Ff`..... Output current F factor in the equation: Eff = Ff*Rey**Fp
`Fp`..... Output current F value exponent

This calculation should be updated for every tube or STUBE within this fin stack. It relies on the REY value of the path, and it returns the Ff factor and the exponent Fp for use in updating the path's FK (K-factor) values. (The Fanning friction factor itself is also returned for reference, or for alternative uses.)

Recall that FLUINT (including the duct friction utility FRICT) normally uses the Darcy friction factor, and that $f_f = f_d/4$.

This utility is easiest to apply in each path's Network Element logic, as shown below (for FloCAD). Logic for a tube or STUBE is shown using the methods described in previous subsections:



7.12 Dynamic Register Utility Routines

By default, interrelationships between registers, parameters, and processor variables are resolved automatically at numerous points during the processor solution. The default update system is adequate for almost all modeling purposes. However, advanced users can augment, control, or replace this update system as needed, using utilities defined in this section.

To turn off all automatic updates (other than those performed in the Solver and the Reliability Engineering modules), state NOUPDATE in OPTIONS DATA (Section 2.7.2). Otherwise, the user can choose to leave the automatic updates in place, but either disconnect parameters and/or add extra (perhaps limited) updates as needed.

This section describes routines used to modify register values and expressions, to propagate the changes to the parameters that used them in their definitions, and to control this propagation. Refer also to Section 4.9.

Note: The length of this section demonstrates the ultimate control that the user is provided over the automatic updating of the model, but it can be also misleading. In reality, most users will find they do not need to employ this level of control. Most of the routines listed in this section are for special needs only.

7.12.1 Updating the Model

Changes to registers automatically propagate through the model to the parameters that used those registers or processor variables in their defining expressions. Also (or perhaps instead, if NOUPDATE is chosen), the user can choose to tell the program when to perform this propagation. The user has control over what data should be changed:

UPREG . . . . . updates all relevant parameters
UPREGT . . . . . updates only thermal submodel parameters
UPREGF . . . . . updates only fluid submodel parameters
UPREG1 . . . . . updates only a single specified parameter
UPREGC . . . . . updates a custom selected set of one or more parameters
FORCER . . . . . forces a complete update and reconnection of all parameters

Actually, even if only part of the network is updated, all internal spreadsheet data will be updated, but those parameters not allowed to change will be held constant during the spreadsheet propagation. For example, if UPREGT is called and a thermal parameter (say, the capacitance of node #3) uses a fluid processor variable (say, the volume of tank #33) in its definition, then C3 will change but VOL33 will be held constant, even if it itself is a parameter defined on the basis of registers or other processor variables. Calling UPREGF will then update VOL33 but hold C3 constant. Calling UPREG, all variables are updated and none are held constant.

In later sections, routines that temporarily disconnect (Section 7.12.2) and reconnect (Section 7.12.3) some or all of these automatic updates are described, providing the user further control over the process if needed. The user can even check to see if a connection exists for a given parameter

(Section 7.12.4).

The default automatic propagation internally calls UPREG at various points in the solution (typically after VARIABLES 1 and FLOGIC 1 but before the solution step). These internal calls to UPREG can be eliminated by choosing the NOUPDATE option in OPTIONS DATA (Section 2.7.2), leaving the user with complete control over the update process. The Solver and the reliability estimation routines (Section 5) also internally call UPREG, and these calls are not disconnected by the NOUPDATE option.

Caution: *A call to one of the UPREG family of routines will cause no changes to parameters unless the registers or processor variables defining those parameters have changed since the last update, even if the current value of those parameters does not correspond to their formulas because the user has overwritten them (perhaps due to a call to RESPAR or RESTAR or RESTDB, as explained in Section 4.9.7).* See FORCER below and Section 4.9.6.3.

Subroutine Name: UPREG

Description: This routine applies any changes that may have been made to registers to all possible parameters which used formula (registers or expressions containing references to registers or processor variables) in their definitions.

If no changes have been made to the value or expression of any register or parameter, then no updates are performed. (To force updates to be performed to assure consistency, use FORCER as described at the end of this subsection.) If changes have been made to *any* register, then *all* registers will be updated along with processor variables. UPREG will then update any parameters whose defining formula has changed as a result of a recent change in a register or processor variable.

UPREG is convenient to call from OPERATIONS, and may be called from any logic block. However, in keeping with the recommendation to update only thermal variables in VARIABLES blocks and only fluid variables in FLOGIC blocks, better results might be obtained by making a call to UPREGT (see below) in a VARIABLES 1 block and a separate call to UPREGF (see below) in a FLOGIC 0 block. However, note that neither UPREGT nor UPREGF updates global control constants nor named user constants, which are global (not owned by any submodel).

UPREG has no arguments.

UPREG is called internally by the Solver and by the reliability estimation routines, and it is called by default (unless NOUPDATE is specified) several times per solution step to propagate any changes users might have made in their logic blocks.

Calling Sequence:

```
CALL UPREG
```

Guidance: Because the routine is careful to avoid unnecessary updates, it incurs little cost by accidently calling it too frequently.

Caution: The updates performed by this routine are independent of updates performed by users in their logic. For example, if the temperature T of a boundary node had been defined by a formula such as “case*one,” then if either *case* or *one* had changed value, the T of the node would be reset, perhaps overwriting other operations performed by the user in his or her logic. But if the user had reset the temperature of that node independently in their logic, and neither *case* nor *one* had changed value, then no update of the nodal T would have been performed. If desired, parameters can be disconnected from being updated by UPREG and similar routines (see DEREG family of routines in Section 7.12.2). The FORCER routine can also be used to update parameters even if their defining registers have not changed.

Guidance: Depending on where the call to UPREG is made, changes may or may not affect the immediate time step or steady state iteration. For example, if the elements of an array had been defined using formula (registers or register containing expressions), and that array had been referenced by one or more SIV conductors, then a call to UPREG would only propagate to the G of those conductors at the end of VARIABLES 1 (or perhaps when GVTEMP is called by the user). As another example, changing the IPDC of an STUBE connector in FLOGIC 1 has no effect on the current solution step. *The rules regarding changing parameters are the same whether made independently by users in logic, or made using the more automated system based on registers.*

Guidance: When lump volumes are updated, an internal call to CHGVOL is made. When lump thermodynamic variables (PL, TL, XGx, etc.) are updated, and internal calls to CHGLMP will be made.

Subroutine Name: UPREGT

Description: This routine is identical to UPREG, except that updates are restricted to data in thermal submodels: all node, conductor, and source data. This update includes any numbered user constants or arrays owned by thermal submodels. Any other parameter is held constant during the propagation of updates. It does not update any fluid submodel data directly, unless those submodels refer to array data contained in thermal submodels. It also does not update named user constants, which are global, nor global control constants.

Although UPREGT can be called from any logic block, most often a VARIABLES 1 block will be the correct location.

Refer to the description of UPREG for more guidance and suggestions.

UPREGT has no arguments.

Calling Sequence:

```
CALL UPREGT
```

Subroutine Name: UPREGF

Description: This routine is identical to UPREG, except that updates are restricted to data in fluid submodels: all lump, path, and tie data. This update includes any numbered user constants or arrays owned by fluid submodels. Any other parameter is held constant during the propagation of updates. It does not update any thermal submodel data directly, unless those submodels refer to numbered constants or array data contained in fluid submodels. It also does not update named user constants, which are global, nor global control constants.

Although UPREGF can be called from any logic block, most often an FLOGIC 0 block will be the correct location.

Refer to the description of UPREG for more guidance and suggestions.

UPREGF has no arguments.

Calling Sequence:

```
CALL UPREGF
```

Subroutine Name: UPREG1

Description: This routine is identical to UPREG, except that updates are restricted to the single parameter provided as an argument. Any other parameter is held constant during the propagation of updates.

UPREG1 can be called from any logic block where a change to the requested parameter is appropriate.

Refer to the description of UPREG for more guidance and suggestions.

Calling Sequence:

```
CALL UPREG1 (param)
```

where:

`param.....` any legal reference to a SINDA/FLUINT parameter that was defined by a formula (expression containing registers or processor variables). This parameter may be subject to translation (e.g., “T400”), or perhaps has already been translated by the user using dynamic translation routines (e.g., INTNOD).

Caution: “A single parameter” does not mean an entire array. UPREG1 functions only on a single data point within an array (see the second example below). To update all the elements in an array, refer to UPREGC.

If the parameter passed into UPREG1 was not defined using a formula, a caution will be issued and no changes made. Similar cautions will be issued if an update is requested but the parameter has been disconnected from being updated via a previous call to one of the routines in the DEREG family (see Section 7.12.2).

Caution: Note that a SOURCE DATA definition such as "100, imhot" defines a source of *imhot* to be applied to the Q of node 100, but d*oes not directly define the Q variable of node 100* since other sources may also be present on that node. Therefore, calling "UPREG1(Q100)" will result in a caution, since Q100 is not directly updated. Similarly, calling UPREG1(G300) will result in a caution if conductor 300 is a SIV conductor or some other automatically updating conductor, since the G value of that conductor was not itself directly input as a register-containing expression. Refer instead to UPREGC (below) for making custom updates on a single node, all the sources on a node, all the factors affecting a single conductor (of whatever type), etc.

Examples:

```
C UPDATE THE TEMPERATURE OF PLENUM 300:
      CALL UPREG1(TL300)
C UPDATE THE ITESTth CELL IN ARRAY 23 OF SUBMODEL "reese:"
      CALL UPREG1(reese.A(23+ITEST))
C UPDATE A NUMBERED USER CONSTANT
      CALL UPREG1(XK3)
C UPDATE A NAMED USER CONSTANT "MOOKIE"
      CALL UPREG1(MOOKIE)
```

Subroutine Name: UPREGC

Description: This routine is identical to UPREG, except that updates are restricted to a selection set defined by the user. Any other parameter is held constant during the propagation of updates. This routine allows highly specific updates to be made to certain classes of data, much like a database query.

UPREGC can be called from any logic block where a change to the requested parameter(s) is appropriate.

Refer to the description of UPREG for more guidance and suggestions.

Calling Sequence:

```
CALL UPREGC (smn, type, param, id)
```

where:

`smn` . . . . . a submodel name to be included in the selection set, in single quotes.Only data owned by this submodel will be affected. Otherwise, use 'ALL' to refer to all submodels, 'ALLT' to refer to all thermal submodels, and 'ALLF' to refer to all fluid submodels.

`type` . . . . . a data type to be included in the selection set, in single quotes:

'ALL' . . . . . all data types
'ARRAY' . . . . array data only
'NUMUSER' . numbered user constants only
'NAMUSER' . named user constants only
'CONTROL'. . control data constants
'SOLVER' . . . solver and reliability engineering controls, constraints, etc.
'NODE' . . . . . node data (except sources) only
'SOURCE' . . . source data only
'COND' . . . . . conductor data only
'LUMP' . . . . . lump data only
'PATH' . . . . . path data only
'TIE'. . . . . tie data only
'FTIE'. . . . . ftie data only
'IFACE' . . . . . iface data only

`param`. . . . . a parameter name to be included in the selection set, in single quotes. Use 'ALL' to denote all relevant parameters. *Examples* include:

'T'. . . . . nodal temperatures only
'TLEN'. . . . . path lengths only
'SPD' . . . . . speed for a PUMP connector

Some parameters are accessible in this manner that cannot be addressed in logic blocks otherwise. Illustrative *examples* include:

'CSIV' . . . . . the F factor on a SIV node
'QTVS . . . . . the F factor on a TVS source
'GSIV' . . . . . the F factor on a SIV conductor
'GDIV'. . . . . the first F factor on a DIV conductor
'GDIV2'. . . . . the second F factor on a DIV conductor
'GDIVA'. . . . . the first array substitution factor on a DIV conductor
'GDIVA2'. . . . the second array substitution factor on a DIV conductor
'AFRAC' . . . . path area fraction for ties

To address upper or lower limits on design variables:

'DESLIMLO'. lower limit of the IDth design variable
'DESLIMHI' . upper limit of the IDth design variable
'RANHI'. . . . . upper limit of the IDth UNIFORM random variable
'RANLO' . . . . lower limit of the IDth UNIFORM random variable
'RANMN' . . . Mean of the IDth NORMAL random variable
'RANSD' . . . . SD of the IDth NORMAL random variable
'RANCV' . . . . CV of the IDth NORMAL random variable

`param` ..... (cont'd) To address upper or lower limits on constraint variables:
'csname' ..... where csname is the name of the optimization or reliability constraint variable, using ID=1 to denote the lower limit, and ID=2 to denote the upper limit.

`id` ..... an ID of an element to be included in the selection set, *not* in single quotes. Enter zero to mean all applicable parameters. For an ID to be nonzero, either a specific type (not 'ALL') *or* a specific parameter (again, not 'ALL') must be provided.

Guidance: The key to using these options is to understand that the more specifications are placed on a call, the fewer parameters will be affected. For example, to update all conductor data, specify the submodel as 'ALL' or 'ALLT' (since conductors only exist in thermal submodels), specify 'COND' as the type, and specify param as 'ALL' and ID as 0. See examples below. Also, note that multiple calls to this routine may be made sequentially to further customize which parameters are to be affected, perhaps using the DEREG and REREG families of routines (see Section 7.12.2 and Section 7.12.3) to temporarily or permanently declare what is *not* to be updated.

If no data matching the requested selection set is found to have been defined using a formula, then a caution will be issued. No caution will be issued if the parameters are there but have been disconnected using the DEREG family of routines (Section 7.12.2).

Examples:

```
C UPDATE ALL DATA (EQUIVALENT TO A CALL TO UPREG):
      CALL UPREGC('ALL','ALL','ALL',0)
C UPDATE ALL THERMAL DATA (EQUIVALENT TO A CALL TO UPREGT):
      CALL UPREGC('ALLT','ALL','ALL',0)
C UPDATE ALL FLUID DATA (EQUIVALENT TO A CALL TO UPREGF):
      CALL UPREGC('ALLF','ALL','ALL',0)
C UPDATE ALL CONDUCTOR DATA:
      CALL UPREGC('ALL','COND','ALL',0)
C UPDATE ALL CONDUCTOR DATA IN SUBMODEL FRED:
      CALL UPREGC('FRED','COND','ALL',0)
C UPDATE ALL DATA ASSOCIATED WITH CONDUCTOR 10 IN SUBMODEL FRED:
      CALL UPREGC('FRED','COND','ALL',10)
C UPDATE THE CONDUCTANCE OF CONDUCTOR 10 IN SUBMODEL FRED:
C EQUIVALENT TO "CALL UPREG1(FRED.G10)"
      CALL UPREGC('FRED','COND','G',10)
C NOTICE THE FOLLOWING IS EQUIVALENT TO THE ABOVE:
      CALL UPREGC('FRED','ALL','G',10)
C UPDATE ALL F FACTORS FOR SIV CONDUCTORS IN SUBMODEL FRED:
      CALL UPREGC('FRED','COND','GSIV',0)
```

```
C UPDATE ALL DATA IN THE ARRAY A23 IN SUBMODEL FRED:
      CALL UPREGC('FRED','ARRAY','A',23)
C NOTICE THE FOLLOWING ARE EQUIVALENT TO THE ABOVE:
      CALL UPREGC('FRED','ALL','A',23)
      CALL UPREGC('FRED','ARRAY','ALL',23)
C UPDATE ALL ARRAYS IN SUBMODEL FRED:
      CALL UPREGC('FRED','ARRAY','ALL',0)
C UPDATE ALL ARRAYS (THERMAL AND FLUID SUBMODELS)
      CALL UPREGC('ALL','ARRAY','ALL',0)
C UPDATE ONLY NAMED USER CONSTANTS
      CALL UPREGC(('ALL','NAMUSER','ALL',0)
```

Subroutine Name: FORCER

Description: This routine is similar to UPREG, except that all parameters are updates whether or not a change has been detected in their defining registers. In other words, FORCER is a more expensive and heavy-handed version of UPREG, but is sometimes needed to assure consistency in a model especially after calls have been made to RESPAR, RESTAR, or RESTDB.

Caution: FORCER not only updates *all* registers and parameters, *including those that have been disconnected* (Section 7.12.2), **it also reconnects any such disconnected parameters**.

Refer to the description of UPREG, DEREG (Section 7.12.2) and REREG (Section 7.12.3) for more guidance and suggestions.

Caution: If plenum states (PL, PL!, TL, or XL) have been defined by expressions, the order in which these expressions are updated is quasi-random, such that the intent of PL! (pressure priority) or TL (default temperature priority) may be lost if FORCER is called. If this is a concern, instead of FORCER use REREG followed by one or more DEREGC calls (to disconnect the relevant lump parameters) followed by UPREG.

FORCER has no arguments.

Calling Sequence:

```
CALL FORCER
```

7.12.2 Disconnecting Automatic Updates

In addition to being able to control what data is updated, the user is allowed to temporarily or permanently disconnect some or all variables from the automatic update sequence using the following routines:

DEREG . . . . . disconnects all automatic updates
DEREG1 . . . . . disconnects only a single specified parameter
DEREGC . . . . . disconnects a custom selected set of one or more parameters

Subroutine Name: DEREG

Description: This routine disconnects all parameters from the automatic update operations performed by the UPREG family of routines (see above), including those performed automatically be default. This action may be revoked completely or partially using the REREG family of routines described below.

DEREG may be called from any logic block. It has no arguments.

Calling Sequence:

```
CALL DEREG
```

Subroutine Name: DEREG1

Description: This routine is identical to DEREG, except that disconnected parameters are restricted to the single parameters provided as an argument.

DEREG1 can be called from any logic block.

Refer to the description of DEREG and the analogous routine UPREG1 for more guidance and suggestions.

Calling Sequence:

```
CALL DEREG1 (param)
```

where:

`param` . . . . . any legal reference to a SINDA/FLUINT parameter that was defined by a formula (expression containing registers or processor variables). This parameter may be subject to translation (e.g., “T400”), or perhaps has already been translated by the user using dynamic translation routines (e.g., NODTRN).

If the parameter passed into DEREG1 was not defined using a formula, a caution will be issued.

Subroutine Name: DEREGC

Description: This routine is identical to DEREG, except that disconnected parameters are restricted to a selection set defined by the user.

DEREGC can be called from any logic block.

Refer to the description of DEREG and the analogous routine UPREGC for more guidance and suggestions.

Calling Sequence:

```
CALL DEREGC (smn, type, param, id)
```

Refer to the UPREGC descriptions of parameters and calling examples, which operated analogously: instead of updating parameters, DEREGC disconnects them from being updated.

7.12.3 Reconnecting Automatic Updates

If some variables have been disconnected from the automatic update sequence, some or all of them can be reconnected using the following routines:

REREG . . . . . reconnects all automatic updates
REREG1 . . . . . reconnects only a single specified parameter
REREGC . . . . . reconnects a custom selected set of one or more parameters

Subroutine Name: REREG

Description: This routine reconnects any disconnected parameters from the automatic update operations performed by the UPREG family of routines (see above). This action may be revoked completely or partially using the DEREG family of routines described above.

To assure that the expression(s) defining the affected parameters have not changed while the variable was disconnected, this option also forces an update of all reconnected parameters.

REREG may be called from any logic block. It has no arguments.

Calling Sequence:

```
CALL REREG
```

Caution: Upon being reconnected, parameters are automatically flagged for update. This means the next time the appropriate update routine is called, the reconnected parameter will be updated even if there has been no change in the defining expression, which could conflict with independent user manipulations of those parameters.

Subroutine Name: REREG1

Description: This routine is identical to REREG, except that reconnected parameters are restricted to the single parameter provided as an argument.

REREG1 can be called from any logic block.

Refer to the description of REREG and the analogous routine UPREG1 for more guidance and suggestions.

Calling Sequence:

```
CALL REREG1 (param)
```

where:

`param.....` any legal reference to a SINDA/FLUINT parameter that was defined by a formula (expression containing registers or processor variables). This parameter may be subject to translation (e.g., “T400”), or perhaps has already been translated by the user using dynamic translation routines (e.g., NODTRN).

If the parameter passed into REREG1 was not defined using a formula, a caution will be issued.

Subroutine Name: REREGC

Description: This routine is identical to REREG, except that reconnected parameters are restricted to a selection set defined by the user.

REREGC can be called from any logic block.

Refer to the description of REREG and the analogous routine UPREGC for more guidance and suggestions.

Calling Sequence:

```
CALL REREGC(smn,type,param,id)
```

Refer to the UPREGC descriptions of parameters and calling examples, which operated analogously: instead of updating parameters, REREGC reconnects them to enable later updating.

7.12.4 Checking a Automatic Update Connection

Because of the detailed control the user is provided over the automatic update sequence, a separate utility is available to test any single parameter to see if it is currently or previously connected.

Subroutine Name: ISREG1

Description: This routine is allows the user to check whether any particular parameter was defined using a register or formula, and if so to see whether or not it is currently connected.

ISREG1 can be called from any logic block.

Calling Sequence:

```
CALL ISREG1 (param, nswer)
```

where:

`param` . . . . . any legal reference to a SINDA/FLUINT parameter that was defined by a formula (expression containing registers or processor variables). This parameter may be subject to translation (e.g., "T400"), or perhaps has already been translated by the user using dynamic translation routines (e.g., NODTRN)

`nswer` . . . . . the returned integer answer to the query:

0 . . . . . .a constant: not defined using a formula

1 . . . . . .a variable: *is* defined using a formula

2 . . . . . .a constant: defined using a formula, but disconnected

Example:

```
C CHECK TO SEE IF THE CAPACITANCE OF NODE 103 WILL CHANGE
C DURING THE NEXT UPDATE, AND IF SO CHANGE REGISTER "FRED" NOW
      CALL ISREG1(C103,NTEST)
      IF(NTEST .EQ. 1) FRED = 2.0
```

Caution: Note that a SOURCE DATA definition such as "100, imhot" defines a source of *imhot* to be applied to the Q of node 100, but d*oes not directly define the Q variable of node 100* since other sources may also be present on that node. Therefore, calling "ISREG1(Q100,NTEST)" will result in an answer of NTEST=0, since Q100 is not directly updated. Similarly, calling ISREG1(G300,NTEST) will result in NTEST=0 if conductor 300 is a SIV conductor or some other automatically updating conductor, since the G value of that conductor was not itself directly input as a register-containing expression.

7.12.5 Register Utilities

This section describes routines that can be used to update registers (without performing other changes), or to change the expression underlying the definition of a register.

CALREG .... updates all registers to reflect recent changes
CHGREG .... change the expression defining a register

Subroutine Name: CALREG

Description: This routine invokes the update of all calculator registers (and *only* those registers) if any have changed. If none have changed, no action is taken. Otherwise, all will be updated iteratively as needed to handle interdependencies. If an arithmetic fault has been created (e.g., IAN = 1.0/ALEX and ALEX = 0.0) or the system of registers does not converge, a processor abort will occur. No changes are propagated to any other SINDA/FLUINT parameter.

Registers are updated automatically called from any of the UPREG family of routines (Section 7.12.1), including those updates performed automatically by default, and so CALREG need not be called separately except to handle special modeling needs.

CALREG can be called from any logic block. It has no arguments.

Calling Sequence:

```
CALL CALREG
```

Subroutine Name: CHGREG

Description: Normally, all a user need do to change a value of a register is to change it like any other variable. For example, to change the value of a register named *regname* via an assignment statement or via a call to a subroutine:

```
regname = 10.0E-4                          $ assignment
call d1deg1(timen, a23, regname)           $ argument
```

Such changes, however, have the side effect of erasing any expression which might have been used to define that register. CHGREG may therefore be used in the more rare instance where the user needs to change the underlying expression defining a register, and not just the current value.

Neither resetting the value of a register nor calling CHGREG causes either registers nor the parameters defined on the basis of registers to be updated immediately. These updates will happen later automatically (by default), or can be forced using the CALREG routine (to update only the registers--see above), or by calling one or more of the UPREG family of routines (see Section 7.12.1).

CHGREG can be called from any logic block.

Restriction: Do not change the value of design variables while the Solver is active.

Calling Sequence:

```
CALL CHGREG (name, expr)
```

where:

`name`.....the name of the register, in single quotes
`expr`.....the new defining expression, also in single quotes

Restriction: **The new expression must be immediately valid or a processor abort will occur.** "Valid" refers both to syntax problems as well as arithmetic faults. For example, if the register named *wilma* is currently zero, the following call will fail:

```
CALL CHGREG('fred','1.0/wilma')
```

Instead, the value of *wilma* must be changed before the call to CHGREG:

```
wilma = 1.0
CALL CHGREG('fred','1.0/wilma')
```

Example:

```
C EQUIVALENT TO HAVING SET IN HEADER REGISTER DATA:
C     REGNAME = 2.0*pi*con*length
C
      CALL CHGREG('REGNAME','2.0*pi*con*length')
C
C NOTE "PI" IS A BUILT-IN AND ONLY HAS MEANING IN EXPRESSIONS,
C NOT IN USER LOGIC. CREATE A REGISTER CALLED "PIE = PI"
C TO USE THIS VALUE IN FORTRAN OR C EXPRESSIONS
```

7.12.6 Changing Nonregister Expressions Dynamically

Subroutine Name: CHGEXP

Description: This utility routine allows the user to assign a new expression to a single SINDA/FLUINT variable, or to change a previously defined expression.

Some parameters cannot be directly assigned an expression in an input block, or they inherit a default expression that may not be appropriate for certain modeling purposes. An example includes the coordinate location (CX, CY, CZ) of a secondary tank in a twinned pair, which defaults to the coordinate location of the primary tank. Also, generation and duct macrocommands (i.e, GEN, SIM, GENLU, HX, etc.) are useful, but they apply the same expression to all generated elements. This routine allows the user to customize the expressions for each such element, or to replace an earlier definition with a new expression, or to add an interrelationship that did not exist previously.

This routine represents advanced access into the spreadsheet database underlying SINDA/FLUINT. In order to enter a new expression into the database, the code requires the user to classify this parameter and usually to provide a current value and pointer to the current value*. The code attempts to check inputs for consistency, but not all possible errors can be detected and avoided. Therefore, *the user should exercise considerable caution to make sure that the inputs to this routine are valid, and that the underlying value is appropriate to be defined by an expression. Pay particular heed to the cautions and restrictions summarized below.*

Calling Sequence:

```
CALL CHGEXP (smn, type, param, id, id2, value, expr)
```

where:

`smn` ..... the thermal or fluid submodel name for the parameter to be changed, in single quotes. If a global control variable, named user variable, or solver variable is being affected (see “type” below), this argument is ignored.

`type`.....a data type to be affected, in single quotes:

'ARRAY'. . . . . array cell
'NUMUSER'. . numbered user variable
'NAMUSER'. . named user variable
'CONTROL' . . control variable
'SOLVER' . . . . solver and reliability engineering controls, constraints, etc.
'NODE' . . . . . node value (except sources)
'SOURCE'. . . . source value
'COND'. . . . . conductor value
'LUMP' . . . . . lump value
'PATH'. . . . . path value
'TIE' . . . . . tie value
'FTIE' . . . . . ftie data only
'IFACE'. . . . . iface value
'FTIE' . . . . . ftie value

`param` . . . . . the parameter name to be affected, in single quotes. *Examples* include:

'T' . . . . . nodal temperature
'TLEN' . . . . . path length
'SPD'. . . . . speed for a PUMP connector

Some parameters are accessible in this manner that cannot be addressed in logic blocks otherwise. Illustrative *examples* include:

'CSIV'. . . . . the F factor on a SIV node
'QTVS . . . . . the F factor on a TVS source
'GSIV'. . . . . the F factor on a SIV, SIVA, or SIVM conductor
'GDIV' . . . . . the first F factor on a DIV conductor
'GDIV2' . . . . . the second F factor on a DIV conductor
'GDIVA' . . . . . the first array substitution factor on a DIV conductor
'GDIVA2' . . . . the second array substitution factor on a DIV conductor
'AFRAC'. . . . . path area fraction for ties

To address upper or lower limits on design variables:

'DESLIMLO' . lower limit of the IDth design variable
'DESLIMHI' . . upper limit of the IDth design variable
'RANHI' . . . . . upper limit of the IDth UNIFORM random variable
'RANLO'. . . . . lower limit of the IDth UNIFORM random variable
'RANMN' . . . . Mean of the IDth NORMAL random variable
'RANSD'. . . . . SD of the IDth NORMAL random variable
'RANCV' . . . . CV of the IDth NORMAL random variable

To address upper or lower limits on constraint variables:

'csname' . . . . . where csname is the name of the optimization or reliability constraint variable, using ID=1 to denote the lower limit, and ID=2 to denote the upper limit.

`id` . . . . . the ID of the element to be affected, *not* in single quotes. Enter zero if no ID is applicable to the selected variable (e.g., a control variable, or named user constant)

`id2` ..... the secondary ID of the element to be affected, *not* in single quotes. Example: the cell number in an array, or the path ID for the AFRAC factor for a convection tie. Otherwise, enter zero if no secondary ID is applicable to the selected variable.

`value` ..... the translated variable to be affected, if applicable or available. For example, SUB1.T33 (*not in quotes*) if affecting the temperature of boundary node 33 in submodel SUB1. Otherwise, enter zero if not applicable (example: an untranslatable variable such as the upper limit on a constraint or the F factor on a SIV conductor). For most variables, it is important that the precise variable to be controlled by the expression is provided as an argument, and not a different variable with the same value. If other words, to change DRLXCA, provide "DRLXCA" or "smn.DRLXCA" (without quotes) as an argument. Setting "DTEST = DRLXCA" and then providing DTEST as the value argument is illegal and can cause errors. See examples below.

`expr` ..... the new expression to be applied to the selected variable, in single quotes (or a reference to a CARRAY, such as UCA1 -- not in quotes). This expression must be immediately valid.

Caution: If the parameter was previously disconnected via the DEREG family of routines, calling this routine will automatically reconnect it even if the expression has not changed (i.e., the call to CHGEXP was redundant).

Guidance: If a problem is encountered, then either no change will be made (and a warning will be issued), or the program will abort. Certain aborted runs can be triggered by internal routines called from CHGEXP. For example, if the user wishes to change the expression underlying the capacitance of node 303 in submodel "hard" and no such node exists, then the program will terminate in an internal call to NODTRN and the error message will not refer to "CHGEXP."

Guidance: No immediate changes to the specified parameter or to other parts of the spreadsheet utilizing that parameter will be made as a result of this call. The spreadsheet will resolve itself later automatically (unless NOUPDATE has been specified in OPTIONS DATA). Otherwise, the user can force an update by calling UPREG explicitly when desired.

Restriction: The parameter to be changed should not be the temperature of a diffusion nor arithmetic node, since these parameters are updated by SINDA. Analogously, the parameter to be changed should not be a thermodynamic state variable (PL, TL, XL, AL, HL, DL, PPGa-z, etc.) for a tank or junction since these parameters are updated by FLUINT.

Restriction: Certain source data options cannot be changed via this routine because of the cumulative nature of nodal sources. *This includes the Q term itself,* but also more esoteric options: the F factor on a SIV source and any factor in a DTV source. The program detects such problems automatically. Recall that multiple sources can be defined on any node; this is the underlying cause of this usage restriction, but is also the means by which it can be circumvented. If needed, use multiple expressions for the source on any node and turn them on or off using register multipliers.

Examples:

```
C CHANGE CONDUCTOR 2103 IN SUBMODEL FRED TO BE SAME AS 2003
      CALL CHGEXP('FRED','COND','G',2103,0,FRED.G2013,'FRED.G2003')
C CHANGE F FACTOR FOR SIVA CONDUCTOR 133 IN RADCAD to be "emis*area":
      CALL CHGEXP('RADCAD','COND','GSIV',133,0,0,'EMIS*AREA')
C CHANGE NAMED USER CONSTANT "SINDY" TO BE "NORTH*NORTHWES*TIMEN"
C WHICH IS STORED IN USER CARRAY 10, SUBMODEL ROD
      CALL CHGEXP(('ALL','NAMUSER','SINDY',0,0,SINDY,ROD.UCA10)
C MAKE 20TH CELL in ARRAY FRED.10 VARIABLE
      call chgexp('fred', 'array', 'a', 10, 20, fred.a(10+20),
      + 'myreg*wilma.C20')
C CHANGE AREA FRACTION FOR PATH 30 IN TIE SPURT.20
      call chgexp('spurt','tie','afrac',20,30, 0, 'square')
```

Guidance: Some of the arguments passed into CHGEXP will be ignored if irrelevant, and some will be required even if redundant. For example, there is no secondary identifier (ID2) for the F term on a SIV node. Also, specifying the value of "fred.a(10+20)" in the call below:

```
 call chgexp('fred', 'array', 'a', 10, 20, fred.a(10+20),
+ 'myreg*wilma.C20')
```

may seem redundant, but it provides the code with a means to not only fetch the current value of "fred.a(10+20)" but also to double-check its internal memory location. (Recall that in Fortran, arguments are passed by address and not by value.) In some instances, the internal memory location of the "value" argument provides the code with a means of deciding which parameter is being changed, and therefore it is critical that, if applicable, the real parameter be passed as "value" and not different but seemingly equivalent Fortran variable.

7.12.7 Parameter Sweep Utility

This section describes a utility for performing a parametric sweep of a single register. In many ways, the same task could be accomplished by a simple DO/ENDDO loop in OPERATIONS.

See also the related routines DVSWEEP (Section 5.15.1) and RVSWEEP (Section 5.26).

Calling Sequence:

```
CALL PSWEEP (regnam, rlo, rhi, nsteps, procnam)
```

where:

`regnam` . . . . . Real or integer register name, in single quotes
`rlo` . . . . . lower limit (starting point, real or integer, depending on register type)

`rhi` ..... upper limit (ending point, real or integer, depending on register type). rhi is normally greater than rlo, but if a sweep in the reverse direction is desired, then rhi may be less than rlo provided that nsteps is negative (as a signal of an intentional reversal rather than accidental).

`nsteps` .... number of points to evaluate, or for integer registers, the increment (integer).

If regnam is a (default) real register, then nsteps is interpreted as the total number of evaluation points, evenly spaced between (and inclusive of) rlo and rhi. In other words, the program will start with rlo and increment by (rhi-rlo)/(nsteps-1) until it reaches rlo. nsteps should be more than one in this case. If rlo>rho, the nsteps should be negative and the *decrement* will be applied as (rhi-rlo)/(|nsteps|-1)

If instead regnam is an integer register, then nsteps is interpreted as the increment: a value of 1 causes each point to be evaluated (rlo, rlo+1, rlo+2, ... rhi), and a value of 2 causes every other point to be evaluated (rlo, rlo+2, rlo+4, ... rhi).

nsteps should be positive if rhi>rho, and should be negative if a sweep from high to low (rlo>rhi) is desired instead.

`procnam` ... procedure or solution name, in single quotes

'steady' . . . run steady state,* calls the normal OUTPUT CALLS (this is the default if procnam is omitted as an argument)

'transient' . run transients,† including internal reset/restart (see below)

'proc' .... run PROCEDURE, calls SOLOUTPUT each time

'relproc' .. run RELPROCEDURE, calls RELOUTPUT each time

Examples:

```
CALL PSWEEP('length', 1./12., 2./12., 10, 'proc') $ forward sweep
CALL PSWEEP('length', 2./12., 1./12.,-10, 'proc') $ reverse sweep
CALL PSWEEP('npipes', 1, 6, 1, 'steady') $ integer example
```

Restarting for Transients: Unlike steady state analysis, transients require specific initial conditions, and hence repeated transient analyses require that the network be reset to the same initial condition. If the user is calling a transient routine within PROC or RELPROC, such resetting is the user's responsibility, as with any call to SOLVER or to statistical analysis routines (e.g., DSAMPLE, RVSWEEP). In PSWEEP, if the user selects 'transient' as the procedure name, then the code internally resets the network (and TIMEO/TIMEN values) to the conditions as they were at the start of the PSWEEP call. This is performed using SAVPAR and RESPAR (saves and restores all parameters, except the swept parameter and formulas based on that parameter). If this action is not appropriate, PSWEEP should not be used and the user should write their own parametric variation logic in OPERATIONS.

* Calls FASTIC internally. Use procnam = 'stdstl' to use STDSTL instead.

† Calls FWDBCK internally. Use procnam = 'forwrd' to use FORWRD instead.

Dynamic Mode in Thermal Desktop: If the parameter to be varied is also a Thermal Desktop symbol (and its update requires Thermal Desktop calculations: “the dynamic mode”), then either PROC or RELPROC must be used instead of STEADY and TRANSIENT because of the need to add extra routines to communicate with Thermal Desktop. For example, to execute a sweep of variable WIDTH from 0.01 to 0.10 in increments of 0.01 using a steady state solution, the following would be needed as a minimum:

```
HEADER OPERATIONS
            ...
      CALL PSWEEP('WIDTH',0.01,0.1,10,'PROC')
HEADER PROCEDURE
      CALL TDSETREG('WIDTH',WIDTH)  $ Send current value to TD
      CALL TDUPDATE                 $ rerun last case with changes
      CALL STEADY
```

7.13 Solver Output and Utility Routines

This section describes utility routines that are useful when working with the Solver (Section 5). The following list includes output routines, routines to help the user manipulate design variables, constraint variables, and their limits, and routines to assist in data-intensive task of correlating models using the Solver.

Recall that, despite the apparent similarities of the routines in the following two sections, design variables are normally changed by the Solver, whereas constraint variables are normally updated by the user in logic blocks.

7.13.1 Design Variable Output and Manipulations

This section describes the following routines:

DESTAB. . . . . Tabulate design variables
CHGDES . . . . Change the limits of a design variable
HLDDES . . . . Hold a design variable temporarily constant
RELDES. . . . . Release a hold on a design variable

Subroutine Name: DESTAB

Description: This routine tabulates (to the output file) design variables, their limits, and whether or not these limits are currently active or perhaps even violated. Otherwise, information is provided on whether the limits are fixed or are parameters (defined by register-containing formulas). Information is also provided on the status of the current Solver execution.

See also REGTAB (Section 7.4.15) for a complete list of registers; design variables are a subset of this list.

DESTAB has no arguments.

Calling Sequence:

```
CALL DESTAB
```

Subroutine Name: CHGDES

Description: This routine can be used to change the upper or lower limits on design variables.

Caution: Sudden or drastic changes to these limits will not be well tolerated by the Solver. Also, an abort will occur if the new limit renders the current value of the design variable or its acceptable range invalid.

Caution: If the limit to be changed was originally defined as a register-containing expression, the change caused by CHGDES will be overwritten later unless DEREGC is used to disconnect that update process.

Calling Sequence:

```
CALL CHGDES (dvname, flag, value)
```

where:

`dvname` . . . . . the name of the design variable, in single quotes
`flag` . . . . . ‘hi’ or ‘high’ to change the upper limit
‘lo’ or ‘low’ to change the lower limit
`value` . . . . . the new value to use for the upper or lower limit

Example:

```
C Change the lower limit on design variable “diameter” to be 0.01
      CALL CHGDES(’diameter’, ’low’, 0.01)
```

Subroutine Name: HLDDES

Description: This routine places a temporary hold on the value of a design variable. Once called, the design variable will not be allowed to change in the Solver from its *current* value (the value at the time HLDDES was called). This routine is useful for dealing with discrete values (Section 5.9.1), or for performing an optimization or model correlation in steps.

When HLDDES is called, upper and lower limits are *internally* set equal to the current value, and external (user) limits are ignored and left unchanged. There is little computational savings in holding a design variable fixed, so if a variable is never changed during an entire run, it should not be included as a design variable (i.e., it should commented out temporarily instead). If different sets of variables are to be permuted at different times within one run, thought should be given to separating the runs.

Calling Sequence:

```
CALL HLDDES (dvname)
```

where:

`dvname` . . . . . the name of the design variable to hold, in single quotes

Example:

```
C Freezes the current value of the design variable “rinner”
      CALL HLDDES(’rinner’)
```

Subroutine Name: RELDES

Description: Releases the hold on a design variable; undoes the action of HLDDES and restores control of the design variable by the Solver subject to the upper and lower user limits.

Calling Sequence:

```
CALL RELDES (dvname)
```

where:

`dvname` .... the name of the design variable to hold, in single quotes

Example:

```
C Release the hold on the design variable “rinner”
      CALL RELDES(’rinner’)
```

7.13.2 Constraint Variable Output and Manipulations

This section describes the following routines:

CSTNAMES . . Identify unnamed constraints by their internal designation
CSTTAB . . . . . Tabulate constraint variables
CHGCST . . . . . Change the limits of a constraint variable
HLDCST . . . . . Make a constraint variable a temporary equality constraint
RELCST . . . . . Release a hold on a constraint variable
OFFCST. . . . . Temporarily disable a constraint (as if commented out)

Subroutine Name: CSTNAMES

Description: Calling this routine will reset the program to refer to unnamed Solver constraints by their internal machine-generated names (YZX99999, YZX99998, etc.). Calls to CSTTAB will then reveal these names, which may then be used as arguments in advanced controls such as CHGCST, HLDCST, and OFFCST (below)

CSTNAMES has no arguments. Repeated calls do not disable this feature: it is not a mode toggle.

Calling Sequence:

```
CALL CSTNAMES
```

Subroutine Name: CSTTAB

Description: This routine tabulates (to the output file) constraint variables and expressions, their limits, and whether or not these limits are currently active or perhaps even violated. Otherwise, information is provided on whether the limits are fixed or are parameters (defined by register-containing formulas). Information is also provided on the status of the current Solver execution.

If no constraint variable was named (i.e., if the constraint were posed purely in terms of expressions containing registers and/or processor variables), then that constraint is listed as “unnamed” by CSTTAB and the first portion of the expression defining that unnamed constraint is listed. Note that if two expressions such as “sub.T1 <= sub.T2” had been used, then the current values of each expression would be listed, but only one expression (“sub.T1” in this case) would be listed as a defining expression. An equivalent constraint of “sub.T1 - sub.T2 <= 0” imposes no such ambiguity.

If CSTNAMES (above) has been called, then instead of the expression for unnamed constraints, the machine-generated name will be revealed instead. This name can then be used as an argument in subsequent calls to CHGCST, HLDCST, and OFFCST (below).

CSTTAB has no arguments.

Calling Sequence:

```
CALL CSTTAB
```

Subroutine Name: CHGCST

Description: This routine can be used to change the upper or lower limits on named constraint variables.

Caution: Sudden or drastic changes to these limits will not be well tolerated by the Solver, especially if the upper or lower limit on the constraint was active. Also, an abort will occur if the new limit renders the current range of the constraint variable invalid.

Caution: If the limit to be changed was originally defined as a register-containing expression, the change caused by CHGCST will be overwritten later unless DEREGC is used to disconnect that update process.

Calling Sequence:

```
CALL CHGCST (csname, flag, value)
```

where:

`csname` .... the name of the constraint variable, in single quotes
`flag` ..... 'hi' or 'high' to change the upper limit
'lo' or 'low' to change the lower limit
`value`..... the new value to use for the upper or lower limit

Example:

```
C Change the upper limit on constraint variable "depress" to be 10000
      CALL CHGDES('depress', 'high', 1.0E5)
```

Subroutine Name: HLDCST

Description: This routine places a temporary hold on the *desired* value of a named constraint variable; makes it an equality constraint instead of a range or upper/lower limit. Once called, the Solver will attempt to maintain the variable at its *current* value (the value at the time HLDCST was called).

External (user) upper and lower limits of the named constraint variable are set equal to the *current value*, as if CHGCST were called, and register-based formula for those limits are temporarily disconnected from the update process. This routine can be called multiple times if needed to reset the limits. Note: the value of the constraint variable should continue to be updated in logic: a departure from the limits is what is monitored by the Solver.

Calling Sequence:

```
CALL HLDCST (csname)
```

where:

`csname` . . . . . the name of the constraint variable to hold, in single quotes

Example:

```
C Sets the constraint variable “gap” to be an equality constraint
      CALL HLDCST(’gap’)
```

Subroutine Name: RELCST

Description: Releases the hold on a named constraint variable; undoes some *but not all* of the action of HLDCST. This routine reconnects the upper and lower limits to the update process if they were defined as formulas, *but RELCST does not restore previous limits* unless they were defined using registers. Otherwise, the CHGCST routine must be used to change the limits after they have been released.

RELCST may also be used to reactivate a disabled constraint: it reverses the actions of OFFCST (described below).

Calling Sequence:

```
CALL RELCST (csname)
```

where:

`csname` . . . . . the name of the constraint variable to release or activate, in single quotes

Example:

```
C Release the hold on the constraint variable “gap”
      CALL RELCST(’gap’)
```

Subroutine Name: OFFCST

Description: Calls to OFFCST will inactivate a Solver (optimization) constraint temporarily (until a subsequent call to RELCST is made), as if it had been "commented out." OFFCST therefore inactivates both upper and lower limits. This is useful as a preparatory step for equality constraints before calling DSCANLH.

OFFCST should only be called in OPERATIONS before or after calls to SOLVER, DSCANLH, DVSWEEP, etc., but it should never be called during such a call (e.g., never in VARIABLES, FLOGIC, PROCEDURES, etc.).

Calling Sequence:

```
CALL OFFCST(csname)
```

where:

`csname` .... the name of the constraint variable to disable, in single quotes

Example:

```
C Temporarily disable the named constraint TEMPMIN
      CALL OFFCST('TEMPMIN')
C Temporarily disable the unnamed constraint revealed using CSTNAMES
      CALL OFFCST('YZX99994')
```

7.13.3 Test Data Correlation Support Routines

This section describes routines intended to help deal with the large data requirements typical of test data comparisons. These routines help prepare and compare large sets of predictions and test data. See also Section 5.10 and Section 5.14.

This section describes the following support routines:

COMPARE . . . Central routine for making and reporting comparisons
PREPLIST . . . . Helps prepare predictions for comparisons in COMPARE
PREPDAT1 . . . Helps prepare test data for comparisons in COMPARE
PREPDAT2 . . . Helps prepare test data for comparisons in COMPARE

Subroutine Name: COMPARE

Description: This routine performs comparison operations on two SINDA/FLUINT arrays. Presumably, one array contains test data and the other contains predictions. (Auxiliary routines PREPLIST, PREDAT1, and PREPDAT2 are available if needed to help prepare the arrays to be used for comparison in COMPARE.)

Normally, this routine is used to calculate values useful for the Solver when used for correlating models against test data. In this case, the returned result is often the OBJECT to be minimized. It can also be used for Minimax curve fits, wherein the user passes the first named constraint of a list (in input order, perhaps using a GEN option) that will contain the calculated errors (deviations).

As an alternative, COMPARE can be called to prepare a table describing the comparison. *This usage is strongly recommended as part of debugging a model calibration task,* but is also very useful for reporting the results of a correlation. An optional integer label (perhaps thermocouple or thermister location number, for example) can be assigned to each comparison point for reporting purposes.

Calling Sequence:

```
 CALL COMPARE (Ap(IC), At(IC), Nams(IC), ndo,
+ func, result)
```

where:

`Ap` . . . . . SINDA singlet array of prediction points
`At` . . . . . SINDA singlet array of test data points
`Nams` . . . . . optional SINDA singlet array of *integer* identifiers for data points; enter zero if not used. This option is *only* used if func = 'REPORT'
`ndo` . . . . . maximum number of comparisons to be made if smaller than the size of Ap and At, otherwise enter zero to compare all points

`func` . . . . . the name of the operation to perform, in single quotes:

‘MAXERR’ . . return the absolute value of the maximum error*
‘AVGABS’ . . . return the average of the absolute values of the errors†
‘SUMABS’. . . return the sum of the absolute values of the errors
‘SUMSQR’. . . return the sum of the squared values of the errors
‘RMSERR’. . . return the root mean square error†
‘SUMCUBE’ . return the sum of the cubed absolute values of the errors
‘RMCERR’. . . return the “root mean cube” error†
‘AVGERR’ . . . return the average value of the raw errors*
‘STDDEV’ . . . return the standard deviation of errors about AVGERR
‘REPORT’ . . . print a table summarizing the comparison
‘MINIMAX’. . store the errors starting in the named constraint parameter provided as the last argument, and proceeding in input order

`result` . . . . the returned value (real or integer, depending on func)
or if func = ’MINIMAX’, the first named constraint to contain the errors, or if func = ’REPORT’, the file name (NOUT, NUSER1, etc.) to be used (NOUT is used by default or if result contains an illegal negative integer value).

Examples:

```
CALL COMPARE( A100, MOD.A200, 0, 0, ‘RMSERR’, OBJECT)
CALL COMPARE( A100, MOD.A200, MOD.NA201, 0, ‘REPORT’, -1)
CALL COMPARE( A100, MOD.A200, MOD.NA201, 10, ‘MINIMAX’, CST101)
```

Caution: If OBJECT is used as the last argument, then any prior value of this variable will be erased (unless func=’REPORT’). To use for more than one comparison point (cumulatively), employ a temporary value such as OTEST as the last argument, and sum into OBJECT instead:

```
CALL COMPARE( A100, MOD.A200, 0, 0, ‘SUMSQR’, OTEST)
OBJECT      = OBJECT + OTEST
```

Caution: When using the MINIMAX option, be sure there are enough variables (named constraints) available to hold all the points for the comparison. The code will not overwrite other data (outside of CONSTRAINT DATA), but may overwrite named constraints that were not intended to be reset if the user is not careful. Use CSTTAB to check the actions of COMPARE if MINIMAX is used.

Guidance: When using the MINIMAX option, different results can be achieved simply by reversing the first two arguments (the “test data” versus “prediction” arrays), which results in negative error. This can provide a simple check on the adequacy of the results.

* *Not* recommended to be used as an OBJECT value to minimized. In the case of MAXERR, use the minimax methods instead, but see also Section 5.10.3.3 and Section 5.14.2 for guidance.
† Recommended to be used as an OBJECT value to be minimized.

Subroutine Name: PREPLIST

Description: This routine may be of use in the preparation of an array of predictions for comparison with test data in the COMPARE routine. The user specifies the list of nodes (or paths or lumps) to be used in the comparison in the form of a singlet integer array, and provides the name of the parameter to be used.

The SPACE command may be used in the definition of the resulting array (in ARRAY DATA) to allocate space without having to specify values for that array.

Calling Sequence:

```
CALL PREPLIST(smn, param, IDs(IC), ndo, Ap(IC))
```

where:

`smn` . . . . . thermal or fluid submodel name, in single quotes
`param` . . . . . parameter name, in single quotes:
'T' or 'Q' for a thermal submodel
'TL', 'PL', or 'FR' for a fluid submodel
`IDs` . . . . . SINDA singlet array of *integer* ID numbers for the chosen nodes, lumps, or paths in the submodel 'smn'
`ndo` . . . . . maximum number of preparations to be made if smaller than the size of IDs or Ap, otherwise enter zero to prepare all points
`Ap` . . . . . SINDA singlet array to contain the results

Example:

```
CALL PREPLIST( 'mod', 'T', mod.na34, 0, mod.a15)
```

For example, to fill an array with current nodal temperatures, the IDs of the desired nodes may be named as integers in ARRAY DATA:

```
1 = 101, 130, 131, 140, 141, 150
```

with space allocated to store the temperatures of those nodes:

```
2 = SPACE, 6
```

Then, at any time within the processor execution the user may update the array #2 to contain the current temperatures of the

```
CALL PREPLIST('mymod','T',A1,0,A2)
```

Array #2 might then be used in a call to COMPARE. In fact, Array #1 might also be used in the call to COMPARE as the Nams comparison point identifier array if appropriate. For example, if the corresponding test data temperatures were stored in array #11:

```
CALL COMPARE(A2, A11, NA1, 6, 'RMCERR', OBJECT)
```

Caution: If any of the requested network elements in the IDs array are not found in the requested submodel, an abort will occur.

Subroutine Name: PREPDAT1

Description: This routine may be of use in the preparation of an array of test data for comparison with predictions in the COMPARE routine. Depending on the method of storage of test data, PREP-DAT2 might be more appropriate.

The user specifies an *integer* list of arrays containing the time-varying test data. Each array will normally contain data for a single data point, either as a doublet array (that can be interpolated using D1DEG1), or as a singlet array with time stored separately in another singlet array (the two being interpolated using D1D1DA).

For example, PREPDAT1 could be used if three data measurements were stored in three separate doublet arrays:

```
1 =     0.0,    6.6
        2.0,    27.
        42.,    29.
5 =     0.0,    6.6
        1.0,    6.6
        2.0,    9.0
        3.0,    10.0
        2.7e+01, 1.9e+01
        2.8e+01, 1.95e+01
        2.9e+01, 2.05e+01
        3.0e+01, 2.2e+01
        3.2e+01, 2.55e+01
        3.3e+01, 2.65e+01
        3.4e+01, 2.7e+01
        4.2e+01, 2.8e+01
 7 =    0.0,    6.6
        1.0,    6.6
        38.,    21.
        42.,    23.
```

or it could be used if these data were stored in three separate singlet arrays with a fourth array (#10) containing the time values:

```
1 =     6.6, 27.0, 28.8, 29.
5 =     6.6, 9.0, 26.7, 28.
 7 =    6.6, 6.6, 21., 23.
10 =    0.0, 2.0, 38., 42.
```

In the former case, each array must contain an even number of values. In the latter case, all arrays must be the same size. Otherwise, errors will be reported internally by the D1DEG1 or D1D1DA routines and the program will abort.

The SPACE command may be used in the definition of the resulting array (in ARRAY DATA) to allocate space without having to specify values for that array.

Calling Sequence:

```
 CALL PREPDAT1 (time, smn, AIDs(IC), Atim(IC), ndo,
+ At(IC))
```

where:

`time`.....current value of problem time (for interpolating test data)
`smn`.....thermal or fluid submodel name, in single quotes, containing test data arrays
`AIDs`.....SINDA singlet array of *integer* ID numbers for the arrays (normally one per data point) containing test data (arrays must be in the submodel 'smn')
`Atim`.....if test data is stored as singlet arrays with a separate singlet array containing the time values, this may by input here. Otherwise, if the test data is stored in doublet arrays, this argument should be input as zero (0.0)
`ndo`.....maximum number of preparations to be made if smaller than the size of AIDs or At, otherwise enter zero to prepare all points
`At` .....SINDA singlet array to contain the results, one cell for each array listed in AIDs that was interpolated

Example:

```
CALL PREPDAT1(TIMEN,'model', model.na34, 0.0, 0, a15)
CALL PREPDAT1(TIMEN,'mod', mod.na36, mod.a1, 0, a15)
```

Subroutine Name: PREPDAT2

Description: This routine may be of use in the preparation of an array of test data for comparison with predictions in the COMPARE routine. Depending on the method of storage of test data, PREPDAT1 might be more appropriate.

To use PREPDAT2, test data *must* be stored in the following arrangement:

```
N = M, time1,     dat11, dat12, dat13, ... dat1M,
       time2,     dat21, dat22, dat23, ... dat2M,
       ...
       timeL,     datL1, datL2, datL3, ... datLM
```

where N is the array ID, M is the number of data points (perhaps thermister or thermocouple locations), and L is the number of time points saved. Note that the above format is *not* the same as the "bivariate array" described elsewhere in the manual, since each data stream (i.e., dat1i through datLi) is independent of other data streams.

If detectable, PREPDAT2 will abort if the input array does not meet the required format specifications.

The SPACE command may be used in the definition of the resulting array (in ARRAY DATA) to allocate space without having to specify values for that array.

Calling Sequence:

```
CALL PREPDAT2 (time, Adat(IC), ndo, At(IC))
```

where:

`time` ..... current value of problem time (for interpolating test data)
`Adat` ..... SINDA test data array stored according to the aforementioned format
`ndo` ..... maximum number of preparations to be made if smaller than the size of At (or the first cell in Adat), otherwise enter zero to prepare all points
`At` ..... .SINDA singlet array to contain the results, one cell for each data stream defined in Adat.

Example:

```
CALL PREPDAT2(TIMEN, a100, 0, a400)
```

7.13.4 Solver Connectivity and Sensitivity Checker

Subroutine Name: SOLCHECK

Description: SOLCHECK is a Solver diagnostic utility intended to help check both connectivity and sensitivity for an initial problem set-up. *Connectivity* means answering the question: “Do changes in design variables cause a noticeable change in the objective and in the constraint functions?” *Sensitivity* means answering the question: “Are these changes significant, not too sensitive, and are they in the right direction?”

SOLCHECK (no arguments) should be called from OPERATIONS. Starting with the current design variable values, SOLCHECK runs the Solver internally* for N+1 procedure calls, where N is the number of design variables.

SOLCHECK then prints a table of sensitivity information: how OBJECT (the first row printed) and each constraint vary as functions of each design variable. Each value is the percent change in OBJECT or constraint function divided by the percent change in design variable value (which is usually equal to +2%, or RDERO, by default). Therefore, “1.0” means that the OBJECT or constraint grows by 1% for every 1% growth in the design variable, and “-1.0” means it shrinks by 1% for

* Error messages might be generated directly from the Solver module as a result of the SOLCHECK call.

every 1% increase in design variable. As a further example, a value of “0.25” means 25 times the sensitivity compared to a value of “-0.01,” but in the opposite direction. An example output table is shown below:

```
                X1           X2           *X3           X4           X5

*OBJECT*     4.7621E-02  0.1429       0.000        0.2857       0.2381
 UNNAMED     -1.5883E-03 -2.9736E-03   0.000         -1.3346E-03 -0.2435
```

Notice that changes to X5 are by far the most important, especially with respect to the unnamed (expression-based) constraint. If any design variables have been held (using HLDDES), all sensitivities to this design variable will be zero, and the variable name will be prefixed with an asterisk “*” to indicate this condition (see “X3” in the above example).

SOLCHECK matches the first N+1 steps of an actual Solver call, and provides a “Solver-eye view:” a snapshot of how the problem initially appears to the Solver. Zeroes and very small values are not by themselves a clear indication of a problem, but they should be scrutinized (as should any values substantially larger than other values: ideally, the values should all be about the same order of magnitude). Zero values indicate either an extremely weak or nonexistent connection between a design variable and the objective or constraints. If these values are unexpected, double check the problem set-up, and consider increased RDERO and/or tightened SINDA/FLUINT convergence and accuracy settings.

A final word is required regarding constraint sensitivities. The Solver is actually not sensitive to the value of the constraint function itself, but rather to how much it moves relative to its upper and lower limits (which themselves might be variables). SOLCHECK does not take into account those complexities, opting instead of a simple single-valued output of constraint function changes, ignoring the effects of limits.

Calling Sequence:

```
CALL SOLCHECK
```

7.14 Reliability Engineering Output and Utility Routines

This section documents routines useful in the application of the Reliability Engineering module, which is documented in Section 5.

7.14.1 Output Options

This section documents the text output routines for the Reliability Engineering module.

Subroutine Name: RANTAB

Description: This routine tabulates random variables to the output file, and is useful primarily for checking inputs. Thus, it is normally called from within OPERATIONS before a call to SAMPLE, DSAMPLE, or RELEST. Information is also provided on the status of any current or previous reliability estimation in case it is called from other places such as RELOUTPUT CALLS.

RANTAB provides mean and standard deviations for all variable types, as well as a few data points from each distribution. Note that the 50% (median) value is only the same as the mean value for symmetric distributions. Uniform and Gaussian distributions are intrinsically symmetric, but ARRAY distributions may not be.

See also REGTAB (Section 7.4.15) for a complete list of registers; random variables are a subset of this list.

RANTAB has no arguments.

Calling Sequence:

```
CALL RANTAB
```

Subroutine Name: RCSTTAB

Description: This routine tabulates reliability constraint variables and expressions, their limits, and the probability that these limits have not been exceeded: the reliability. RCSTTAB is the primary output of the Reliability Engineering module, and is normally called in OPERATIONS following a call to SAMPLE, DSAMPLE, or RELEST. Alternatively, it might be called from RELOUTPUT CALLS to monitor progress on SAMPLE or DSAMPLE runs. Information is also provided on the status of the current or previous reliability estimation.

There are two ways in which reliability is estimated: by tally and by calculation assuming a normal (Gaussian) response distribution. Both are shown, with negative numbers indicating that either data is not available or applicable. For example, tally methods are not available in RELEST.

Overall reliability (the variable OVEREL) is printed for SAMPLE and DSAMPLE runs. This is a simply tally of cases in which no reliability constraints were violated divided by the total number of samples. If all constraints are independent or in series, then this value is an estimate of the overall reliability of the design.

If no reliability constraint variable was named (i.e., if the reliability constraint were posed purely in terms of expressions containing registers and/or processor variables), then that constraint is listed as “unnamed” by RCSTTAB. Instead, its number (corresponding to the input order) is listed to help identify it. The routines RCGET can be used to dynamically fetch the output of RCSTTAB for a named reliability constraint, while RCGETNO can be used to fetch similar output for numbered (unnamed) constraints.

RCSTTAB has no arguments.

Calling Sequence:

```
CALL RCSTTAB
```

7.14.2 Database Options

This section describes the routines that read and write the database files for the Reliability Engineering module, as described in Section 5.23.

Note that these routines produce files or folders that are of the same format as the SAVE and RESAVE routines and read by the RESTAR routine. Thus, *these data sets may be created for one purpose and used for another.* Key differences in REDB files and folders are that they can be (1) read within the same run like a parametric file (SAVPAR, SVPART, RESPAR) and (2) created in one run and appended in a later run.

Subroutine Name: SAVEDB

Description: This subroutine writes all program variables (temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays, and flow data) to the binary reliability engineering database (REDB) file or folder. SAVEDB requires that a user's file name (or CSR folder name) be equated to the name REDB in the OPTIONS DATA. Each time SAVEDB is called, a sequential restart record number and current problem time will be printed for future reference, and will be also returned as the second argument, NREC. At the user's option, parts of the program variables (e.g., capacitances, Q-sources) may be left out of the output process.

Restrictions and Guidance: Should never be called from a VARIABLES or FLOGIC block. Since its output is not restricted to a single submodel, calls to SAVEDB should appear in only one OUTPUT CALLS block in a multi-submodel problem. Generally, *it is intended to be called from the RELOUTPUT CALLS block when used in conjunction with the Reliability Engineering module.* The "ALL" option should be used unless the REDB file threatens to grow to an unmanageable size. To simplify program control, *named (global) user constants are never saved.*

Calling Sequence:

```
CALL SAVEDB ('ARGS', NREC)
```

where `'ARGS'` is a combination of characters that define the types of data to be written to the restart file. It works as follows:

'ALL' . . . . . All parameters are written to REDB (default if argument is missing)
'T' . . . . . Write Temperatures to REDB
'C' . . . . . Write Capacitance values to REDB
'Q' . . . . . Write Q-source values to REDB

'G'. . . . . Write Conductor values to REDB
'N'. . . . . Write control constants to REDB
'U'. . . . . Write *numbered* user constants to REDB
'A'. . . . . Write user arrays and CARRAYs to REDB
'L' . . . . . Write lump PL, TL, XL, and QDOT to REDB (for plots)
'P' . . . . . Write path FR to REDB (for plots)
'M' . . . . . Save advanced mixture and iface data
'F' . . . . . Save remaining FLUINT data
'R'. . . . . Save register names and expressions
'xxxxxx' . . . . . xxxxxx is any combination of T,C,Q,G,U,N,A,L,P,R,M and/or F
'-xxxxx' . . . . . save all data *except* the combination of 'xxxxx'

Note: ARGS must always be enclosed in character delineators ('). In order to restart FLUINT models, ARGS must be 'ALL' or include 'LPFM'.

NREC is a returned record number, making SAVEDB a superset of SAVE (or RESAVE) and SAVPAR (or SVPART). This record number can be used to return to a previous state in the same run by passing it to RESTDB, much as record numbers from SAVPAR or SVPART can be passed to RESPAR. ***Data written to REDB can be accessed in a later run, or the same run, or both.***

Subroutine Name: RESTDB

Description: This subroutine reads in all program variables (temperatures, capacitances, conductances, Q-sources, control constants, user constants, user arrays and flow variables) from the binary REDB (reliability engineering database) file or folder. Each call to SAVEDB will print and return an associated record number to be used with subroutine RESTDB.

The exact composition of the data read in depends on argument 'ARGS' used with subroutine SAVEDB when the restart data was written to the REDB data set. If being used to access a crash file or folder, the data source contains the information equivalent to a call to SAVEDB with the argument 'ALL'.

Unlike RESTAR, RESTDB can be used in one of three modes, as determined by the sign of the single argument, NREC:

1. If NREC is positive, then that record (creating during the current or previous run) is restored, *and subsequent calls to SAVEDB begin overwriting after that record*.

2. If NREC is zero, then the last record (creating during the current or previous run) is restored, *and subsequent calls to SAVEDB begin appending to the file after that record (i.e, at the end of the file)*. This is the most common usage of RESTDB.

3. If NREC is negative, this signals the code that an existing REDB file or folder is present and should be appended instead of overwritten. However, the last state will not be restored.

Restrictions and Guidance: Calls to RESTDB require that REDB be identified with a user's file name (or CSR folder name) in the OPTIONS DATA block. If any network elements, constants, arrays or array elements have been added or deleted, a collision will be detected and a restart cannot be performed. In this case, an error message will be printed and the program will abort.

Caution: Calls to RESTDB do not affect the dormant/awake status of thermal submodels, and do not affect the location of ties. In other words, the effects of DRPMOD and PUTTIE calls in previous runs are irrelevant. Also, note that the collision checks cannot detect changes in input order, which should be avoided unless purposefully employed by an advanced user.

Caution: Values of parameters (variables defined by registers or register-containing expressions) may be overwritten when calling RESTDB. This action may overwrite previous updates caused by changes to registers. Furthermore, subsequent calls to the UPREG family of routines (Section 7.12.1) will have no effect unless the registers *themselves* have changed, in which case a call to FORCER should be considered. (See also Section 4.9.6.3 and Section 4.9.7.)

Caution: Values of registers may be overwritten when calling RESTDB. Care should be taken to avoid overwriting values of design variables if the Solver is being used, or random variables if the Reliability Engineering module is being used (Section 5). Otherwise, the user will need to make a subsequent calls to one of the UPREG family of routines (Section 7.12.1) to propagate the changes.

Calling Sequence:

```
CALL RESTDB (NREC)
```

where NREC is the restart file record number associated with the data to be read in.

Caution: If NREC is positive, ***subsequent calls to SAVEDB will overwrite any records following NREC.*** Otherwise, subsequent calls to SAVEDB will append to the REDB file. *RESTDB is not identical in usage to either RESPAR nor RESTAR in this respect.*

7.14.3 Utility Routines

The following are routines intended to customize execution of the Reliability Engineering module.

Subroutine Name: CURMEAN

Description: This routine takes the current value of a register and applies it as the mean value for one or more NORMAL random variables. *CURMEAN has no effect on UNIFORM and ARRAY random variables.* CURMEAN might be used, for example, to evaluate the reliability of a design which was just produced using the Solver, or in any case where the mean values are not known ahead of time.

Otherwise, the mean values or any other distribution parameter (e.g., UNIFORM range limits, even the array data underlying an ARRAY random variable) can always be set to registers or other expressions and changed at any time between (but not during) calls to SAMPLE, DSAMPLE, and RELEST.

Caution: If a NORMAL random variable was defined using CV (coefficient of variation) instead of SD (standard deviation), then changing the mean will change the value of the standard deviation as well.

Calling Sequence:

```
CALL CURMEAN ('name')
```

where:

`'name'` .... the name of a NORMAL random variable, in single quotes or 'ALL' to affect all NORMAL random variables

Subroutine Name: RESETM

Description: This routine sets the registers for one or more random variables to their mean values. RESETM is somewhat the opposite of CURMEAN, except that *it applies to any or all random variables, not just NORMAL ones.* It is useful to return to mean values after having called SAMPLE, DSAMPLE, or RELEST, which will otherwise leave the random variables at their last perturbed value upon return.

Calling Sequence:

```
CALL RESETM ('name')
```

where:

'name' . . . . . the name of a random variable, in single quotes, or 'ALL' to affect all random variables

Subroutine Name: RESAMP

Description: By default, SAMPLE and DSAMPLE are cumulative: they will append their results to the results of a previous call to the same routine (as long as no other type of Reliability Engineering module driver was called in between). To avoid this cumulative behavior, a call to RESAMP can be made before starting a second SAMPLE or DSAMPLE run. RESAMP does not affect whether or not SAVEDB overwrites or appends to the REDB file, since that is controlled independently. RESAMP is useful when SAMPLE or DSAMPLE are being called within PROCEDURE as part of using reliability as an objective or constraint for the Solver (see also Section 5.27).

RESAMP has no arguments.

Calling Sequence:

```
CALL RESAMP
```

Subroutine Name: RCGET

Description: RCGET is used to return the same results as does the routine RCSTTAB (Section 7.14.1), but in a Fortran callable fashion such that the resulting reliabilities can be used by the user in logic (perhaps as a constraint or objective for the Solver, see also Section 5.27). Also, given the mean and standard deviation returned by RCGET, NORMPROB and NORMVAL can be used to estimate the probability of other limits being exceeded.

Calling Sequence:

```
CALL RCGET ('name', RM, RSD, RTLO, RNLO, RTHI, RNHI)
```

where:

'name' . . . . . the name of a reliability constraint variable, in single quotes
RM . . . . . Mean of the response (returned)
RSD . . . . . Standard deviation of the response (returned)
RTLO. . . . . Tallied reliability of the lower limit (returned, may be -1.0 if not available or not applicable)
RNLO. . . . . Reliability of the lower limit assuming a Gaussian (normal) response (returned, may be -1.0 if not available)

RTHI . . . . . Tallied reliability of the upper limit (returned may be -1.0 if not available or not applicable)

RNHI . . . . . Reliability of the upper limit assuming a Gaussian (normal) response (returned, may be -1.0 if not available)

Example:

```
CALL RCGET('TMAX', Tmean, Tsd, Rello, Relnlo, Relhi, Relnhi)
```

Subroutine Name: RCGETNO

Description: RCGETNO represents the same functionality as RCGET, but is used for unnamed reliability constraints. To refer to them, the first argument is replaced by an integer identifier instead of an alphanumeric name in quotes such as RCGET uses. This identifier, NUMRC, is simply the number of the constraint in input order (including named constraints), as printed in the first column of RCSTTAB.

Calling Sequence:

```
CALL RCGETNO (numrc, RM, RSD, RTLO, RNLO, RTHI, RNHI)
```

where:

numrc . . . . . the number of an unnamed reliability constraint

RM. . . . . Mean of the response (returned)

RSD . . . . . Standard deviation of the response (returned)

RTLO . . . . . Tallied reliability of the lower limit (returned, may be -1.0 if not available or not applicable)

RNLO . . . . . Reliability of the lower limit assuming a Gaussian (normal) response (returned, may be -1.0 if not available)

RTHI . . . . . Tallied reliability of the upper limit (returned may be -1.0 if not available or not applicable)

RNHI . . . . . Reliability of the upper limit assuming a Gaussian (normal) response (returned, may be -1.0 if not available)

Subroutine Name: NORMVAL

Description: NORMVAL is a normal (Gaussian) distribution utility routine that returns the value X that corresponds to the input probability P, where P is the probability that a random value between negative infinity and X will occur. For example, if P=0.01 then a value less than X will occur 1% of the time, and a value greater than X will occur 99% of the time. If P=0.5, the mean (which is equal to the median for normal distributions) will be returned.

NORMVAL can be used with either the mean and standard deviations input with NORMAL random variables, or with the resulting means and standard deviations resulting from RCGET or RCGETNO (or RCSTTAB from a previous run) to calculate additional failure limits other than those posed in RELCONSTRAINT DATA, assuming that the response is Gaussian.

Calling Sequence:

```
CALL NORMVAL (P, A, SD, X)
```

where:

P. . . . . Probability (input, must be between 0.0 and 1.0)
A. . . . . Mean or average (input)
SD . . . . . Standard deviation (input, must be positive)
X. . . . . returned value corresponding to P, where P is the probability that a value less than X will occur

Example:

```
CALL NORMVAL( 0.999, Tmean, Tsd, Tlimit)
```

Subroutine Name: NORMPROB

Description: The reverse of NORMVAL: a normal distribution utility routine that returns the probability P that a random value less than the input value X will occur. For example, if P is returned as 0.8, then a value less than X will occur 80% of the time, and a value greater than X will occur 20% of the time.

NORMVAL can be used with either the mean and standard deviations input with NORMAL random variables, or with the resulting means and standard deviations resulting from RCGET or RCGETNO (or RCSTTAB from a previous run) to calculate the reliabilities for additional failure limits other than those posed in RELCONSTRAINT DATA, assuming that the response is Gaussian.

Calling Sequence:

```
CALL NORMPROB (X, A, SD, P)
```

where:

X. . . . . Value (input)
A. . . . . Mean or average (input)
SD . . . . . Standard deviation (input, must be positive)
P. . . . . returned probability (between 0.0 and 1.0) that a value less than X will occur

Example:

```
CALL NORMPROB( 30.0, Tmean, Tsd, Relnhi)
```

Appendix A Frictional Pressure Drop Correlations

This section describes the correlations used to calculate frictional pressure drops in tubes and STUBE connectors when IPDC is not 0. See also Section 3.16, which describes the flow regime determination logic that is applied when IPDC=6 or a negative value.

IPDC specifies the number of the two-phase pressure drop correlation to use. There is only one single-phase correlation, and this correlation is sometimes used in turn by the two-phase correlations. First, the single-phase correlation will be described, followed by a short description of each two-phase correlation.

A.1 Single-phase Correlation

Single-phase pressure drops are calculated using a Darcy friction factor. This factor (as represented on a Moody chart) is a function of Reynolds number (Re) for laminar flow, and a function of both Reynolds number and wall roughness ratio (roughness over diameter, e/D) for turbulent flow. A function from Churchill is used to analytically represent the Moody chart. This function overpredicts the friction factor somewhat in the transition region, but agrees very closely with experiments at both high and low Reynolds numbers. The formula is:

$$f = 8 \times \left(\left(\frac{8}{Re}\right)^{12} + \frac{1}{(A+B)^{3/2}} \right)^{1/12}$$

$$A = \left[-2.457 \times \ln \left(\left(\frac{7}{Re}\right)^{0.9} + \frac{0.27\,\varepsilon}{D} \right) \right]^{16}$$

$$B = \left(\frac{37530}{Re} \right)^{16}$$

This simple formula was selected in favor of more complicated representations of the Moody chart because of its speed and, perhaps more importantly, its numerical smoothness. The latter is important with regard to stability during steady-state iterations or slow transients. For low Reynolds numbers, roughness has no effect and the formula reduces to the familiar f = 64/Re.

A.2 Two-phase Correlations

This section is prefaced by a warning to the user that there is little agreement in the community as to two-phase correlation preferences. No single correlation can produce consistently acceptable predictions for all fluids under all flow regimes. In fact, disagreements between experiment and correlation (and even between two correlations) on the order of 50% are not uncommon. The situation is even more complicated by the addition of body force or gravity variations—most of the following correlations naturally focussed on air and water flow under 1g conditions. FLUINT therefore offers a choice of six correlations that have been developed for several fluids.

All of the two-phase correlations for IPDC=1, 2, 3, 4 and 5 were taken from Hetsroni.* This is strongly recommended reading for users who wish to explore these options in more depth.

McAdam's Homogeneous (IPDC=1)—The simplest (and smoothest and fastest) two-phase correlations are those based on the homogeneous model, which simply means that an effective density and viscosity are used and single-phase correlations are applied. Of these, McAdam's is the most widely used. Although homogeneous models often compare poorly with other correlations and underpredict pressure drops, they are the safest generalized correlation to use for uncommon fluids, or when there is no flow regime data available, or when the analysis is complicated by rapid boiling or condensation. Also, zero gravity flow regimes are expected to be "more homogeneous" than 1g flow regimes. It is for this reason that this is the default option.

Lockhart & Martinelli, Curve Fit by Chisolm (IPDC=2)—Like most of these correlations, the Lockhart/Martinelli correlation was first developed for water. It can only be used with caution for other fluids. Also, this correlation is not acceptable in the near-critical region. Therefore, it is not recommended, but is available for use at the user's discretion. On the other hand, this correlation is still one of the best for annular flow, which is dominant in most systems.

Baroczy, Curve Fit by Chisolm (IPDC=3)—This correlation is based on empirical fits to experimental data. It cannot be strongly recommended because it tends to oversimplify the correlating parameters and is not completely dimensionless, but is available for use at the user's discretion.

Friedel (IPDC=4)—This correlation contains an important parameter missing in previous correlations: the effects of surface tension. Also, it is based on a large experimental database covering many fluids. However, it overpredicts the pressure drop (sometimes by an order of magnitude) for the low Reynolds numbers typical of spacecraft thermal management systems.

Whalley Recommendations (IPDC=5)—This is not a correlation, but rather a quantified selection of one of the above three correlations based on (1) the relative viscosities of the phases, and (2) the order of magnitude of the mass flux. The correlation usually selected is Friedel's. Therefore, this "correlation" is recommended to users who dislike or distrust both McAdam's correlation and the following option, but don't have any alternatives.

Flow Regime Based (IPDC=6)—Refer to Section 3.16 for in-depth discussion, and Section 3.17 for descriptions of the associated slip flow modeling using twinned paths when IPDC=6 (or negative). Briefly, this option causes the program to estimate the current flow regime based on void fractions, velocities, body force magnitude and direction, fluid properties, etc. Four simplified regimes are recognized: bubbly, slug, annular, and stratified. Once the regime has been determined, the pressure drop is calculated on the basis of that regime. If the paths are also twinned, interface friction and wall friction apportionment are also based on the predicted regime.

If the user dislikes the homogeneous assumption but does not have another preference, this option should be used. The methods contained in this approach should extrapolate best to fluids other than water, and to microgravity. The only caution is that the methods used for the stratified regime do not apply well to single (homogeneous) paths. The pressure drop will be overpredicted in that case because of the use of a homogeneous void fraction, which is much higher than will occur

* G. Hetsroni, *Handbook of Multiphase Systems,* 1982

when slip flow is modeled. This option is strongly recommended for models involving two-phase mixtures (see also Appendix B).

Negative values of IPDC (-1, -2, -3, or -4) may also be used to force a regime selection, as described in Section 3.16.

Appendix B Forced Convective Heat Transfer Correlations

Whenever an HTN, HTNC, or HTNS tie (Section 3.6) is made between fluid and thermal submodels, FLUINT automatically invokes a set of convective heat transfer correlations for internal duct flow. These correlations all assume that the heat rate is constant over the solution interval. This heat rate is calculated based on the simple relation:

$$QTIE = AHX * H * (T - TEF)$$

where:

QTIE . . . . . Heat rate through tie positive *into* the lump and *out of* the node
AHX. . . . . Heat exchange area (returned as the AHT parameter)
H . . . . . Heat transfer coefficient (returned as the UB parameter)
T. . . . . Node temperature
TEF . . . . . Effective fluid temperature (usually the lump temperature)

The methods for calculating H are described below, according to the flow regime and the temperatures in the above equation. (With HTNS ties, the heat transfer rate is also a function of the state of the upstream lump.) Exact numeric representations of the two-phase correlations are not repeated here because of their complexity. However, the references from which they were extracted are noted. Note that all of these predictions may be high because of the absence of fouling factors.

This correlation set is called from the routine STDHTC, which is also user-callable (see Section 7.11.3). Extensive customization of this correlation set can be performed using the methods described in Section 3.6.6 and Section 3.6.7. Note also that these heat transfer predictions can be overridden by supplying subroutines with equivalent names and arguments in SUBROUTINES (see Section 7.11.3 for subroutine descriptions). User-defined (HTU and HTUS) ties may also be used for alternate convection correlations.

B.1 Single-phase Flow, Heating or Cooling

Heat transfer calculations for single-phase vapor or liquid are divided into three regimes: laminar, turbulent, and transition. The laminar flow (Re < 1960) heat transfer coefficients are given by the Nusselt number for circular isothermal ducts, namely XNUL, which defaults to 3.66. The Nusselt number is defined as:

$$NU = H * DH / TK$$

where:

NU . . . . . Nusselt number.
DH . . . . . Hydraulic diameter.
TK . . . . . Fluid thermal conductivity.

This Nusselt number (3.66) not only represents the most likely case, it is a good first guess for near-circular geometries or nonisothermal heating. However, it may underpredict heat transfer by up to a factor of two in thin rectangular cross-sections. If a better number is known, the user may always override these calculations either by using the XNUL tie parameter.

The turbulent (Re > 6420) single-phase Nusselt number is given by the Dittus-Boelter correlation:*

NU = CDB * (RE**UER) * (PR**PE)

where:

RE	Reynolds number.	
PR	Prandtl number.	
PE	Prandtl exponent:	UEC (default 0.3) for cooling,
		UEH (default 0.4) for heating.

CDB defaults to 0.023, and UER defaults to 0.8, but may be adjusted as needed per tie. See Section 3.6.6 for the extensive customizations available.

Transitional (1960 < Re < 6420) single-phase heat transfer is given by Hausen's correlation:†

NU = 0.116 * (RE**(2./3.) - 125.0) * (PR**PE)

Because the Prandtl number has no effect in laminar heat transfer, a smoothing function‡ is used between the above correlation and the laminar Nusselt number.

As discussed in Section 3.6.3, single-phase lumps employing HTN and HTNC ties use an assumption of downstream-weighted heat transfer: the heat transfer rate is calculated on the temperature at which fluid exits the perfectly mixed lump. This implicit formulation is stable, and is entirely consistent with the fundamental assumptions of a lumped-parameter network. However, when lumps are coarsely discretized (i.e., too few lumps are used to adequately represent the flowwise temperature gradient), the heat transfer rate will be underestimated, especially if the node is perceived as representing an average wall temperature for the segment. In accordance with a lumped-parameter analysis, the user should use an adequate number of elements to capture the desired number of states when using these ties.

As an alternative, the HTNS tie is specifically designed to overcome this limitation by interpreting heat transfer as occurring over a segment defined by upstream and downstream fluid states and an average wall temperature. For steady single-phase flows, HTNS (and HTUS) ties use a log mean temperature difference (LMTD) solution that tolerates very coarse spatial resolution.

---

* American Society of Heating, Refrigerating, and Air Conditioning Engineers, Inc. (ASHRAE) Handbook, *1981 Fundamentals,* 1981

† Karlekar and Desmond, *Engineering Heat Transfer*, 1977. This transition is only possible if parameters such as CDB, UER, etc. have been left at their default values. Otherwise, the transition is abrupt but continuous.

‡ Specifically, PE' = PE*F for Pr>1, where PE' is the corrected exponent, and F is the fraction along a cubic spline between Re_L=1960 and Re_T=6420: F = (3-2*R)*R^2 where R = (Re-Re_L)/(Re_T-Re_L). For Pr<1, PE' = PE*F^2.

Note that corrections for entrance lengths, rough walls, bends and coiled tubes, etc. are not performed by default. See Section 7.11.4 for relevant correlations.

B.2 Two-phase Flow, Boiling

Boiling heat transfer has enjoyed a great deal of attention for many years. This attention is largely due to the fact that boiling under "constant" heat flux conditions (due perhaps to electric dissipation or nuclear heating) can result in unstable or even dangerous conditions such as high wall superheating.

Extensive boiling calculations are performed by default that are valid for a wide variety of fluids and flow regimes, and should be adequate for most analyses, at least as a starting point. Users are always welcome to override these calculations if they have correlations that are more tailored for individual cases.

Two fundamental boiling heat transfer regimes are recognized: nucleate and film. Nucleate boiling is characterized by the formation of bubbles at the wall that drift into the bulk liquid stream. Film boiling is characterized by vapor-dominated heat transfer at high fluxes and/or high qualities (low liquid fractions). Nucleate boiling is assumed to exist for qualities from 0.0 to XNB (default: 0.7) only. For qualities between XNB and 1.0, the heat transfer coefficient is interpolated between the predictions of the nucleate correlation and the single-phase vapor correlation. This interpolation simulates the transition in a numerically smooth and stable manner that only roughly approximates the actual transition, which depends on upstream conditions and exhibits hysteresis.

It should also be noted that transitions between single- and two-phase coefficients are numerically smooth with the correlations used, although heat transfer coefficients rise steeply when the liquid begins to boil. Partial (subcooled) boiling begins as soon as the wall temperature exceeds saturation, with no delay (hysteresis) due to incipient superheat. See Section 3.6.7 for details.

Critical heat flux calculations are performed by default, and high-temperature film boiling regime is also recognized at even higher wall temperatures. See Section 3.6.7 for details.

In the limit of zero flow rate, single-phase vapor (laminar Nusselt number) heat transfer conductances are applied.

High Quality Boiling Interpolation—The high quality film boiling correlation is simply the single-phase Dittus-Boelter correlation for vapor using the current vapor mass fraction and void fraction. This correlation somewhat underpredicts the actual heat transfer because it neglects wall to liquid radiation and the unsteady impinging of liquid droplets on the wall. However, it provides a smooth and fast estimate and a transition between the detailed nucleate boiling predictions and the single-phase vapor heat transfer. This interpolation is to be distinguished from film boiling correlations, which apply at higher wall temperatures and lower qualities.

Nucleate Boiling Correlation—The basis of the nucleate boiling correlation is Chen's* (1963). This is a very popular correlation that is valid for most fluids although best for water. It provides a good transition from single phase liquid through the entire nucleate boiling range (which is usually further subdivided). The encodable version of this correlation was taken from Collier. It may be called independently via the CHENNB routine (Section 7.11.3).

B.3 Two-phase Flow, Condensation

Condensation heat transfer has received much less attention than has boiling, principally because of the high heat transfer coefficients usually encountered and the thermal stability. Consequently, there have been few correlations developed that attempt to include a wide variety of fluids and a wide range of flow regimes. One such correlation (Rohsenow†) is available in the default set of heat transfer calculations. This correlation was developed for condensation in annular flows using the Martinelli parameter. Smooth transitions into the single-phase regimes are also included to avoid numerical instabilities. The transition to single-phase liquid roughly accommodates the breakdown of annular flow into slug flow at low qualities (less than about 0.1), using scaling parameters similar to the Shah correlation.

In the limit of zero flow rate, single-phase liquid (laminar Nusselt number) heat transfer conductances are applied.

This correlation may be called independently via the ROHSEN routine (Section 7.11.3).

B.4 Two-phase Flow, Mixtures without Phase Change

Neglecting phase change within multiple constituent flows (mixtures) has a significant effect on the calculation of two-phase convective heat transfer. If a single-constituent two-phase mixture is heated, the temperature will remain relatively constant, and the quality will grow. If a multiple-constituent two-phase mixture is heated and no condensible/volatile substance is present, then both phases will simply get warmer and the quality will remain relatively constant.

If no flow regime information is available (i.e., IPDC is not 6 for the pertinent path), then the heat transfer conductance is calculated on the basis of an effective homogeneous fluid. Otherwise, the flow regime information is employed if available to improve the conductance estimate.

This same effective homogeneous fluid method is used if the flow is determined to be bubbly. For annular flow, the heat transfer is calculated on the basis of the liquid and vapor film conductances in series, with an estimation of the liquid film conductance provided by the same methods employed in the Rohsenow correlation (refer to Section B.3). For slug flow, an interpolation between the above two methods is made on the basis of void fraction.

---

* American Society of Heating, Refrigerating, and AIr Conditioning Engineers, Inc. (ASHRAE) Handbook, *1981 Fundamentals,* 1981

† Rohsenow and Harnett, *Handbook of Engineering Heat Transfer,* 1973.

For stratified flow, the heat transfer is calculated on the basis of the liquid and vapor film conductances in parallel, with an appropriate weighting applied that is based on wetted perimeter (assuming a circular cross section). Because of the parallel nature of this heat transfer mechanism, this regime tends to exhibit the higher heat transfer coefficients than other regimes at high qualities. On the other hand, at very low qualities in homogeneous flow (where the void fraction is overestimated), the conductance can actually be lower than other regimes because of blockage by the low-conductivity vapor.

The above treatment represents a departure from the previously described two-phase correlations, which neglect flow regime information. Furthermore, no distinction is made between heat addition and heat rejection, since no phase change takes place.

The annular method accounts for the proximity of the high-conductivity liquid layer to the wall. The stratified method predicts a degradation in heat transfer at low qualities over a homogeneous approach, and an increase at low qualities. Both effects are caused by the disproportionate coverage of the perimeter by the “minority” phase. Such nonmonotonic behavior is present in all regimes, including the homogeneous effective fluid method. This does not cause numerical instabilities since the flow quality is not a function of heat transfer, as it is with a pure substance.

This “correlation” may be called independently via the HTCMIX routine (Section 7.11.3).

Note that the flow regime prediction methods and the related slip flow options both require that the surface tension of all 9000 series fluids be supplied.

B.5 Two-phase Flow, Condensation of Mixtures

In the limit of no condensible/volatile species, the flow-regime based methods of Section B.4 are used. In the limit of a pure condensible/volatile species, the single-constituent condensation correlation is used (Appendix B.3). What happens in between is more than a matter of interpolations between these two extremes, although such interpolations do play a role.

When a pure substance condenses, the limiting factor is the conductance of the liquid film layer. This is the basis of the current default Rohsenow correlation (ROHSEN), which essentially includes an implicit assumption of annular flow due to its use of the Martinelli parameter. When a pure substance condenses, the liquid/vapor interface and the core vapor are essentially at the same temperature.

When a condensible species mixed with a noncondensible species condenses, the resistance in the vapor portion is no longer negligible. A gas layer builds up next to the condensate film, which both slows mass transport of the condensate due to a diffusion barrier, and allows the interface temperature to drop below the normal saturation or dew-point temperature of the core vapor. Normally, the thermodynamic effect (e.g., the shifting of the saturation condition) is dominant. However, the diffusive effect has been noted by some investigators to be just as significant. In either case, condensation heat transfer is very sensitive to small amounts of gas, and both effects are included by FLUINT when convection ties are used.

In FLUINT, the lump is assumed to be in thermodynamic equilibrium. In other words, any radial gradients within the fluid itself are modeled as a change in the heat transfer coefficient between the wall and the bulk fluid temperature. The following discussion describes how the heat transfer coefficient predicted from ROHSEN is modified to account for mass transfer through the gas diffusion barrier.

There are several means of calculating the mass transfer effects: the heat/mass transfer analogy (assuming a substitution of dimensionless parameters), the use of unity for a Lewis number, and the Chilton-Coulburn "j-factor" analogy (assuming that the dimensionless j-factors can be equated). All rely ultimately on heat transfer extrapolated to mass transfer, but to differing degrees. All assume low rates of mass transfer, which may or may not be the case but more encompassing methods would become specific to particular systems of fluids. The Chilton-Coulburn analogy, which requires estimation of diffusion coefficients (described below), is used because it was the most complete choice without assuming any system of fluids, and thereby losing generality.

The formulation of the Chilton-Coulburn analogy* used in the default FLUINT correlations is as follows:

$$\frac{h_v}{\dot{m}''/(\rho_{C\infty}-\rho_{Ci})} = \left[\frac{P_{tot}}{P_{NCG\infty}}\cdot\rho_{NCG\infty}\cdot\frac{\ln\left(\frac{\rho_{NCGi}}{\rho_{NCG\infty}}\right)}{\rho_{NCGi}-\rho_{NCG\infty}}\right]^{-1}\cdot\left[\rho\cdot C_p\cdot\left(\frac{k}{D_{im}}\right)^2\right]^{1/3}$$

which relates the vapor core heat transfer coefficient to the mass transfer flux as a function of the densities of the gases and condensible substances in the core and in the interface, and as a function of core mixture properties and the diffusion coefficient. The interface temperature must then be solved for iteratively using the following energy balance:

$$h_{film}\cdot(T_i - T_{wall}) = h_v\cdot(T_{bulk} - T_i) + \dot{m}''\cdot\Delta h_{fg} = h_{eff}\cdot(T_{bulk} - T_{wall})$$

In the above correlation, the condensate film resistance (h_{film}) is assumed to be identical to the current ROHSEN correlation for pure substances, corrected for liquid mixture properties (if any nonvolatile liquids are present). In other words, *the predictions of the ROHSEN routine (or any other pure substance correlation, for that matter), are degraded due to the presence of one or more noncondensible gases.*

The routine that calculates the degradation, HTCDIF, is available for direct calls, or for replacement in the event that the user desires to use an alternate method (see Section 7.11.3). See also HTUDIF, in which h_v is user-supplied, or HTFDIF, which may also be used for vaporization.

* ASHRAE Handbook, 1985 Fundamentals, Inch-Pound Edition, p4.11 and p5.10.

Estimating Diffusion Coefficients: FLUINT applies by default the method of Fuller, Schettler, and Giddings for estimating diffusion factors through binary mixtures. This relationship predicts the diffusion coefficient (D_{AB}) as a function of the mixture temperature (degrees K), pressure (atmospheres), molecular weights (M_A and M_B), and "diffusion volumes" (Σv_A and Σv_B) as follows:

$$D_{AB} = \frac{10^{-3} \cdot T^{1.75} \cdot [(M_A + M_B)/(M_A M_B)]^{1/2}}{P \cdot [(\Sigma v)_A^{1/3} + (\Sigma v)_B^{1/3}]^2}$$

Although this relationship is theoretically limited to low pressures, it provides reasonable estimates up to 10 atmospheres. The above method agreed with independent correlations developed specifically for air/water systems, with a maximum error of 10% at very high temperatures (>1000K).

To estimate the diffusion coefficients of a species i through a mixture of n gaseous substances, Blanc's law was chosen in modified form. Blanc's law describes how to combine individually calculated binary coefficients in the limiting case of trace amounts of species i:

$$D_{im} = \left(\sum_{j=1,\, j \neq i}^{n} \frac{x_j}{D_{ij}} \right)^{-1}$$

where x_j is the molal concentration of species j. This relationship was modified, exchanging the partial pressure ratio (PPG_i in FLUINT terms) for the molal concentration to better account for real gas effects. Also the above relationship is normalized such that the binary coefficient results in the limit of trace amounts of other species. This normalization is necessary in the limit of dominant amounts of a condensible species:

$$D_{im} = \frac{1 - P_i/P}{\left(\sum_{j=1,\, j \neq i}^{n} \frac{P_j/P}{D_{ij}} \right)}$$

B.6 Two-phase Flow, Boiling of Mixtures

In the limit of no condensible/volatile species, the flow-regime based methods of Section B.4 are used. In the limit of a pure condensible/volatile species, the single-constituent boiling correlation is used (Appendix B.2).

Currently, little correction is made to the boiling characteristics of pure substances in the presence of nonvolatile liquids. The Chen nucleate boiling correlation is employed with mixture film properties. At high concentrations of nonvolatiles, the results are interpolated with the non-phase change

flow-regime-based estimate, resulting in a degradation of heat transfer. However, no attempt is made to model the effects of an immiscible film, and because of the assumption of independent fluids, no change in the boiling point of the condensible substance is applied. (With a gas present, of course, the saturation condition is shifted due to the partial pressure of the gas.)

B.7 Dissolution and Evolution of Gases

This subsection describes some of the methods underlying the default gas dissolution modeling options described in Section 3.23.

The heterogeneous dissolution of gas into a solvent liquid is governed by a Fick's Law diffusion, and is proportional to a mass transfer rate constant for each species x (G_x), the interface surface area (A), and the difference between the saturation concentration and the current concentration of the solute in the liquid phase. In the evolution of gas out of a solvent liquid, heterogeneous (surface diffusion-limited) evolution also occurs, but two additional terms apply: a homogeneous (bulk fluid) nucleation rate (FRH_x), and the rate at which gas is advected ($FRNCV_x$). The total rate of dissolution/evolution (positive in the direction of dissolution) is therefore:

$$FRDx = FRHx + FRNCVx + (Xx - XFx) \cdot A \cdot Gx$$

where:

FRDx . . . . . mass flow rate of dissolution for species x
FRHx . . . . . mass flow rate of species x evolving homogeneously (negative if present)
FRNCVx. . . . . mass flow rate of species x evolving due to advection (negative if present)
Xx . . . . . the equilibrium mass concentration of species x
XFx. . . . . the current mass concentration of species x within the liquid
A . . . . . the interface surface area
Gx . . . . . the mass transfer rate constant for dissolution/evolution of species x

This section describes the methods used by the default options in FLUINT for calculating these terms.

B.7.1 Heterogeneous Dissolution and Evolution

The gas diffusion constant, G_x, is computed as $1/G_x=(1/k_l + P/(H_x*k_g))$, where:

k_l . . . . . liquid phase mass transfer coefficient
k_g . . . . . gas phase mass transfer coefficient
P . . . . . total system pressure
H_x . . . . . Henry's Law constant for species x

The gas phase mass transfer coefficients, k_l and k_g, are to be computed using the Chilton-Coulburn j-factor analogy:

$$\left(\frac{k_g}{V_g}\right)(Sc_g)^{2/3} = \left(\frac{h_g}{c_{p,g}V}\right)(Pr_g)^{2/3}$$

$$\left(\frac{k_l}{V_l}\right)(Sc_l)^{2/3} = \left(\frac{h_l}{c_{p,l}V}\right)(Pr_l)^{2/3}$$

where:

V_g . . . . . Gas phase mass velocity
V_l . . . . . Liquid phase mass velocity
V . . . . . total mass velocity
Sc_g . . . . . Schmidt number for gas
Sc_l . . . . . Schmidt number for liquid
h_g . . . . . gas to interface heat transfer coefficient
h_l . . . . . liquid to interface heat transfer coefficient
$c_{p,g}$ . . . . . specific heat for gas
$c_{p,g}$ . . . . . specific heat for liquid
Pr_g . . . . . Prandtl number for gas
Pr_l . . . . . Prandtl number for liquid

The diffusion coefficient is required for the calculation of the Schmidt number. The calculation of diffusion coefficients for gaseous species is described in Section B.5. For liquid species, the correlation proposed by Wilke and Chang, as described in Reid et al, is utilized:

$$D_{AB} = Y\frac{(\phi M_B)^{1/2}T}{\eta_B V_A^{0.6}}$$

where:

Y . . . . . 1.1728E-16 in SI units, 3.22593E-8 in ENG units
D_{AB} . . . . . liquid diffusion coefficient of solute A into solvent B
ϕ . . . . . association factor of solvent B (PHI in FPROP DATA blocks)
M_B . . . . . molecular weight of solvent B
T . . . . . temperature of liquid
η_B . . . . . viscosity of solvent B
V_A . . . . . liquid molar volume of solute A at its normal boiling temperature (VNB in 8000 series FPROP DATA blocks)

For the case where the solvent is a mixture, Perkins and Geankoplis have suggested that the viscosity of the mixture replace the viscosity of the solvent B, and the following relation for the association factor be used:

$$\phi M = \sum_{\substack{j=1 \\ j \neq A}}^{n} x_j \phi_j M_j$$

B.7.2 Homogeneous Evolution (Nucleation)

In addition to the diffusive evolution of gases, the homogeneous nucleation (bulk generation of bubbles within the fluid itself, versus heterogeneous evolution at nucleation sites in the walls and at liquid/vapor interface) is also modeled. Homogeneous evolution is a sudden and violent event, and the gas generation rates (FRH_x) are extremely high and not amenable to correlation. Once homogeneous evolution is initiated, the volumetric rate at which gas is evolved (ZR_x, such that $FRH_x = ZR_x$*VOL) is defaulted to a huge number,* but may be specified by the user as flagged using the IDGC constant.

The point at which homogeneous nucleation begins, however, can be estimated in terms of a critical pressure, PH, below which homogeneous nucleation can spontaneously occur.† PH can be negative, indicating infinite tolerance of supersaturated solutions.

The equations used to compute the homogeneous nucleation pressure are normally given as follows:

$$P_h = P_{sat} + \frac{CP_h}{C_s} - \left(\frac{16\pi\sigma^3}{3kT \ln\left(\frac{Z}{J}\right)} \right)^{\frac{1}{2}}$$

$$\ln\left(\frac{f^o(1 + C_s)}{C_s} \right) = \ln H + \frac{\nu P_h}{RT}$$

$$f^o = P_h \exp\left(\frac{P_h B_{nn}}{RT} \right)$$

* By default, the ZR is calculated such that equilibrium will be achieved in 10ms in the absence of other changes.

† Once it starts, homogeneous nucleation is assumed to continue until equilibrium is achieved since an interface now exists. In other words, the nucleation continues even if the pressure exceeds PH during gas evolution.

where:

T. . . . . temperature of the liquid
σ . . . . . surface tension
P_h . . . . . Homogeneous nucleation pressure
W . . . . . bubble mass
r_c . . . . . critical bubble diameter
B_{nn} . . . . . second virial coefficient, which can be estimated using the acentric factor and the critical temperature and a function of pressure.
C. . . . . moles dissolved gas/moles liquid
C_s . . . . . equilibrium concentration of dissolved gas at total pressure P_h
Z. . . . . rate constant
J . . . . . volumetric bubble generation rate
f^o . . . . . fugacity of pure gas
H . . . . . Henry's constant
k . . . . . Boltzmann constant

These equations would require an iterative solution to solve for P_h. They are simplified using the fact FLUINT currently handles only ideal gases as solutes. The equilibrium concentration and fugacity calculations are eliminated by substitution of P_h/C_s with Henry's constant, H:

$$P_h = P_{sat} + CH - \left(\frac{16\pi\sigma^3}{3kT \ln\left(\frac{Z}{J}\right)} \right)^{\frac{1}{2}}$$

When multiple solutes are present, a summation of the molar fractions term "CH" is used for all solutes, and the rate constant and the nucleation rate Z/J are weighted by molar fraction. Because these variables are in the logarithmic term, changes of several orders of magnitude result in only a minor change in the result.

Z/J is defaulted to a large number, $ZJ=10^{26}$. Using the IDGC control constant, the user can elect instead to provide this term.

Appendix C Available Fluids and Range Limits

Each fluid submodel may consist of one or more fluid species, either user-supplied or built-in. This section documents the built-in fluids, and also describes range limits applied to any fluid. Property limits are applied to any thermodynamic state within the fluid submodel, including lumps, path throat states, and local wall states for ties.

C.1 Standard Library Fluids

Table A-1 lists the standard library fluids that are immediately available in FLUINT. All fluids are referenced by their ASHRAE refrigerant number.

Users are cautioned that the accuracy of the standard library fluid properties is approximately 5% on average, and can be significantly worse at extreme temperatures for some fluids. More accurate descriptions of ammonia and water (as well as many cryogenic fluids) are available and should be used preferentially. ***See Section 3.21.7.9 and Section 3.21.7.10.***

The user may also describe the properties of alternate single-phase fluids such as a perfect gas (8000 series numbers), a simple incompressible liquid (9000 series), a simplified two-phase fluid (7000 series), or a complete two-phase fluid (6000 series). Alternate fluids are input using a HEADER FPROP DATA block, as described in Section 3.21. The user may also apply the PR8000 and PR9000 utilities (Section 7.11.2.5) to produce fast, simplified FPROP DATA descriptions from the above list.

Table A-1 Library Fluids

Number	Description
11	Refrigerant 11
12	Refrigerant 12
13	Refrigerant 13
14	Refrigerant 14
21	Refrigerant 21
22	Refrigerant 22
23	Refrigerant 23
113	Refrigerant 113
114	Refrigerant 114
216	Dichlorohexafluoropropane
290	Propane
318	Octafluorocyclobutane
500	73.8% R-12, 26.2% R-152a
502	75% R-22, 25% R-12
503	40.1% R-23, 59.9% R-13
505	78% R-12, 22% R-31
506	55.1% R-31, 44.9% R-114
717	Ammonia
718	Water*
1270	Propylene

* This fluid description (718) is no longer available. For more accurate alternatives, see for example f6070_water.inc (FID=6070).

Table A-2 lists the pressure and temperature values that delimit the valid range for the standard library fluids. (“Valid” does not imply accurate, merely “legal” or not nonsensical.) Excursions beyond these limits will produce and error message and FLUINT will reset the wandering lump to within limits. Note that such excursions can be disregarded within STEADY and STDSTL runs as long as they are temporary.

Table A-2 Range Limits for Standard Library Fluids

FLUID ASHRAE NO.	LOWEST TEMP. (R)	HIGHEST LIQUID TEMP. (R)	HIGHEST VAPOR TEMP. (R)	LOWEST PRESSURE (psia)	HIGHEST VAPOR PRESSURE (psia)
11	300.00	848.07	2544.2	1.61787E-03	639.50
12	300.00	693.30	2079.9	8.99569E-02	596.90
13	271.80	543.60	1630.8	0.83375	561.30
14	204.75	409.50	1228.5	0.77399	543.16
21	300.00	800.00	2438.7	3.20782E-03	670.49
22	300.00	664.50	1993.5	0.16170	721.84
23	269.17	538.33	1615.0	0.60119	701.42
113	428.70	871.00	2631.0	0.28871	443.78
114	322.50	753.95	2261.9	2.98006E-02	473.18
216	325.00	815.67	2447.0	1.85549E-03	399.46
290	300.00	665.95	1997.9	0.23209	617.40
318	417.10	699.27	2097.8	2.5725	403.60
500	300.00	681.59	2044.8	0.12214	641.90
502	300.00	639.56	1918.7	0.26453	591.00
503	263.39	518.00	1580.3	0.76153	563.66
505	300.00	703.66	2111.0	8.80965E-02	685.59
506	300.00	747.40	2242.2	2.46063E-02	749.37
717	351.80	730.08	2190.2	0.87683	1636.4
1270	300.00	657.09	1971.3	0.31651	667.01

FLUID ASHRAE NO.	LOWEST TEMP. (K)	HIGHEST LIQUID TEMP. (K)	HIGHEST VAPOR TEMP. (K)	LOWEST PRESSURE (Pa)	HIGHEST VAPOR PRESSURE (Pa)
11	166.67	471.15	1413.5	11.155	4.40921E+06
12	166.67	385.17	1155.5	620.23	4.11547E+06
13	151.00	302.00	906.00	5748.5	3.87001E+06
14	113.75	227.50	682.50	5336.5	3.74494E+06
21	166.67	444.44	1354.8	22.117	4.62287E+06
22	166.67	369.17	1107.5	1114.9	4.97692E+06
23	149.54	299.07	897.22	4145.0	4.83612E+06
113	238.17	483.89	1461.7	1990.6	3.05973E+06
114	179.17	418.86	1256.6	205.47	3.26249E+06
216	180.56	453.15	1359.5	12.793	2.75415E+06
290	166.67	369.97	1109.9	1600.2	4.25684E+06
318	231.72	388.48	1165.5	17737.	2.78272E+06
500	166.67	378.66	1136.0	842.14	4.42575E+06
502	166.67	355.31	1065.9	1823.9	4.07480E+06
503	146.33	287.78	877.95	5250.6	3.88631E+06
505	166.67	390.92	1172.8	607.40	4.72697E+06
506	166.67	415.22	1245.7	169.65	5.16675E+06
717	195.44	405.60	1216.8	6045.5	1.12824E+07
1270	166.67	365.05	1095.2	2182.2	4.59889E+06

The rules behind the property limits are as follows, with the appropriate property function given in parentheses:

1) The lowest temperature is given by the maximum of (the freezing point and the minimum of (300R and one half of the critical temperature)). One exception: 325R for fluid 216. (see VTMIN, Section 7.11.2)

2) The lowest pressure is the saturation pressure corresponding to the lowest temperature. (VPS(VTMIN))

3) The highest allowable temperature for liquids is the critical temperature, or the saturation temperature for the highest allowable pressure if less than critical. (VTLMAX)

4) The highest allowable temperature for vapor is three times the critical temperature. (VTGMAX)

5) The highest allowable pressure for vapor is the critical pressure (P_c) for most fluids, or $0.9*P_c$ for fluids 21, 1270, and 503. (VPS(VTLMAX))

Note that the temperature and pressure cannot be simultaneously above their critical values. The highest vapor temperature is arbitrary and is often unrealistically high, but allows a wide range for controlling excursions. The user should avoid analyses that rely on accurate properties at such high values of superheat.

For all fluids, the values of maximum and minimum conditions may be found with calls to VTMIN, VTLMAX, VTGMAX, VPS, and VPGMAX as shown above. Implicit limits (i.e., those implied by inputs but not specifically stated) of user-defined fluids are calculated by the program and echoed to the output file at the start of a run.

C.2 User Defined Fluids

Because of their simplicity, the valid ranges of user-defined single-phase fluids (8000 and 9000 series) are large. Pressures for liquid (9000 series) may assume any value ... including negative values. Pressures for gases (8000 series) must be positive, but can be arbitrarily high. The valid temperature range for user-defined fluids is determined by the FPROP DATA inputs (Section 3.21), whether explicitly or implicitly. While the temperatures must always be positive and can be arbitrarily high, the user can inadvertently limit the valid temperature range for extrapolated data. For example, viscosity must be positive. If the user input V = 1.0 and VTC = -0.01 at TREF = 300.0, then viscosity can become zero at T = 300.0+(1.0/-0.01) = 200.0 degrees. The *lower* temperature limit for that fluid would then be *at least* 200.0 degrees. While coefficient-style inputs can only cause one such limit, AT array data can be extrapolated in both directions and so cause both an upper and lower limit. FLUINT scans all data for user-defined fluids to determine the valid range at the start of the processor operation, and issues a message defining any limits and the reason for them. Afterwards, those limits are enforced and warning messages will be produced if the solution exceeds them. The user can explicitly influence temperature limits using the TMIN and TMAX input keywords.

When a two-phase working fluid is desired that is not contained in the standard library, the user may input a description of the fluid as either a 6000 series or 7000 series fluid. Section 3.21.6.3 and Section 3.21.7.6 describe implicit range limits for such fluids, and suggests explicit ones.

The range limits of a 6000 series fluid are as wide as the user sets them; this is the only fluid that can currently exceed both critical temperature and pressure. Enforcement of the limits is partially enforced by the program, but must also be enforced by the user within his or her functional subblocks. Range limits for the supercritical regime ($P>P_{crit}$) can even affect whether or not the fluid in that regime is considered liquid (XL=0.0) or vapor (XL=1.0). If PGMAX < PCRIT, then any fluid above PGMAX is considered to be a liquid. Any fluid about TCRIT is considered to be a vapor. The difference between being considered a “vapor” or a “liquid” includes both the selection of which properties to apply (densities, viscosities, etc.), but also whether or not the fluid may be considered compressible or not. Refer also to Figure 3-41.

Because of the simplified equation of state, the limits on a 7000 fluid are more restrictive. For such a fluid, the range limits may be set explicitly by the user, but the program performs consistency checks to further narrow the valid range. These checks are along the same lines as those limits noted for library fluids, except that the maximum liquid temperature and vapor pressure are usually sub-critical. Furthermore, the P_{sat}-T_{sat} relationship is scanned for implicit range limits. For example, the derivative $(dP/dT)_{sat}$ must always be positive. Furthermore, the calculated specific heat for saturated liquid must always be positive.

C.3 REFPROP 6000 Series fluids

Access to REFPROP fluids is described in Section 3.21.7.9 and Section 3.21.7.10. Table A-3 lists the available fluids in REFPROP,* along with their range limits.† (CRTech offers extensions to this list as well, such as He^3, for which only partial or preliminary data is available.)

For two-phase descriptions, the lowest allowed pressure is that of saturation for the lowest allowed temperature.

C.4 Range Limits of Mixtures

The minimum temperature in a lump cannot drop below the largest of all the lower temperature limits for all species currently found in the lump: the minimum common range applies. For example, if one constituent cannot drop below 150K, and another constituent in the same fluid submodel cannot drop below 175K, then the mixture is not allowed to drop below 175K in a lump containing both of these species.

* More complete information is in the “User Information” tab of REFPROP.XLS for users who have purchased and installed REFPROP separately.

† The values shown are actually for only the equations of state (thermodynamic properties). Transport properties and other limits (including maximum density) often mean that the valid ranges in F-files are smaller.

Table A-3 REFPROP: Available Fluids and Range Limits

Short Name	CAS number	Temp. Range, Max Pressure	Short Name	CAS number	Temp. Range, Max Pressure
acetone	67-64-1	178.5-550 K, 700 MPa	Novec-649	756-13-8	165-500 K, 50 MPa
ammonia	7664-41-7	195.495-700 K, 1000 MPa	octane	111-65-9	216.37-600 K, 100 MPa
argon	7440-37-1	83.8058-2000 K, 1000 MPa	orthohydrogen	1333-74-0o	14.008-1000 K, 2000 MPa
benzene	71-43-2	278.674-725 K, 500 MPa	oxygen	7782-44-7	54.361-2000 K, 82 MPa
butane	106-97-8	134.895-575 K, 200 MPa	o-xylene	95-47-6	247.985-700 K, 70 MPa
butene	106-98-9	87.8-525 K, 70 MPa	parahydrogen	1333-74-0p	13.8033-1000 K, 2000 MPa
carbon dioxide	124-38-9	216.592-2000 K, 800 MPa	pentane	109-66-0	143.47-600 K, 100 MPa
carbon monoxide	630-08-0	68.16-500 K, 100 MPa	perfluorobutane	355-25-9	189-500 K, 30 MPa
carbonyl sulfide	463-58-1	134.3-650 K, 50 MPa	perfluoropentane	678-26-2	148.363-500 K, 30 MPa
cis-butene	590-18-1	134.3-525 K, 50 MPa	propane	74-98-6	85.525-650 K, 1000 MPa
cyclohexane	110-82-7	279.47-700 K, 250 MPa	propylcyclohexane	1678-92-8	178.2-650 K, 50 MPa
cyclopentane	287-92-3	179.7-550 K, 250 MPa	propylene	115-07-1	87.953-575 K, 1000 MPa
cyclopropane	75-19-4	273-473 K, 28 MPa	propyne	74-99-7	273-474 K, 32 MPa
D4	556-67-2	290.25-673 K, 30 MPa	p-xylene	106-42-3	286.4-700 K, 200 MPa
D5	541-02-6	300-673 K, 30 MPa	R11	75-69-4	162.68-625 K, 30 MPa
D6	540-97-6	270.2-673 K, 30 MPa	R113	76-13-1	236.93-525 K, 200 MPa
decane	124-18-5	243.5-675 K, 800 MPa	R114	76-14-2	273.15-507 K, 21 MPa
diethyl ether	60-29-7	270-500 K, 40 MPa	R115	76-15-3	173.75-550 K, 60 MPa
deuterium	7782-39-0	18.724-600 K, 2000 MPa	R116	76-16-4	173.1-425 K, 50 MPa
dimethyl carbonate	616-38-6	277.06-600 K, 60 MPa	R12	75-71-8	116.099-525 K, 200 MPa
dimethylether	115-10-6	131.66-525 K, 40 MPa	R1216	116-15-4	117.654-400 K, 12 MPa
dodecane	112-40-3	263.6-700 K, 700 MPa	R123	306-83-2	166-600 K, 40 MPa
ethylbenzene	100-41-4	178.2-700 K, 60 MPa	R1233zd(E)	102687-65-0	195.15-550 K,100 MPa
ethane	74-84-0	90.368-675 K, 900 MPa	R1234yf	754-12-1	220-410 K, 30 MPa
ethanol	64-17-5	159-650 K, 280 MPa	R1234ze(E)	29118-24-9	168.62-420 K, 20 MPa
ethylene	74-85-1	103.986-450 K, 300 MPa	R124	2837-89-0	120-470 K, 40 MPa
fluorine	7782-41-4	53.4811-300 K, 20 MPa	R125	354-33-6	172.52-500 K, 60 MPa
hydrogen chloride	7647-01-0	155-330 K, 20 MPa	R13	75-72-9	92-403 K, 35 MPa
heavy water	7789-20-0	276.97-800 K, 100 MPa	R134a	811-97-2	169.85-455 K, 70 MPa
helium	7440-59-7	2.1768-2000 K, 1000 MPa	R14	75-73-0	120-623 K, 51 MPa
heptane	142-82-5	182.55-600 K, 100 MPa	R141b	1717-00-6	169.68-500 K, 400 MPa
hexane	110-54-3	177.83-600 K, 100 MPa	R142b	75-68-3	142.72-470 K, 60 MPa
hydrogen (normal)	1333-74-0	13.957-1000 K, 2000 MPa	R143a	420-46-2	161.34-650 K, 100 MPa
hydrogen sulfide	7783-06-4	187.7-760 K, 170 MPa	R152a	75-37-6	154.56-500 K, 60 MPa
isoctane	540-84-1	165.77-600 K, 1000 MPa	R161	353-36-6	130-450 K, 5 MPa
isobutane	75-28-5	113.73-575 K, 35 MPa	R21	75-43-4	200-473 K, 138 MPa
isobutene	115-11-7	132.4-550 K, 50 MPa	R218	76-19-7	125.45-440 K, 20 MPa
isohexane	107-83-5	119.6-550 K, 1000 MPa	R22	75-45-6	115.73-550 K, 60 MPa
isopentane	78-78-4	112.65-500 K, 1000 MPa	R227ea	431-89-0	146.35-475 K, 60 MPa
krypton	7439-90-9	115.775-750 K, 200 MPa	R23	75-46-7	118.02-475 K, 120 MPa
md2m	141-62-8	205.2-673 K, 30 MPa	R236ea	431-63-0	240-412 K, 6 MPa
md3m	141-63-9	192-673 K, 30 MPa	R236fa	690-39-1	179.6-400 K, 70 MPa
md4m	107-52-8	300-673 K, 30 MPa	R245ca	679-86-7	191.5-450 K, 10 MPa
mdm	107-51-7	187.2-673 K, 30 MPa	R245fa	460-73-1	171.05-440 K, 200 MPa
methane	74-82-8	90.6941-625 K, 1000 MPa	R32	75-10-5	136.34-435 K, 70 MPa
methanol	67-56-1	175.61-620 K, 800 MPa	R365mfc	406-58-6	239-500 K, 35 MPa
methyl linoleate	112-63-0	238.1-1000 K, 50 MPa	R40	74-87-3	230-630 K, 100 MPa
methyl linolenate	301-00-8	218.65-1000 K, 50 MPa	R41	593-53-3	129.82-425 K, 70 MPa
methyl oleate	112-62-9	253.47-1000 K, 50 MPa	RC318	115-25-3	233.35-623 K, 60 MPa
methyl palmitate	112-39-0	302.71-1000 K, 50 MPa	RE143a	421-14-7	240-420 K, 7.2 MPa
methyl stearate	112-61-8	311.84-1000 K, 50 MPa	RE245cb2	22410-44-2	250-500 K, 20 MPa
methylcyclohexane	108-87-2	146.7-600 K, 500 MPa	RE245fa2	1885-48-9	250-500 K, 400 MPa
MM	107-46-0	273-673 K, 30 MPa	RE347mcc	375-03-1	250-500 K, 20 MPa
m-xylene	108-38-3	225.3-700 K, 200 MPa	sulfur dioxide	7446-09-5	197.7-525 K, 35 MPa
neon	7440-01-9	24.556-700 K, 700 MPa	sulfur hexafluoride	2551-62-4	223.555-625 K, 150 MPa
neopentane	463-82-1	256.6-550 K, 200 MPa	toluene	108-88-3	178-700 K, 500 MPa
nitrogen	7727-37-9	63.151-2000 K, 2200 MPa	trans-butene	624-64-6	167.6-525 K, 50 MPa
nitrogen triflouride	7783-54-2	85-500 K, 50 MPa	trifluoroiodomethane	2314-97-8	120-420 K, 20 MPa
nitrous oxide	10024-97-2	182.33-525 K, 50 MPa	undecane	1120-21-4	247.541-700 K, 500 MPa
nonane	111-84-2	219.7-600 K, 800 MPa	water	7732-18-5	273.16-2000 K, 1000 MPa
			xenon	7440-63-3	161.405-750 K, 700 MPa

For gas mixtures (and states with a quality of unity within gas and nonvolatile liquid mixtures), the temperature cannot exceed the smallest of all the limits of each species in the mixture: the minimum common range again applies. Typically, limits for constituent gases can easily be raised* to assure the mixture has sufficient range, providing the user assures that those constituents never see such elevated temperatures to avoid inaccuracies in results.

The maximum pressure in a gas mixture is the smallest maximum pressure of all species present, following the same rule as is followed by temperatures. However, the minimum pressure in a gas/vapor mixture is the *sum* of the minimum pressures of each species, so that each species can occupy as much or as little of the lump as is necessary: it has the full range of its potential partial pressure.

No mixture is allowed if there is not *some* state (some minimum range of temperatures and pressures) for which all species in that submodel can coexist. This limit is applied at the submodel level. Otherwise, the valid temperature and pressure range at any local lump state might be larger than this minimum range if one or more species is missing.

For example, consider a high pressure helium bottle pressurizing a vessel that contains two-phase water. The bottle can start at a higher pressure than the maximum allowed by the water description, provided that no water is present in the bottle, and provided that the pressure of helium is decreased enough before it mixes with the downstream water.

The user is cautioned that exploiting the more liberal range limits that are applied to lumps (based on their local concentrations) can result in modeling problems.

For example, sudden jumps in temperature or pressure will occur as a new species is introduced to a lump for the first time, assuming that the initial temperature and pressure of that lump was not within the range of the newly introduced species. Consider a region of a model that contains only 9000 series fluids (whose pressure can drop below absolute zero pressure). If a species with a higher minimum pressure (such as a gas or two-phase fluid) is introduced into that region, sudden jumps in thermodynamic state, drops in time step, etc. can result.

Other issues (such as error messages and aborts) can occur while the program is exploring hypothetical states (as is necessary to calculate thermodynamic derivatives, or to check dew points of cold walls, etc.).

Therefore, *the user should always provide the widest range possible for all constituents.*

* Hypothetical values at extended ranges may be specified as long as they are consistent (e.g., extrapolations). Such “artificial” extensions are in fact encouraged providing that the user assures that the final lump and throat states do not traverse these fake data points.

Appendix D Unit Systems

When a fluid submodel is used in SINDA/FLUINT, the user is required to name the set of units desired using the global constant UID. There are two choices: metric (SI) and English (or U.S. Customary Engineering) units. Metric units use the MKS (meter, kilograms, seconds, Kelvin) basis. English units use feet, pounds-mass, hours, Rankine, BTUs, and psi. Table A-4 lists the assumed units for each option or parameter. Note that once a unit system is selected, all calculations including heat transfer with thermal submodels will be performed in that system of units. The user should therefore use the same unit system throughout the entire model (FPROP and MIXTURE blocks are exceptions).

ABSZRO and PATMOS can be used to change the units for temperature and pressure from relative to absolute (R to F, K to C, psia to psig, etc.), and that temperature, pressure, and flow rate conversion factors (PLFACT, PLADD, TLFACT, TLADD, and FRFACT) are available to convert inputs.

Table A-4 Unit Systems, Assumed Units

Parameter	Metric Units	English Units
Length	m	ft
Mass	kg	lb_m
Time	s	hr
Temperature	C, K	F, R
Pressure	Pa	psia
Quality, Mass Fraction	-	-
Void Fraction	-	-
Sp. Enthalpy	J/kg	BTU/lb_m
Energy	J/kg	BTU/lbm
Sp. Heat	J/kg-K	BTU/lb_m-R
Area	m^2	ft^2
Volume	m^3	ft^3
Density	kg/m^3	lb_m/ft^3
Flow Rate	kg/s	lb_m/hr
Viscosity (Dyn.)	kg/m-s	lb_m/ft-hr
Conductivity	W/m-K	BTU/hr-ft-R
Surface Tension	N/m	lb_f /ft
Gas Constant	J/kg-K	ft-lb_f /lb_m-R
VDOT	m^3/s	ft^3/hr
QL, QDOT, QF, QTIE	W	BTU/hr
COMP	1/Pa	1/psi
HC, DPAB	Pa	psi
AC, FC*	1/m-kg	1/lb_m -ft
WRF, UPF, FPOW, FK	-	-
CFC	m^3	ft^3
GK	kg/s-Pa	lb_m/hr-psi
HK	kg/s	lb_m/hr
EI, EJ	1/J-s	1/BTU-hr
DK, AM	-	-
FD, FG	1/s	1/hr
XGx, XFx, PPGx, MFx	-	-
UA, GF	W/K	BTU/hr-R
HEN	Pa/mass ratio	psia/mass ratio
GL, GD, GU, GT	$kg/s-m^2$	$lb_m/hr-ft^2$
UVI, ULI, UVPU etc.	W/m^2-K	BTU/hr-ft^2-R

* When FPOW, which itself is unitless, equals 1.0. Otherwise, FC units are dependent on value of FPOW

Appendix E Summary of FLUINT Numerical Methods

This appendix contains a summary of the mathematics behind the FLUINT solution algorithms. The derivation is too lengthy and the actual equations are too cumbersome to present here. However, the interested user will find sufficient information in this section for a general understanding of the approach and the approximations. With such an understanding, the user will be better able to apply FLUINT to unusual situations.

The basic methods described herein are used for both STDSTL and for transient analyses. For STDSTL analyses, an artificial time step is used to relax the network. These methods are incorporated into an algorithm documented in Table A-5. STEADY (aka, “STEADY”) uses a simple iterative relaxation method similar to solving junction equations for both tanks and junctions. See Section 4.2.4 for a discussion of STEADY.

Nomenclature

AC . . . . . Tube recoverable loss (area change and acceleration) coefficient
AF . . . . . Tube flow area
AM. . . . . Tube added mass coefficient
COMP . . . Tank wall compliance factor
DK . . . . . Path flow rate/twin's flow rate coefficient
EI . . . . . Path flow rate/enthalpy coefficient (defined upstream end)
EJ . . . . . Path flow rate/enthalpy coefficient (defined downstream end)
e . . . . . Flow Rate rectifier (+1 if defined upstream, -1 if downstream)
FC . . . . . Tube irrecoverable loss (friction) coefficient
FD . . . . . Tube interface drag coefficient
FK . . . . . Tube head loss coefficient (K factor)
FG . . . . . Tube momentum transfer coefficient (phase generation)
FPOW . . . Flow Rate exponent if FC (irrecoverable loss) term
FR . . . . . Mass flow rate
GK . . . . . Connector flow rate/pressure coefficient
h . . . . . Donor enthalpy (includes phase suction action, kinetic energy, etc.)
HC . . . . . Head coefficient (pressure, body force)
HK . . . . . Connector offset factor
M . . . . . Tank mass
PL. . . . . Lump pressure
Q . . . . . Nodal energy source or sink term
QDOT . . . Lump energy source or sink term
t . . . . . Time
TLEN. . . . Tube length
U . . . . . Tank internal energy
VDOT . . . Rate of change (growth) of tank volume
X . . . . . Solution vector
α. . . . . Area fraction (void fraction or liquid fraction)
ε . . . . . Phase donor rectifier (1 if recipient phase, 0 if donor phase)

Superscripts

n . . . . . Current time value
n+1. . . . . next time value

Subscripts

k . . . . . path
l . . . . . lump
i . . . . . defined upstream lump
j . . . . . defined downstream lump
t . . . . . twin path

Table A-5 FLUINT Solution Algorithm in Pseudo-Code

```
Call FLOGIC 0.
Perform component model simulation calculations.
Perform heat transfer calculations (calculate UA, QTIE, QDOT, update junctions)*
Perform duct macro and reducer/expander spatial acceleration (AC, FG) calculations
Perform connector device simulation calculations (calculate GK, HK, DK, EI, EJ, QTMK).
Perform tube friction factor calculations (calculate FC, FPOW, FD).
Call FLOGIC 1.
Estimate impending changes in hard tanks.
Calculate hard tank property derivatives.
Calculate soft tank intensive property derivatives.
Find allowable time step.

[PERFORM THERMAL SUBMODEL SOLUTION. REPEAT ABOVE ONLY FOR BACKUPS]

Calculate soft tank extensive property derivatives.
Fill coefficient matrix and constant vector.
IF matrix structure has changed THEN:
        Swap primary workspace with secondary.
        If old structure inappropriate redo symbolic solution.
END IF.
Perform matrix factorization and solution.
IF resulting solution is not acceptable THEN:
        Reset parameters and reduce the time step.
        Return to the start of the integration.
END IF.
Update tank states and path flow rates.
Update junction solution.
Call FLOGIC 2 if transient or if finished with steady-state.
Call OUTPUT CALLS if needed.
```

E.1 Basic Governing Equations

Lump Equations—Lumps are assumed to be perfectly mixed and hence are described by a single thermodynamic state. Energy and mass are conserved at every lump. If more than one constituent exists in the model (whether or not it currently exists in any one lump), then a separate equation is used to conserve constituent mass as well.

Plena are sources or sinks, supplying any necessary mass or energy needed to maintain a given thermodynamic state. There are no governing equations for plena—they appear only as boundary condition terms in other equations.

* Junction solution iterates hydrodynamic and energy equations until junction states converge.

Junctions have no volume and therefore no storage capacity; mass and energy flows are balanced at all times. Junctions may have a heat source/sink term, QDOT (composed of QL, QTM, QTIE, QF terms). The equations for a junction are algebraic conservation equations for mass and energy:

$$\text{Mass:} \quad \sum_k e_k \cdot FR_k = 0$$

$$\text{Energy:} \quad \sum_k e_k \cdot h_k \cdot FR_k + QDOT_l = 0$$

Tanks have a definite volume and therefore mass and energy storage capability. They have the capability to grow or shrink with time according to VDOT, and may also grow or shrink with pressure according to COMP. Tanks also have a source or sink term for heat, QDOT. The equations for tanks are therefore differential conservation equations for mass and energy:

$$\text{Mass:} \quad \sum_k e_k \cdot FR_k = \frac{dM_l}{dt}$$

$$\text{Energy:} \quad \sum_k e_k \cdot h_k \cdot FR_k + QDOT_l - PL_l \left(VDOT_l + \frac{dPL_l}{dt} \cdot VOL_l \cdot COMP_l \right) = \frac{dU_l}{dt}$$

Note that the assumption of perfect mixing means that vapor and liquid phases must be in thermal equilibrium in a two-phase tank *at all times*. This means, for example, that adding a little subcooled liquid to a saturated tank will cause an immediate pressure and temperature drop. Nonequilibrium simulations may be handled by subdividing control volumes and utilizing twinned tanks (Section 3.25).

The donor enthalpy term from path k (h_k) takes into account phase suction options, twinned paths, capillary options, kinetic energy, potential energy, etc. As such, it may not be the same value for all paths flowing out of a single lump. For outflow, the donor enthalpy usually represents the enthalpy of the lump itself, but it may alternatively represent saturated liquid or saturated vapor (or 100% gas). If the lump is a junction, the donor enthalpy may represent some state in between saturation and the upstream state, as needed to balance energy flows. The enthalpy of the lump may therefore differ from that which would be calculated from a mixing of the incoming flows because of single phase suction out of the lump.

Path Equations—Paths are assumed to be one-dimensional and hence described by a single mass flow rate. The assumption of one mass flow rate means that vapor and liquid move at the same velocity, with no phase drift or slip flow. Just as imperfect mixing analyses require more than one control volume, unequal phase velocities simulation requires more than one parallel path (as provided by the optional use of twinned paths).

Tubes conserve momentum—the flow rate in tubes cannot change instantaneously. Rather, the fluid inside of the tube must undergo acceleration or deceleration. Hence, the governing equation for tubes is simply a complex form of Newton's second law:

$$\frac{dFR_k}{dt} = \frac{AF_k}{TLEN_k}\left(PL_{up} - PL_{down} + HC_k + FC_k \bullet FR_k \bullet |FR|_k^{FPOW_k} + AC_k \bullet FR_k^2 - \frac{FK_k \bullet FR_k \bullet |FR|_k}{2 \bullet \rho_{up} \bullet AF_k^2}\right)$$

The exponent FPOW is a function of the current flow regime, and may vary from 0.0 (for laminar) to nearly 1.0 (for fully turbulent). Note that both the FC term and the FK term always result in a retarding force, independent of flow direction, while the AC term is a retarding force is one direction and a boost in the other.* The inertia within a tube is proportional to the density times TLEN/AF (i.e., ρL/A). While the TLEN/AF appears inverted in the tube equation, the user should note that the density is contained within the FC and AC terms.

For twinned tubes, the above momentum equation becomes:

$$\frac{dFR_k}{dt} = \frac{AF_k}{TLEN_k}\left(\alpha_k (PL_{up} - PL_{down} + HC_k) + \frac{1}{\alpha_k}\left(FC_k \bullet FR_k \bullet |FR|_k^{FPOW_k} + AC_k \bullet FR_k^2 - FK_k \bullet FR_k |FR|_k / (2\rho_k AF_k^2)\right)\right) +$$

$$\frac{1}{\alpha_k} FD_k \bullet FR_k - \frac{1}{\alpha_t} FD_t \bullet FR_t + \varepsilon_k \left(\frac{1}{\alpha_k} FG_k \bullet FR_k + \frac{1}{\alpha_t} FG_t \bullet FR_t\right) +$$

$$\alpha_t \bullet AM_t \bullet \frac{dFR_t}{dt} - \alpha_k \bullet AM_k \bullet \frac{dFR_k}{dt}$$

Reversing the subscripts k and t results in the equation for the second twin.

Connectors can change flow rates instantaneously. They are governed by a linear algebraic constraint equation between flow rate and pressure differential. This generic equation is a vehicle used for the simulation of a wide variety of devices, and does not represent any conservation relation. It is expressed only in differential form; that is, for changes in flow rate:

Constraint: $$\Delta FR_k^{n+1} = GK_k^n (\Delta PL_i^{n+1} - \Delta PL_j^{n+1}) + HK_k^n$$

* For stability, the AC term maybe subdivided into terms that are linear with flow rate and constant. If this happens (and the AC term is reduced), the effective AC term (before such reductions) may be viewed in path tabulation output routines as "ACeff," and the total force term as "DP ACCEL."

In the above equation, GK is the current partial derivative of flow rate with respect to pressure differential:

$$GK_k = \frac{\partial FR_k}{\partial (PL_j - PL_i)}$$

Note that this partial is evaluated at the flow rate that *would* exist at the current pressure differential, not at the current flow rate. This distinction may seem trivial, but it greatly increases the stability of the algorithm. HK is defined as this hypothetical flow rate minus the current flow rate:

$$HK_k = FR_k^{n+1} - FR_k^n$$

For twinned STUBE connectors, then the full equation is:

$$\Delta FR_k^{n+1} = GK_k^n(\Delta PL_i^{n+1} - \Delta PL_j^{n+1}) + HK_k^n + EI_k^n \cdot \Delta HL_i^{n+1} + EJ_k^n \cdot \Delta HL_J^{n+1} + DK_k^n \cdot \Delta FR_t^{n+1}$$

Using more partial derivatives as defined below (also evaluated at the next hypothetical state):

$$EI_k = \frac{\partial FR_k}{\partial HL_i} \qquad EJ_k = \frac{\partial FR_k}{\partial HL_j} \qquad DK_k = \frac{\partial FR_k}{\partial FR_t}$$

E.2 Formulation of the Equation Set

The equations actually solved in FLUINT are variations of the above set. Iterative solutions of these equations are not feasible except in STEADY—they must be solved simultaneously. This fact has three ramifications on the solution scheme: (1) the equations must be linearized, (2) the number of equations (i.e., order of the solution) must be minimized, and (3) a sparse matrix algorithm should be used. In a network consisting of L nonplenum lumps and P paths, the solution vector is 2L+P long. Actually, for submodels with N constituents, M soluble substances, and S tanks involved in a subvolume (as a result of IFACEs), the vector is 2L+P+N+M+S long. FLUINT reduces this to a compact set of 2L+N+M+S linear equations with the least restricting assumptions possible. (A smaller set of P equations could be produced, but this would disallow many program features.)

FLUINT uses differential forms of the above equations. In other words, the program solves for differences between the last solution vector and the next. The basic form of all of the finite differenced differential equations is:

$$\Delta X^{n+1} = \Delta t \left(\dot{X}^n + \Delta \dot{X}^{n+1} \right)$$

This is a first-order implicit method that is "unconditionally stable" when $\Delta \dot{X}^{n+1}$ is known exactly. This means that an arbitrarily large time step may be taken and still produce a stable answer. However, all this formulation guarantees is that the answers won't artificially diverge—no guarantees can be made regarding the accuracy. This is especially true since the original equations were not linear. FLUINT selects an appropriate maximum time step for each method, as described in Section E.6, to maintain a consistent level of accuracy.

The derivative at the next time step, $\dot{X}^{n+1}$, is not known. However, the difference between this derivative and the last one, $\dot{X}^{n}$, can be estimated to a high degree of accuracy. In order to approximate this change in the derivative, it is necessary to make some assumptions. These assumptions are listed at the end of this subsection. Note that every element (e.g., lump and path) is completely described by the attributes of the adjacent elements, and not by those of more remotely located elements. Because of this, the network can be arbitrarily constructed and the above equation is comparatively simple. In mathematical terms, the Jacobian of the coefficient matrix for this system is trivial.

The algebraic equations are similarly expressed in differential form. Higher order terms in the junction energy equation must be neglected to avoid nonlinear equations. Because of this and because of computer word round-off errors, a fast iterative correction to the junction mass and energy flows is performed after each solution step. As long as flow rate does not change too much during this interval, the associated error is acceptable.

When the tube momentum equation is finite differenced into the form described above, it can be expressed in the same form as the connector equation once the time step is known.

Thus, during each solution interval, all paths are only dependent on the changes in pressure differential and perhaps the enthalpy of endpoint lumps. This allows the final equation set to be reduced to 2L, representing an energy and a mass equation for each tank and junction. Because of the reduced size of the solution set, the mathematical expressions for the remaining equations are very lengthy and cumbersome, and so will not be presented.

The following is a list of major assumptions made within FLUINT necessary to arrive at the present formulation:

1) Lumps are perfectly mixed at all times and are at thermodynamic equilibrium (i.e., described by one state). This does not preclude dissolved gases from being out of equilibrium between each phase, nor does it preclude adjacent lumps (twinned tanks) from modeling phases that are out of equilibrium with adjacent phases.

2) Paths have one flow rate. All fluid within the path moves at the same velocity: the flow profile is one dimensional.

3) Tube attributes (AC, FC, FPOW, HC, FD, FG, AM, FK, etc.) are constant over the solution interval. Connector attributes (GK, HK, EI, EJ, DK) are likewise constant for each interval.

4) Liquid is incompressible (infinite speed of sound) by default (if the COMPLIQ option is not used): density is a function of temperature only for liquids.

5) Specific thermodynamic derivatives (e.g., the partial of specific enthalpy with respect to density at constant pressure) for tanks are constant over each interval.

6) During the solution interval, the operation of each path is a function only of end states (lumps on each end), and the operation of each lump is a function only of the mass and energy conducted to it by adjacent lumps and of the heat and volume rate terms imposed on it. These rates (QDOT and VDOT) are also constant over the solution interval, as is COMP.

7) The second order term $\Delta h^{n+1} * \Delta FR^{n+1}$ in the junction energy equation can be neglected (if need be) during the solution, and corrected iteratively afterwards along with round-off induced mass flow rate imbalances. The first order term $h^{n+1} * \Delta FR^{n+1}$ could be included, but is also usually neglected to avoid destabilizing error terms as a result of the above assumption. In other words, junction enthalpy changes during the main solution are due to upstream enthalpy changes and not to flow rate changes. (Junctions and connectors are updated before and after main solutions, which rectifies this truncation.)

   If twinned paths are active, their EI and EJ terms become nonzero. The summation term $\Sigma EI - \Sigma EJ$ is usually then nonzero for junctions adjacent to those paths, in which case the $h^{n+1} * \Delta FR^{n+1}$ terms in their energy equations must be reinstated along with the Δh^{n+1} term in their mass equations. Fortunately, no stability problems are caused in this special case. (It is the inclusion of such terms that forces the STEADY matrix solution, which is normally strictly a hydrodynamics solution, to include also the energy equations—effectively doubling the vector size.)

8) The term $\Delta X^{n+1} * \Delta FR^{n+1}$ is analogously neglected, where X is the mass fraction of each species within the junction.

9) Tank walls with COMP=0.0 are inflexible (infinitely stiff) if they are not part of a subvolume, and even though they may grow or shrink at user prescribed rates using VDOT).

10) FRD is assumed to stay constant over the solution interval for tanks.

11) Iface parameters VA, VB, PA, PB, and FPV are assumed to stay constant over the solution interval.

E.3 Incompressible Tanks

FLUINT assumes that the liquid state is incompressible by default (if the COMPLIQ option is not used for 6000 series fluids), and therefore density is a function of temperature only. This assumption simplifies the description of the fluid and eliminates the requirement for a time step small enough to resolve liquid compressibility effects such as the propagation of pressure waves. However, since this assumption is also equivalent to an infinite speed of sound in liquid regions, special numerics must be employed to assure that accuracy requirements are met.

Tanks, therefore, are either incompressible (*hard*, even if compliant) or compressible (*soft*). They may change freely from one state to another automatically within the integration, and the user need not keep track of these transitions.

The equations for hard and soft tanks differ greatly from each other. The solution variables for soft tanks are extensive internal energy and mass, and pressure is a dependent variable. Hard tanks, on the other hand, use pressure as an independent variable, with mass a dependent variable. In soft tanks, changes cannot occur discontinuously. In hard tanks, however, pressure may change discontinuously. These facts have significant ramifications on the solution methods used.

For example, the density in a hard tank and the pressure are unrelated, while the net mass flow rate must be always nearly balanced. This can cause instantaneous (junction-like) reactions in hard-filled portions of the network. The events to be aware of include sudden changes in the volume rate (VDOT) in a hard-filled incompressible tank, and, less intuitively, sudden changes in the heat rate (QDOT) and/or the enthalpy flow. This last phenomena is caused by thermal expansions or contractions, which must be accommodated by a change in net flow rate in or out of the tank.

Note that giving a tank any compliance, however small, can eliminate sudden and unrealistic pressure spikes and notches. Perhaps more importantly, pressure is a time-dependent quantity for compliant tanks, which means that the time step controller (described below) assumes responsibility for its variation. (Otherwise, the time step controller must ignore variations in time-independent quantities.) If the tank participates in a subvolume using an iface, it “inherits” some compressibility from its neighboring tanks.

E.4 Phase Suction Options

FLUINT allows violations of the assumption of homogeneous flow. The user may specify that a path accepts only one phase from such a lump when that phase is present. This may be used to simulate stratified tanks, liquid/vapor separators, and is used internally with twinned paths and with capillary devices. All correlation routines obey the current phase suction status when predicting heat transfer, frictional and acceleration pressure drops, and body forces. See Section 3.11.1.

In order to accommodate such an option, a further assumption is made in these cases: the specific enthalpy of the phase being extracted is constant over the solution interval if there is enough of that phase available and the upstream lump is expected to remain in a two-phase state over the length of the next solution interval. This does not apply to junctions or STEADY tanks for which the outflowing phase suction is incompletely satisfied, in which case the enthalpy term is allowed to vary.

E.5 Interface (iface) Implementation

For each tank in a subvolume (see Section 3.8 in general and Section 3.8.1 in particular), there is an extra equation added that replaces the “VDOT + COMP*VOL*dPL/dt” term in the tank energy equation with a $\Delta VOL/\Delta t$ term. In other words, the tank volume becomes an independent variable instead of a dependent one. The corresponding equations are not the same for all tanks. If there are S tanks in a subvolume, then the extra equations correspond to S-1 iface pressure/volume equations (Section 3.8.2.1) plus one “conservation of volume” equation (Section 3.8.2.3). Because ifaces can

become active or inactive at any time, subvolumes can be created, destroyed, joined, or divided each solution step.

E.6 Time Step Algorithm

FLUINT predicts the maximum time step allowed by each submodel to preserve the desired accuracy. This maximum time step is calculated based on both the current and previous (i.e., last time step) network status, and represents a conservative estimate.

Basically, the time step is chosen such that no key variable changes by more than DTSIZF (or RSSIZF for STDSTL) percent. These key variables include all whose derivative is known, such as tank mass, volume, extensive internal energy, and pressure (if soft). Tube flow rate changes are analogously controlled by DTTUBF (or RSTUBF).

The time step predictor must also restrict the allowable time step if a transition is imminent. These transitions include hard to soft and vice versa, the amount of time until a particular phase is depleted, anticipated deprime or reprime of capillary devices, and reversals in path flow rates. Transition time steps are calculated such that the impending transition can occur but the solution does not go very far past the discontinuity.

Finally, the solution will backup and repeat itself if changes are unacceptably large.

Section 4.5.2 describes the mechanics of the time step selection in more detail.

Appendix F Sparse Matrix Inversion

In both steady states and transients, thermal submodels can be solved iteratively, simultaneously alone (each submodel in a separate matrix), or simultaneously together (specified submodels ... usually all of them ... in a single matrix). These options correspond to the MATMET values of 0, 1 or 11, and 2 or 12, respectively.

When MATMET=1 or MATMET=2, the YSMP sparse matrix package is used (Section F.1). YSMP is a direct solver that was developed at Yale University in the 1970s, and was last updated in 1981. “Direct” means that it does not use internal iterations, and therefore does not use an accuracy criteria.

A more modern alternative to YSMP is available that has been specifically designed for large models: an advanced Algebraic Multigrid (AMG) method that internally exploits the conjugate gradient (CG) method as well. This hybrid AMG-CG method (Section F.2) represents an improvement over YSMP in almost all categories, including speed and memory utilization, especially for large problems and problems with dense matrices.*

For this reason, the AMG-CG method is the default for Version 5.2 and later, as invoked by an expanded meaning of the control constant MATMET:

MATMET=11 (each submodel in its own matrix: like MATMET=1, but using the AMG-CG method instead of YSMP), and

MATMET=12 (single matrix for group: like MATMET=2, but using AMG-CG).

In fact, the improved scaling of the AMG-CG method means that there is little penalty for collecting all the nodes into a single matrix (unlike YSMP). Therefore, the default for MATMET is 12: all nodes in a single matrix, solved using the AMG-CG method.

In the rare event that the AMG-CC method is unable to solve the matrix, the MATMET value will automatically be decremented by 10: YSMP is used as a back-up, with iterative methods (MATMET=0) being used if YSMP also fails. The user might also consider setting MATMET=0 and limiting NLOOPS to a few iterations as a network or logic debugging tool.

Mixing of MATMET=2 and MATMET=12 is not allowed: only one type of collected matrix is permitted.

F.1 YSMP (MATMET=1,2 and FLUINT solutions)

The normal solution set and the separated hydrodynamic solution in STEADY are solved simultaneously because their sensitive and highly coupled nature makes iterative solutions almost impossible. The matrix to be inverted is not symmetric but is usually very sparse.

* SINDA matrices can become dense when radiation exists, especially with small values of SPARSEG and/or small values of the RadCAD B_{ij} cut-off factor.

When MATMET=1 or 2 is selected, thermal submodels will also be solved using a matrix inversion.

All such matrix inversions use a three step sparse matrix solution: the Yale Sparse Matrix Package, which is public domain software available from the National Institute of Standards and Technology (formerly the NBS). The first step is a symbolic solution of the nonzero elements. The second and most time consuming step is an LDU factorization using actual numbers. The third step is the final production of the solution vector.

The reason that this package was selected (in 1984) was that not all of these steps need be repeated, saving time for similar matrices. For instance, if the coefficient matrix does not change at all, only the last step is needed after the first pass. If the elements *do* change, but the structure (location of nonzero elements) *doesn't* change, then only the last two steps need be repeated. This last example is relevant to SINDA/FLUINT. If the network is changing gradually, then the coefficient matrix will have different values but the same structure, permitting the symbolic solution to be skipped. This reduces the CPU time to invert the matrix by 5 to 30%.

In SINDA, the matrix structure will almost always remain the same. Usually, it will change only when a conductor is zeroed or a new BUILD command is issued.

In FLUINT, a matrix will maintain the same structure as long as no flow rates reverse direction and as long as no tanks transition from hard to soft or vice versa. Because there are many cases where a solution matrix toggles between one form and another during an integration, FLUINT saves the latest two symbolic solutions: the most recent is called the primary and the older solution is called secondary. Thus, when the matrix structure changes and the primary symbolic solution is no longer valid, FLUINT checks to see if the secondary solution is valid. If so, it becomes primary. If not, a new primary solution is generated. In either case, the former primary solution is saved as the secondary for future use.

F.2 AMG-CG (MATMET=11,12)

AMG-CG relies on recent technology, and is in fact subject to further improvements (unlike YSMP).

Note that there is little advantage to MATMET=11 (individually solved submodels) versus collecting all submodels into a single matrix (MATMET=12), and in fact the MATMET=11 method is often slower and subject to stalling problems if two submodels are tightly connected (share large conductors between them). Therefore, while MATMET=11 is available if needed, the use of AMG-CG is almost synonymous with MATMET=12.

AMG-CG automatically allocates its own memory. However, a trade-off is possible between speed and memory requirements. In the unlikely event that huge model is requiring excessive memory, consider using the METAMG control constant (Section 4.4.1).

Unlike YSMP, it is not a direct solver: the equations are solved simultaneously, but AMG-CG uses internal iterations and is therefore subject to a speed-accuracy trade-off. This trade-off is dictated by the AMGERR control constant (Section 4.4.1). The default setting of AMGERR is small such that it results in approximately the same accuracy as does the direct solver YSMP.

Appendix G Run-Time Control and Plotting

G.1 SINDAWIN Run-time Control

On a Windows PC,the SINDAWIN utility can be used to run SINDA from a text file, as shown in Figure A-1. SINDAWIN.exe is located in the /bin directory of the installation.

Figure A-1 SINDAWIN Utility

Once running from either SINDAWIN or, more commonly as launched from Thermal Desktop, the processor execution can be controlled and monitored as shown in Figure A-2.

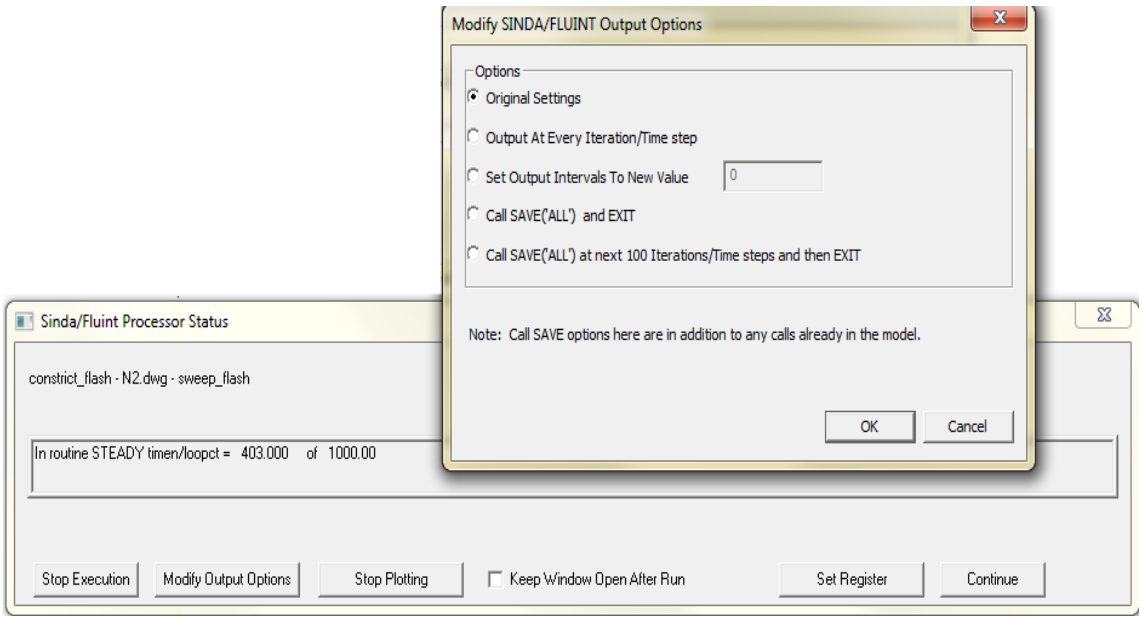


Figure A-2 Processor Status and Run-time Modifications

Stop Execution should be used only to abort a run. Otherwise, the **Pause** and **Continue** button (only one or the other is shown at a time) can be used to investigate or make adjustments to settings.

Although the run need not be paused at the time, **Modify Output Options** opens the pop-up dialog shown at the upper right in Figure A-2. This dialog allows the output interval to be adjusted (including returned to original settings) during a run.

The bottom two options allow the run to be aborted, but unlike the **Stop Execution** button, important outputs are generated first, so those are the preferred means to stop a run. If the model is not converging or is taking tiny time steps, the **Call SAVE('ALL') at next 100 Iterations/Time steps then EXIT** option is particularly helpful since it allows post-run plotting of the problem area using EZXY (H.1) or Thermal Desktop.

The **Set Register** option opens a new dialog that allows the user to either query the value of a SINDA/FLUINT user variable, or to alter its value or even defining expression. This option can also be used at any time, though pausing the run with the **Pause** button is recommended during such operations.

The **Stop Plotting** and **Keep Window Open After Run** options are relevant to the Progress Plotter, described next.

G.2 Progress Plotter

EZXY (Appendix H.1) provides a post-solution means of plotting key variables. For both debugging unconverged models and monitoring progress of a transient, an additional tool is available a during runtime.

The **Sinda/Fluint Processor Status** window shown in Figure A-2 will greatly expand during the run if such extra tools have been invoked, and a dynamically updated plot will appear versus TIMEN or LOOPCT. These choices are best managed from Thermal Desktop's Case Set Manager, and the Thermal Desktop User's Manual contains more information. Nonetheless, brief guidelines for text-based control are included below, documenting additional calls that can be added to OPERATIONS before a solution routine is invoked.

First, set up a plot using various SETPLOT utilities:

`CALL SETPLOTDELAY(1)` ... determines the interval for updating *in wall clock units of seconds*. This is an integer argument. One second is shown here. Enter zero (0) to capture every time step or steady state iteration.

`CALL SETPLOTPOINTS(500)` ... determines the number of points to be shown on the plot for each line. 500 points is shown here, meaning the last 500 points will be shown at any one update. Use a smaller number to focus on more recent data and a larger number to watch a larger run history.

`CALL SETPLOTPPL(1)` ... adds point markers to each line to help distinguish between lines. (If the argument had been zero, or if this call were missing, no point markers would have been added.)

`CALL SETPLOTHOLD(1)` ... leaves the plot visible for final inspection when the run terminates, so that the final results can be reviewed before dismissing the plot and terminating the run. (If the argument had been zero or if this call were missing, the plot will disappear when the run ends.)

Next, up to 15 lines can be added in a plot. These lines can monitor processor variables (like “model.T100”) or they can monitor registers. Each line is added by a call to ADDPLOTVAR. In the following examples, the void fraction in lump 1 of fluid submodel FLOW is added, as is register *Heat_box*. Descriptions can be added, but should be brief.

```
CALL ADDPLOTVAR( FLOW.AL1, 'VOID IN VESSEL')
CALL ADDPLOTVAR( Heat_Box, 'Box Heater Power')
```

The first argument in ADDPLOTVAR must be a translatable expression, since it will be used as a pointer to the actual data to be monitored.

Appendix H External Programs

In addition to the preprocessor executable and the processor library, SINDA/FLUINT includes two separate companion programs that are described in the following sections. These programs are:

1) EZXY®, a PC-based plotting program (that can also plot SAVE or RSO folders created on Unix machines). EZXY is intended for users who do not use Sinaps®, or for those needing histograms to support the Reliability Engineering module.

2) RAPPR, a pre-preprocessor that creates fast alternative fluid descriptions (i.e., FPROP DATA blocks) from standard library descriptions.

H.1 EZXY® Plotting Utility

For users lacking either the Thermal Desktop® suite of codes (see page xliii and page xliv of the preface) or the nongeometric GUI, Sinaps® (see page xlii of the preface), the EZXY plotting program may be used on a PC to view results produced from any SINDA/FLUINT version, including Unix versions. (See page xlvi of the preface.) Even if Sinaps is available with its extensive plotting capabilities, users may find EZXY useful because of its unique capabilities such as multiple axes and histograms.

EZXY, like Sinaps and Thermal Desktop, uses the binary files or CSR folders produced by the SAVE, RESAVE, CRASH, and SAVEDB routines as input. In addition to producing plots, EZXY can be used to read binary data and produce ASCII outputs.

Along with EZXY, spreadsheet templates are installed that allow a Microsoft Excel™ user to read and perhaps plot data directly from SAVE files or folders. Other Excel templates allow SINDA/FLUINT runs to be launched and controlled from within Excel.

EZXY is documented via its own on-line help facility.

H.2 RAPPR: Speed Improvements Using Simplified Fluids

[*RAPPR is no longer distributed nor supported.* It has been obsoleted by the PR8000 and PR9000 routines, as described in Section 7.11.2.5.]

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z