<intentionally blank for 2-sided printing>

Prepared, distributed, and supported by:

C&R Technologies, Inc. (“CRTech”)
Boulder, Colorado USA
Phone: 303.971.0292
FAX: 303.971.0035
Web site: www.crtech.com

Authors:

B. A. Cullimore
S. G. Ring
D. A. Johnson

REVISION HISTORY

Draft Version		November 1984
First Publication	(Version 1.0)	February 1986
Revision 1	(Version 2.0)	August 1986
Revision 2	(Version 2.1)	November 1987
Revision 3	(Version 2.2)	September 1988
Revision 4	(Version 2.3)	March 1990
Revision 5	(Version 2.4)	December 1991
Revision 6	(Version 2.5)	September 1992
Revision 7	(Version 2.6)	September 1993
Revision 8	(Version 3.0)	September 1994
Revision 9	(Version 3.1)	September 1995
Revision 10	(Version 3.2)	October 1996
Revision 11	(Version 4.0)	October 1997
Revision 12	(Version 4.1)	October 1998
Revision 13	(Version 4.2)	February 2000
Revision 14	(Version 4.3)	June 2000
Revision 15	(Version 4.4) <publishing software and font upgrade>	July 2001 December 2001
Revision 16	(Version 4.5)	September 2002
Revision 17	(Version 4.6)	September 2003
Revision 18	(Version 4.7)	September 2004
Revision 19	(Version 4.8)	October 2005
Revision 20	(Version 5.0)	October 2006
Revision 21	(Version 5.1)	October 2007
Revision 22	(Version 5.2)	October 2008
Revision 23	(Version 5.3)	November 2009
Revision 24	(Version 5.4)	February 2011
Revision 25	(Version 5.5)	October 2011
Revision 26	(Version 5.6)	May 2013
Revision 27	(Version 5.7)	August 2014
Revision 28	(Version 5.8)	April 2015

FOREWORD

SINDA/FLUINT has been well received since its first introduction in 1985 (under the name SINDA ‘85). Originally intended to satisfy the ever-changing modeling needs of the spacecraft and launch vehicle thermal community, the program has diffused into other industries such as power generation, aircraft, electronic packaging, HVAC, experimental physics, medical, and automotive, and into other specialties and subsystems such as propulsion and environmental control.

SINDA was originally intended for simulating thermal systems represented in electrical analog, lumped parameter form. Given properly prepared inputs, SINDA can solve both finite difference and finite element equations simultaneously: it is best classified as an “equation solver” than its original classification as a “finite difference solver.” Potential SINDA users who are unfamiliar with thermal system modeling techniques should refer to page xlvii in this preface for training materials.

The FLUINT capability was first added to SINDA in 1986. It is now arguably the most advanced one dimensional thermal/hydraulic code available, solving arbitrary fluid flow networks without presuming the application. The networks may contain single-phase vapor or liquid as the working fluid, or combinations of liquid and/or vapor at various locations in the network, mixtures of working fluids and even chemical reactions. These fluid networks are solved in conjunction with one or more accompanying thermal networks, accounting for the energy flows in and out of the fluid system(s).

Thanks to a new generation of graphical interfaces, SINDA/FLUINT is primarily used as a “solution engine,” with the text-based interfaces being used less and less every year. These graphical interfaces include a nongeometric sketchpad-style Sinaps®, and the geometry-based Thermal Desktop® (for SINDA conduction/capacitance calculations based on finite elements and/or finite differences) with its companion modules RadCAD® (SINDA radiation calculations) and FloCAD® (FLUINT circuits, heat pipes, and convective heat transfer calculations).

Sinaps avoids the need to work with text-based input and output files, allowing users to sketch their models on the screen. It is best suited for system-level analyses, conceptual designs, and fluid schematics.

Analysts requiring radiation calculations, contact conductances, geometric modeling of surfaces and solid parts, data exchange with CAD and structural software, heat pipes, surface recession, two-phase vessels, TEC devices, etc. would be well advised to explore Thermal Desktop (page xliii).

The entire Thermal Desktop suite is fully parametric. More importantly, SINDA/FLUINT can be run dynamically from within it, calling for new Thermal Desktop/RadCAD/FloCAD solutions *during* SINDA/FLUINT execution. This enables parametric and sensitivity analyses to include complex geometric, property, and environmental variations. It also brings SINDA/FLUINT’s Advanced Design modules (Section 5) to be used as drivers for the entire analysis chain. With the Measures and Data Logger features in Thermal Desktop, this means models can be automatically correlated to temperature measurements.

The Thermal Desktop tool suite has received an additional boost from new tools that simplify the preparation and maintenance of thermal models that originate from CAD geometry. These tools include CRTech TD Direct®, an expansion of SpaceClaim®,* which allows import, manipulation, healing and simplification of virtually any CAD part or assembly. TD Direct allows users to mark-

up geometry with properties, boundary conditions, and so forth such that changes update automatically (including re-meshing) to Thermal Desktop. Version 5.6 includes structured and swept meshes, which provide important capabilities to thermal analysts including alternatives to midsurfacing: 2.5D modeling tools. Version 5.7 introduced curved elements, allowing fast-solving system-level models to be constructed without losing mass, volume, and surface area when using coarse meshes. Version 5.8 introduces Compartments and Ports for modeling complex-walled two-phase vessels like fuel tanks and boilers.

The user should also note the availability of tools not documented in either this manual. These include a Microsoft's Excel® interface that can be used to launch, control, and postprocess SINDA/FLUINT or combined SINDA/FLUINT-Thermal Desktop runs. (Excel can also be used to read results files or generate plots as an alternative to EZXY®.) The underlying Microsoft COM-based access to Excel can be exploited for developing other connections as well. For example, a MATLAB®* interface to SINDA/FLUINT also exists for controlling simultaneous execution: a SINDA/FLUINT model as part of a larger MATLAB simulation.

The improvements in this version are summarized in VERSION 5.8 vs VERSION 5.7 on page xl for more detailed information.

* SpaceClaim is a registered trademark of SpaceClaim Corporation.
* MATLAB is a registered trademark of MathWorks, Inc.

VERSION 4.1 vs. VERSION 4.0

The major differences between Version 4.1 and Version 4.0 are summarized in this subsection. The next subsection summarizes differences between Version 4.2 and Version 4.1.

EXPRESSIONS: MINIATURE LOGIC BLOCKS—Spreadsheet-like registers and register-containing expressions continue to be a popular and powerful feature of the code. Expressions, which can be used to define almost any value in SINDA/FLUINT, have been tremendously expanded. First, the user can refer to processor variables such as “fred.T33” or “abszro” or “drlxcc” or “timen” in the expressions, completing the spreadsheet capabilities by allowing inputs to be declared functions of not only other inputs, but also of output (response) variables. Together with the ability to use conditional operators (IF/THEN/ELSE type logic), each expression can be as powerful as a logic block, and in fact many traditional input options and applications of logic blocks are now largely obsolete (although still supported). [See Section 2.8]

SOLUBLE GASES—Mixture properties can be extended to include the effects of gases dissolving into liquids and evolving out of them. Multiple solutes and solvents can be defined. These effects can be included during either steady-state or transient solutions. Applications include non-condensible gases in single- and two-phase thermal transport loops and vapor compression cycles, pressurant gases in liquid propulsion and in fire retardant delivery systems, and water quality assessments. [See Section 3.22 and Section 3.23]

INTERFACE ELEMENTS—A new path-like FLUINT network element, the “iface,” is available for specialized fluid system modeling needs such as subdividing control volumes (including nonequilibrium control volumes or thermally stratified vessels), and for modeling springs, bellows, pistons, servo-control valves, and wicks. [See Section 3.8]

SIMULTANEOUS THERMAL SOLUTIONS—Simultaneous (sparse matrix solution) methods may now be applied to thermal submodels to replace or augment traditional iterative methods in both steady state and transients. Matrix methods are particularly suited to conductively dominated problems (such as translated finite element models from structural meshes) including those in which large conductors are unavoidable. [See MATMET on page 83 of Section 4.4.1]

MISCELLANEOUS IMPROVEMENTS—

(1) The propagation of spreadsheet (register and processor variable) changes is now performed automatically via internal calls to UPREG. These automatic updates can be regulated, supplemented, or even globally disabled.

(2) With the advent of ifaces, the stability and accuracy of nonequilibrium options have been greatly improved. Also, the BELACC option has been rendered obsolete by the introduction of SPRING ifaces. BELACC is available but undocumented.

(3) A new phase change material (PCM) simulation routine, FUSION, is now available for modeling latent energy within a diffusion node.

(4) An improved method has been added for predicting the density of liquid mixtures: the assumption of additive volumes has been supplanted with the modified Hankinson-Brobst-

Thomson method. Therefore, the PVF parameter has been globally replaced with MF, the molar fraction (a real, floating point variable).

(5) Negative values of IPDC may be used to impose a user-specified flow regime (turning off the automatic selection available when IPDC=6).

(6) Pressure regulating values and back-pressure regulators are now available as PRVLV and BPRVLV connectors.

(7) NEADCC has become obsolete. The code always adds an extra constituent equation to assure accurate tracking of trace species.

(8) As a convenience in switching between junctions and tanks, junctions now have volumes (“VOL”). These volumes are ignored by most program options, important exceptions being the SUMFLO routine and several gas dissolution/evolution options.

(9) A routine exists for estimating irrecoverable (K-factor) losses in reducers and expanders: FKCONE.

(10) Two new SINDA output routines are available: HRPRINT for printing heat rates through conductors, and NODTAB for tabulation-style summaries of nodal attributes including energy imbalances.

(11) CONSTRAINT DATA, part of the Solver, now accepts “unnamed” constraints to simplify the implementation of common constraints. For example, the user can specify “sub1.T15 <= 200.0” directly in CONSTRAINT DATA rather than first defining a constraint variable (e.g., “THEAT <= 200.0”) and then updating that constraint variable later in logic blocks (e.g., THEAT = sub1.T15).

(12) CONTRN, NODTRN, INTCON, and INTNOD dynamic translation routines for nodes and conductors have been rewritten to execute much faster, especially for repeat calls. This enhancement is transparent to the user.

(13) Sparse matrix inversion underlying FLUINT solutions (and now optionally applied to SINDA solutions as well) have been expanded to include an ordering optimization which reduces both run times and memory requirements. This enhancement is transparent to the user.

(14) The model size limits (number of nodes, etc.) have been increased. Refer to Section 6.19.

VERSION 4.2 vs. VERSION 4.1

As an aid to users of Version 4.1, the major differences between Version 4.2 and Version 4.1 are summarized in this subsection. With one notable exception, previous models are accepted with little difference in execution.

FTIEs—Heat can flow directly between lumps using *fties* (fluid-to-fluid heat transfer ties), a new network element. Advective transport of energy is usually always dominant over conductive transport within a fluid. However, in systems with very slow flow rates or that use highly conductive fluids (such as liquid metals), neglecting conductive transport may not be acceptable. Another use of fties is modeling thin-walled heat exchangers between two sections of the same fluid system (e.g., regenerators). Fties may be applied with a constant or user-varying conductance ("UF"), or with a constant or user-varying heat flux ("QF"), or as a companion to a tube or STUBE connector that automatically updates the ftie conductance to take into account the current fluid flow within those paths. In fact, such an "AXIAL" ftie can easily be generated along with any tube or STUBE connector. [See Section 3.7]

Auxiliary routines (WETWICK, PLAWICK, CYLWICK) are available to calculate and apply ftie conductances for flows through conductive porous media. [See Section 7.2]

TWINNED TANKS—Whenever the assumption of perfect mixing in two-phase control volumes (equal temperatures and pressures between liquid and vapor) is inappropriate,* tanks can be twinned. When twinned, the code will use one tank to model the vapor and one tank to model the liquid within the control volume, analogous to the way twinned paths may be used to solve separate momentum equations for each phase. In fact, twinned paths may be used in combination with twinned tanks for full nonhomogeneous, nonequilibrium modeling. Other elements such as ties and fties may also be twinned to facilitate their application to twinned tanks.

Twinned tanks use fties and ifaces between each pair, and also apply a new concept called a *superpath* to handle mass transfer between them. Most of these calculations and manipulations are transparent to the user when default options are used. However, in order to provide full access and control for custom solutions, the number of new variables and features is considerable. For this reason, and because twinned tanks will only be needed in specialized cases, the new features are described in a separate section. [See Section 3.25]

Twinned tanks make use of the NONEQ family of routines obsolete. These routines are still available, but are now undocumented. Users are encouraged to migrate to twinned tanks, which (unlike the NONEQ routines) can be used within LINE and HX macros, can be used with twinned paths, and can handle dissolution/evolution phenomena. The NONEQ routines will eventually become unavailable.

---

* Examples: fluid within quasi-stagnant vessels, wave propagation within ducts containing both phases, strong circumferential temperature gradients caused by severe environments.

CAUTION: IPDC=6 as Default—Because of the preponderance of options requiring flow-regime mapping in modern SINDA/FLUINT, the prior default of IPDC=1 (homogeneous pressure drop, unmapped regime) has been changed to IPDC=6 (automatic regime selection and pressure drops and other calculations based on regime). *This may cause two-phase models built for prior versions to give difference results.*

CAUTION: QDOT versus QL—In previous versions, a heating rate, QDOT, could be applied to a lump, or QDOT could be calculated automatically if ties were attached to it. To allow for the expansions associated with fties, and to permit users to add an additional source onto a tied lump, the prior meaning of QDOT as both an input and output variable has been split. A new input variable, "QL", now defines the heat applied to a lump *in addition to* heat flowing to it from nodes (via ties) and from other lumps (via fties):

$$QDOT = QL + \Sigma QTIE + \Sigma QF$$

QDOT retains its original meaning as the total power applied to a lump, but can no longer be used as an input variable: it is now strictly an output variable. ***This means that occurrences of "QDOT =" in FLOW DATA sections will now cause a preprocessor abort***. They must be changed to "QL =" *and* the user must assure that this power is intended as an additional dissipation (if positive) if ties are present on that lump to get the same results as before. *Each occurrence of QDOT in logic and expressions must be examined* to see whether it is appropriate to refer to the applied power QL, or to the total resulting power QDOT.

MISCELLANEOUS IMPROVEMENTS—

(1) On PC f90/f95 versions, memory is allocated dynamically for sparse matrix solutions (which are used for all FLUINT solutions as well as any SINDA submodels using MATMET=1 or 2). This eliminates excessive allocations, aborted runs, and any need to use the NWORK or NWORKS options. No user involvement is required to use this feature.

(2) A replacement of the INCLUDE command, the INSERT command, has been introduced. INCLUDE will still be accepted and is documented but no longer recommended. INSERT works the same as INCLUDE, except that an INSERTed file can also contain another INSERT command: INSERT commands can be nested indefinitely.

(3) The use of negative signs on the multiplication factors ("F") of SIV and SPV conductors has been historically used as a signal to use a Node A's temperature instead of an average temperature of the two end-point nodes. New conductor options (SIVM, SIVA, etc.) have been introduced to avoid conflicts with more modern usage (expressions, FEM conversions).

(4) A routine, PUTPAT, has been introduced to allow paths to be relocated between lumps.

(5) A new routine, CHGEXP, allows the expressions defining parameters to be added or changed dynamically (during processor execution).

(6) A new utility, QFLOW, exists for reporting the heat flows from one or more nodes to any other collection of nodes. Nodes defining each group ("to" or "from") may be specified individually, by submodel, or as arbitrary lists.

(7) Auxiliary modeling tools have been added to make modeling of wyes and tees (merging and diverging flows) more convenient, and to make manifold modeling more accurate. These tools include correlation routines (YTKALI, YTKALG) and a K-factor conversion utility (YTKONV).

(8) A variation of the SUMFLO routine, SUMDFLO, has been introduced. Unlike SUMFLO, SUMDFLO takes into account path duplication factors when calculating total network mass, volume, etc.

(9) A new routine, HTRNOD, allows any arithmetic or diffusion node to temporarily become a heater node (a type of boundary node). The action is released by RELNOD. These actions may be performed on all arithmetic and/or diffusion nodes within a submodel using HTRMOD and formulas.

(10) The HEATER (thermostatic heater) routine has been expanded to function during steady states using an internal call to HTRNOD.

(11) A new proportional heater routine, PHEATER, is now available.

(12) Multiple header blocks of the same type will no longer cause errors as long as they are syntactically identical. If encountered, such redundant blocks will be joined into a single block before preprocessing.

(13) Short-cut commands now exist to build all available thermal or fluid submodels into the current configuration: BUILD ALL, BUILDF ALL. Also, all current thermal models can be removed from the current configuration using “BUILD NONE”, as can all fluid submodels using “BUILDF NONE.”

(14) An empty (fake) “HEADER NODE DATA” is no longer required for conductor-only submodels.

(15) SAVE, RESAVE, and SVPART now accept a minus flag syntax meaning “save all except the following.” For example, “-R” means “save everything except registers,” a convenient option when using SVPART and the Solver.

(16) For the convenience of new users, the most commonly used steady-state routine (FASTIC) may now be referred to using the alias “CALL STEADY.” the most commonly used transient routine (FWDBCK) may now be referred to using the alias “CALL TRANSIENT.”

(17) Internal improvements have been made in the Solver algorithms, especially for METHO=2 in constrained problems.

(18) Slight nonconservation of mass and energy in transients due to unavoidable numeric truncation and similar causes has been rectified by accumulating error terms and folding them back into the solution, improving the accuracy of the predictions over several time steps. (Refer to the new RMFRAC control constant.)

(19) A new control constant, FRAVER, has been added to allow the FLUINT user to denote a characteristic average flow rate for a loop such that it can better decide what level of flow can be neglected as zero and what amount of change in flow rate is negligible per time step.

(20) A variation of USRFIL called USRFIL2 is available that does not abort if it can't open a file. Instead, it returns an error flag.

(21) A new PC-based plotting package, EZXY™, is now available on PCs for those users who do not use Sinaps®. EZXY produces plots or text from SINDA/FLUINT SAVE (or RESAVE) files. [See page xlvi]

(22) C&R Thermal Desktop® has been available since December, 1998 ... shortly after the release of SINDA/FLUINT Version 4.1. Thermal Desktop is a CAD-based geometric interface to SINDA, providing CAD (IGES, STEP, DXF, etc.) and FEM structural communication pathways to permit concurrent engineering without sacrificing traditional thermal modeling practices. RadCAD®, available since 1997, is now available as the radiation analysis module within Thermal Desktop. [See page xliii]

VERSION 4.3 vs. VERSION 4.2

As an aid to users of Version 4.2, the major differences between Version 4.2 and Version 4.3 are summarized in this subsection. Previous models are accepted with little difference in execution.

Reliability Engineering—A new top-level module has been added that enables analysts to treat unknowns and uncertainties statistically, estimating the chances of exceeding arbitrarily complex failure limits. Dimensions, properties, boundary conditions, usage scenarios, etc. are normally not fixed quantities, but can assume different values within a prescribed range. Margins and safety factors are normally used to deal with such uncertainties, stacking up worst-case scenarios and minimum/maximum values to produce conservative designs. The Reliability Engineering module allows the degree of over- or under-design to be quantitatively measured by predicting the chances that the design will pass muster: the reliability of the design. Combined with the Solver, it can be used to synthesize a design that meets reliability requirements up front, intelligently balancing cost against risk. [Section 5]

MISCELLANEOUS IMPROVEMENTS—

(1) Thermal Desktop and RadCAD calculations can now be made dynamically from within PC (4.3L or 4.3D) SINDA/FLUINT launched from within Thermal Desktop. This major expansion enables parametric analyses, sensitivity studies, optimization, correlation, reliability estimation, and robust design activities to include variations in geometry, optical properties, thermophysical properties, insulation, contact conductance, orbital parameters, etc. The traditional separation of Thermal Math Modeling (TMM) and Geometric Math Modeling (GMM) has been eliminated. Refer to the Thermal Desktop User's Manual for more information, or contact CRTech.

(2) A link with Engineous' iSIGHT® software has been created.

(3) Psychrometric utilities have been added for convenience in dealing with the dew points and relative humidities in air/water mixtures (as well as other mixtures of condensibles and noncondensibles). [Section 7.11.2.4]

(4) PID controller simulation utilities are now available. [Section 7.7.8]

(5) By default, SINDA/FLUINT now checks array data to see if mixed integer and real data have been input within a single array. This prevents the common mistake of forgetting the decimal point for a real value. This option can be overridden (e.g., when bivariate or trivariate arrays are input) by specifying MIXARRAY in OPTIONS DATA.

(6) Pump and fan curves may now be non-monotonic: they may include regions with positive slopes in the head vs. volumetric flow rate curve.

VERSION 4.4 vs. VERSION 4.3

As an aid to users of Version 4.3, the major differences between Version 4.3 and Version 4.4 are summarized in this subsection. Previous models are accepted with little difference in execution.

High Speed and Compressible Flow Improvements—Although FLUINT always handled the full thermodynamic and momentum effects associated with compressible flow (including choking), it previously made the assumption of kinetic energy being negligible compared to thermal energy (enthalpy-based or advective transport).* Kinetic energy transport is now included in the flow solutions, such that certain high speed flow phenomena (e.g., Fanno line flow, etc.) can be modeled.

While this enhancement is mostly transparent to the user, there are a few subtle changes and cautions required. First, *all* paths now have a definable flow area (AF) and throat area (AFTH), although their use is optional for many connector device types. If a path has no defined flow area, then kinetic energy cannot be taken into account and the flow will "stagnate" upstream of that path, perhaps to be accelerated again downstream. Also, the optional choking calculations (CHOKER, CHKCHL, etc., Section 7.11.9.5) no longer assume that the upstream lump represents stagnation, and therefore an expansion process need only be included if the throat area is less than the flow area (AFTH < AF). Finally, any lump in which velocities *should* be assumed zero can be marked as such using a new "LSTAT=STAG" option in lump subblocks (Section 3.3). This designation affects not only choking and kinetic energy calculations, it will also automatically add an accelerational entrance loss (equivalent to FK=1.0) to any appropriate path (tube, STUBE, loss, etc.) attached to and flowing out of that stagnant lump.

MISCELLANEOUS IMPROVEMENTS—

(1) Previously, simultaneous (sparse matrix based) thermal solutions could be invoked on a submodel by submodel basis using MATMET=1. Submodels that are tightly coupled can now be combined into a single matrix using MATMET=2 (Section 4.4.1). Other values of MATMET retain their original meaning: MATMET=0 means iterative, and MATMET=1 means simultaneous (per submodel).

(2) A network connectivity checker has been added that detects isolated groups of nodes: nodes having no connection to a boundary node in steady states, and arithmetic nodes having no connection to either boundary or diffusion nodes in transients.

(3) Thermal submodel size limits have been greatly increased (Section 6.19).

(4) In an attempt to avoid confusion for new users, the built-in spell checker, controlled by the NODEBUG (Options Data) and DEBOFF and DEBON (logic blocks) options, may now be controlled using the SPELLON and SPELLOFF commands instead (Section 2.7, Section 4.1.1.3). The old options are still available but have become undocumented.

(5) In connector subblocks, general path inputs (FR= ..., DUP = ... , STAT = ... etc.) are now tolerated following the "DEV=" specification.

---

* Potential energy is even more negligible, but has been included previously.

(6) Defaults now exist for NLOOPS, OUTPUT, and OUTPTF to avoid aborted runs. The default for EXTLIM has been reduced from 50 to 5 to avoid instabilities in models with fluid systems, heat pipes, etc.

(7) SINDA now tolerates nodes with zero (or absent) conductors as long as ties are present.

(8) The Fortran reserved word list has been expanded to include many f90/f95 keywords (“WHILE”, “ENDDO”). This prevents erroneous spell checker (Section 4.1.1.3) halts, but also prevents these names from being used as registers, named control data, etc.

(9) RESTAR, RESPAR, RESTDB, etc. now check the input record number for validity.

(10) A new routine, HEATPIPE (Section 7.7.9), has been added to assist in simulating fixed (constant) conductance heat pipes and variable conductance heat pipes.

(11) CSGFAC can now be used as a CSGMIN divisor for time step control in TRANSIENT as well as FORWRD. In TRANSIENT, use of negative DTIMEI is now taken to mean overriding of the default (DTIME=0) internal time step control, using instead a time step equal to CSGMIN/CSGFAC. Positive DTIMEI retains its original meaning as the directly specified time step.

VERSION 4.5 vs. VERSION 4.4

As an aid to users of Version 4.4, the major differences between Version 4.4 and Version 4.5 are summarized in this subsection. Previous models are accepted with little difference in execution.*

Conductor HR values†—The absolute value of the heat rate through a conductor is now available as the parameter "HR." This output variable may be accessed in logic or expressions. Note that because an absolute value is used, there is no dependence of HR on direction: no "from node" nor "to node." [See Section 2.14.3]

Built-in Sonic Limits (Choking)—By default, almost all paths (including tubes) are now checked for sonic limits in their throats (AFTH), and if such limits are detected they are automatically modeled. [See Section 3.18]

This new system makes the CHOKER and CHKCHP routines obsolete, and to avoid confusion their use is discouraged though still accepted for backward compatibility. Users are encouraged to delete those calls from their models, and replace them with appropriate MCH and HCH values.

Unfortunately, this new default (checking for choking everywhere and applying nonequilibrium expansions if needed) stresses many fluid property descriptions, hitting lower limits or requesting off-chart thermodynamic properties. The user may need to selectively or globally turn off such choked flow modeling (using MCH=0) if improvements to the underlying fluid properties cannot be made and if choking is deemed to be irrelevant to the problem at hand. In other words, the default was chosen as a safety measure, and it may be irrelevant or disruptive in some models.

Expanded Heat Pipe Modeling—A new routine, HEATPIPE2, has been introduced to extend the prior HEATPIPE routine to cover pipes with strong circumferential gradients: the use of 2D heat pipe models using with more than one node to each circumference, and including flat-plate vapor chamber fins. Other improvements include the ability to disregard incorrectly sized/charged VCHPs, greater flexibility of input units (HPUNITS), and a new routine (HPGLOC) is available to extract the gas front location from previous HEATPIPE or HEATPIPE2 calls. [See Section 7.7.9] (Heat pipe users are urged to review recent FloCAD® enhancements specifically designed for heat pipe modeling inside 3D FEM/FDM thermal models.)

MISCELLANEOUS IMPROVEMENTS—

(1) Many model size limits have been relaxed or even eliminated (for versions based on Fortran 90/95 compilers, which currently includes all PC versions). [See Section 6.19]

(2) More complex syntax is now allowed for conditional operators inside of spreadsheet-like expressions. For example, "heat = ((sub.t1 > 300) && (sub.t1 <= 400))? on : off" meaning "if the temperature of node 1 in submodel *sub* is above 300 degrees but less than or equal to 400 degrees, set the register *heat* to *on*, otherwise set it to *off*." [See Section 2.8.7]

---

* Users of the PID routines represent an exception: See Section 7.7.8 for more details.

† To enable this option, "conductors with multiple node pairs," an undocumented and discouraged feature of SINDA, was finally disabled. However, multiple node pairs can be reinstated by using the NODEPAIR command in OPTIONS DATA, but doing so disables the use of conductor HR.

(3) Routines for sizing and/or simulating Peltier devices, including Thermoelectric Coolers (TECs) are now available. [See Section 7.7.10]

(4) A new connector device has been added to simplify modeling of common orifices: the ORIFICE connector. Built-in correlations for sharp-edged and long orifices are available, or the user can provide their own coefficients. The resistance of the orifice is estimated, as well as the prediction of its vena contracta for the purposes of choked flow detection and modeling. [See Section 3.5.12]

(5) A new connector device has been added to allow users to specify how a component behaves (pressure drop versus flow rate characteristics) using tables input as ARRAY data: the TABULAR connector. The tables can use a variety of units for pressure/head and flow. Either a single curve can be input, or a family of parametric curves based on a user-defined independent parameter. While intended to assist in the modeling of valves and other devices using vendor-supplied data, TABULAR devices represent a general-purpose means of describing a flow resistance for a variety of components. [See Section 3.5.16]

(5) Much greater flexibility now exists for defining units for pump curves, enabling users to apply vendor data directly without conversions. [See HU and GU in Section 3.5.15]

(6) To avoid any confusion, the PRVLV and BPRVLV connectors have been renamed to UPRVLV and DPRVLV connectors, respectively. The older designations are still accepted in input files, but are no longer documented to discourage their continued use.

(7) A new utility routine, REPATH, has been added to allow users to easily estimate the Reynolds number for any tube or STUBE.

(8) Double precision is now on by default.

(9) The values of ITHOLD and ITHLDF now default to zero: the temporary suspension of submodels during steady states (an attempt at speed savings that is not universally successful) will no longer be performed by default.

(10) The COMPLQ routine has been extended slightly to apply a proportioned compliance to two-phase tanks. Previously, COMPLQ only applied compliance to hard-filled liquid tanks. This change avoids a discontinuity at the saturated liquid line.

(11) A new option (YTHUSH routine) is available to eliminate fatal errors when the wye-tee (converging/diverging flow) routines such as YTKALG and YTKALI encounter conditions for which their correlations do not apply. Use of this option means the user accepts the burden of checking the final results, but permits both steady-state excursions (temporarily reversed flow directions, for example) as well as extrapolations of the correlations.

(12) A change has been made to the PID controller routines formulation: they now return values of the control variable (CV) instead of changes to the control variable (ΔCV). ***This change is not backward compatible with previous models.*** [See Section 7.7.8]

(13) New phase suction options, STAT=LRV and STAT=VRL, are available.

SIGNIFICANT EXTERNAL PROGRAM IMPROVEMENTS—

(1) An Excel-based utility for reading and plotting results files (all the binary files produced by the SAVE, RESAVE, SAVEDB etc. routines) is now available as part of the EZXY® installation.

(2) FloCAD® has been significantly expanded to include “pipe” methods for drawing 1D lines representing fluid ducts and heat pipes, and for attaching these to Thermal Desktop® surfaces and solids. These improvements make FLUINT modeling significantly easier for common pipe-flow and heatpipe problems.

(3) C&R Thermal Desktop® now features native finite difference solids for enhanced parametric modeling.

VERSION 4.6 vs. VERSION 4.5

As an aid to users of Version 4.5, the major differences between Version 4.5 and Version 4.6 are summarized in this subsection. Previous models are accepted with little difference in execution.

Convection Heat Transfer Expansions—Major expansions of convection heat transfer (FLUINT ties etc., Section 3.6) have taken place, including:

(1) Many new scaling factors and customization options for all heat transfer regimes, but especially for single-phase heat transfer including high pressure, near critical, noncircular, and other specialized regimes. (See parameters CDB, UER, UEH, UEV, UEC, UEP, UET, UECP, UED, XNUL, XNLA, XNLM, XNTA, XNTM, and MODR.)

(2) Expanded treatment of boiling heat transfer, including critical heat flux limits and a film boiling regime, and also inclusion of subcooled boiling and superheated condensation. (See parameters XNB, CHFF, HFFL, MODW, and TEF.)

(3) Inclusion of the high speed flow regime, wherein the effective wall temperature departs from the actual wall temperature. (See parameters TEF, DTEF, UVEL, IPUV, and MODW.)

(4) Addition of nonlinear tie conductances (see parameters UB, AHT, UEDT) for more stable and robust treatment of phenomena such as pool boiling and natural convection.

Many new heat transfer correlations have been added to support such situations as external flow, natural convection, jet impingement, pool boiling, enhanced heat transfer (ribs and protrusions), coiled tubes, rough walls, tube banks and pin fin heat sinks, etc. See Section 7.11.4.

Pressure Drop (momentum) Expansions—New path parameters and supporting correlations are available to assist in specialized modeling tasks. For noncircular flow passages with extreme aspect ratios, for which a hydraulic diameter assumption is invalid (especially for laminar flow), a new effective diameter DEFF is available. For curved or coiled tubes, for which additional pressure drops apply (and for which transition to turbulent can be delayed), the radius of curvature RCURV can be specified. Single-phase laminar and turbulent friction calculations can be customized using the FCLM and FCTM scaling factors. New parameters are also available to model axial and perpendicular forces in spinning and rotating fluid passages. Refer to Section 3.4 and Section 3.5 for details. Finally, a new correction for frictional effects in strongly heated or cooled passages is available (Section 7.11.5).

Co-solved Ordinary Differential Equations—A suite of routines is available to co-solve auxiliary (not implicit) ordinary differential equations, such as $F=m*a$ or $T=I*d^2\omega/dt^2$. This capability is almost as if a new type of mechanical network (“HEADER MECHANICAL DATA,” perhaps?) had been added, since it is expected to be most often used for describing motions of translational and rotational parts that influence a thermal/fluid solution. Refer to Section 7.9 for details and examples.

Parametric Sweep and Design Space Scanning—Utilities are now available to perform parametric sweeps of registers (PVSWEEP, Section 7.12.7), design variables (DVSWEEP, Section 5.15.1), and random variables (RVSWEEP, Section 5.26). More elaborate exploration of multiple design variables can be performed using either full factorial scans (DSCANFF, Section 5.15.3) or

the usually more efficient Latin hypercube sampling (DSCANLH, Section 5.15.2). DSCANLH is particularly useful prior to a call to the SOLVER, since it helps eliminate both initial starting point problems as well as scaling problems.

MISCELLANEOUS IMPROVEMENTS—

(1) Sink temperature (model reduction). Utilities have been provided to calculate and output sink temperatures. See Section 7.4.20 for more details.

(2) Surface recession simulation. A new simulation utility has been added for handling receding surface materials (melting, sublimation, ablation). Refer to the RECESS routine (Section 7.7.11) for more details.

(3) Immiscible liquids. Liquid species within mixtures can be specified to be wholly or partially immiscible with other species. This feature is useful not only for modeling polar/nonpolar mixtures, but also for certain classes of precipitation and solidification processes. See Section 3.21.5.3.

(4) Real gas mixtures. Mixtures may contain any number of real gases (6000 series fluids with the NEVERLIQ option). This enables both mixtures of real gases, plus a real gas mixed with a volatile fluid (e.g., high pressure pressurant gases with propellants or fire retardants). The limit of a single volatile/condensible substance remains.

(5) New spatial acceleration formulation. The calculation and application of the spatial acceleration term (AC) for tubes and STUBE connectors has been completely rewritten. Other than modestly increased stability, these changes are almost exclusively internal. Exceptions include the fact that special treatment is no longer needed for diffusers operating with nearly constant static pressures. Also, not all of the spatial acceleration term is contained within the AC term for paths within LINE and HX duct macros. In the latter case, the TB2TAB routine prints the entire equivalent pressure force associated with the spatial acceleration term.

(6) Faster matrix inversions. Improvements have been made in the execution of sparse matrix inversions, resulting in faster execution for all fluid submodels as well as any thermal submodels using nonzero MATMET. The user interface remains unchanged.

(7) Time step control. The lower limit for time step size during automatic control (nonzero DTIMEI in TRANSIENT) has been reduced from CSGMIN to 0.1*CSGMIN. This step might cause smaller time-steps, but helps prevent excessive error accumulation.

(8) Improved liquid mixture property estimations have been improved for conductivities, viscosities, and surface tensions. Refer to Section 3.19.1 for details.

(9) Two new routines are available for changing the results or OUTPUT file names during a run: CHGSAVE and CHGOUT (see Section 7.4.19).

(10) An array or part of an array can be filled with the same value or expression similar to the way the SPACE command fills N cells with zero values when “SPACE, N” is encountered within an array definition. To fill all or part of an array with identical values, use the syntax

"ALLSAME, N, expr" where the integer, real number, or expression *expr* will be repeated N times.

(11) Three new path output variables are available: REY (upstream Reynolds number, effective if two-phase), XMA (upstream* Mach number, effective if two-phase), MREG (an integer two-phase flow regime identifier): 0 = single-phase, 1 = bubbly, 2 = slug, 3 = annular, 4 = stratified. Note that all of these values *are for reference only*: the internal calculations make use of more complicated formulations including the differences between path inlets and outlets.

(12) A new top-level output routine, SIZETAB (no arguments), lists model sizes (number of nodes, lumps, etc.).

(13) A diffusion node can be made to temporarily act as an arithmetic node in transients using a call to ARITNOD (Section 7.6.21). While a diffusion node is in the arithmetic mode, its capacitance is nonzero and might continue to be updated but is treated as zero by program options (including time step control). ARITNOD may be used to convert a diffusion node to arithmetic if its capacitance has grown so small (via parametric changes, surface recession, etc.) such that it can be considered negligible.

(14) The underscore character ("_") may now be used within (but not as the leading character of) register names.

(15) ITERXT=0, 1, and 2 are now applied to matrix solutions as well to be consistent.

(16) The last argument for user calls to STDHTC (Section 7.11.3) has changed from the internal model sequence number to the fluid identifier, consistent with other correlation calls.

(17) BISTEL limits temperatures for property look-ups to the range -100°C to 60°C. Values outside of that range return the limits at -100°C or 60°C. This limit applies to TEC1 too if Ncp is nonzero.

(18) In fluid mixtures, the reference point for perfect gas enthalpies is no longer offset. This does not normally affect results, but will lower the lump HL values, which are printed by LMPTAB.

(19) An error in YTKONV has been corrected for diverging flows, and the routine is now compatible with use of duct macros.

SIGNIFICANT EXTERNAL PROGRAM IMPROVEMENTS—

(1) The capability has been added to execute and postprocess SINDA/FLUINT runs from Microsoft's Excel™, including indirectly: launching Thermal Desktop, which in turn launches SINDA/FLUINT using the Dynamic Mode. Example/template spreadsheets are included with the PC versions.

---

* Note that the upstream state does not include any restrictions or expansions. Use FR/FRC as a rough *estimate* of the Mach number at any throat or other expanded state (e.g., due to acceleration from stagnant inlets) within the path.

(2) FloCAD® has been significantly expanded to include 2D “pipe” methods for drawing complex surfaces (arbitrary cross sections swept along arbitrary centerlines) representing fluid ducts and heat pipes, and for attaching these to Thermal Desktop® surfaces and solids. These improvements make FLUINT modeling significantly easier for common pipe-flow and heatpipe problems.

VERSION 4.7 vs. VERSION 4.6

As an aid to users of Version 4.6, the major differences between Version 4.6 and Version 4.7 are summarized in this subsection. Previous models are almost always accepted with little difference in execution.

Simplified or Combined Fluid Properties—Two new utilities, PR8000 and PR9000, are available for creating simplified fluid descriptions: perfect gas (8000 series) or simple nonvolatile liquid (9000 series). These utilities may be used to simplify a more complicated fluid description, such as a full two-phase description (library, 6000, or 7000 series) or even a mixture. A model using a simplified description will often execute many times faster than the original.* Refer to Section 7.11.2.5 for details.

MISCELLANEOUS IMPROVEMENTS—

(1) New default correlations are used for critical heat flux (CHF) and post-CHF heat transfer, as well as subcooled boiling. These changes include a new meaning for CHFF if negative (Griffith correlation invoked), and a new output column for TIETAB.

(2) A 2D version of the TEC simulation utility, TEC2, has been introduced along with non-backward-compatible units conventions, new units options (TECUNITS), and "shallow solution" options for simulation of multi-stage devices. *Thermoelectric device users are urged to review Section 7.7.10 carefully.*†

(3) The PTHTAB "MACH" column has been changed to better distinguish itself from the path XMA value. XMA is an inlet Mach number useful for determining the importance of kinetic energy and elevated wall temperatures. The PTHTAB column, which is now |FR/FRC| as long as MCH is nonzero, is a more appropriate measurement of choking. In certain instances (e.g., a path with no constriction and a LSTAT=NORM static upstream lump), the two numbers will be the same.

(4) A new option to the RECESS routine allows the user to stop the surface recession at the last nodal layer for models that cannot tolerate complete melt-through.‡ Also, the restart and parametric options (SAVE, SVPART, etc.) now store nodal states, which facilitates parametric and sizing (optimization) runs which using RECESS. RECESS now also uses BDYNOD internally instead of HTRNOD such that HNQCAL calls, perhaps inserted automatically by Thermal Desktop, do not disrupt the Q of surface nodes.

(5) A new friction correction correlation (FCRAREF) is available for rarefied or mixed rarefied/continuum single-phase flows.

(6) A new utility routine, HTUDIF, is available for calculating diffusion-limited condensation for application to user ties (HTU, HTUS).

---

* These routines render obsolete the external program RAPPR, which is no longer available.
† Thermal Desktop V4.7 features an interface to the thermoelectric/Peltier device (TEC2) simulation.
‡ Thermal Desktop V4.7 features an interface to the surface recession simulation.

(7) A new utility routine, BALTPL, is available to help balance two-phase loops and other systems with fixed charge mass during steady states ("STEADY"). Example applications include loop heat pipes (LHPs), thermosyphons, detailed heat pipe models, vapor compression and other power cycles. (Section 7.11.9.8)

(8) A new output routine, SUBMAP, is available for submodel-level information about temperatures and heat flows.

(9) A new utility routine, CLRSAVE, is available to allow users to reset binary output files during a run.

(10) New simulation routines, COOLCON and PCOOLCON, are available for controlling coolers, whether thermostatic or proportional. HEATER and PHEATER, upon which the new routines are based, have also been expanded to be more compatible with cooler control.

(11) The actions of HTRLMP are released (as if RELLMP called) upon the start of a transient analysis, and therefore must be reinstated explicitly before any later STEADY calls, including those inside of repetitive procedures (PROCEDURE, RELPROCEDURE, etc.).

(12) Variants of HLDLMP/RELLMP are available that hold and release all tanks within a fluid submodel: HLDTANKS and RELTANKS. For example, "CALL HLDTANKS('pipette')" makes all tanks in submodel *pipette* act as temporary plena.

(13) FK is accepted as an *initial* value for UPRVLV and DPRVLV connectors. If it is missing, it defaults to a logarithmic average of FKL and FKH. (Previously, FK defaulted to 1.0. If FKL and FKH are defaulted, this new method corresponds to an initial FK of 1.0E4.)

(14) Utilities have been added to fetch the current value and derivatives for first and second order ordinary differential equations (ODEs). Four routines are available: one each for single and double precision, and one each for first and second order. These routines are named DIFFEQ1G, DIFFEQ2G, DIFFDP1G, and DIFFDP2G.

(15) Saturation pressures may be optionally added to 9000 series (simple nonvolatile liquid) descriptions. Warnings can be produced if fluid drops below the saturation pressure at other places in the network, indicating a two-phase model is needed, by calling the routine CHKFLASH. In future releases the net positive suction head (NPSH) at pump inlets will also be checked.

(16) Variants of HTRNOD and HTRMOD are available to change diffusion and arithmetic nodes into true boundary nodes instead of "heater nodes," whose Q is updated by the HNQCAL utility. The routines are BDYNOD and BDYMOD, which are used internally by the RECESS routine to preserve surface node's Q terms. The effects of these adjustments (as well as those of ARITNOD) are now preserved in restart files.

(17) The gravity level of natural convection correlation routines can now be adjusted, and a minimum temperature difference has been added for increased numerical stability.

(18) A method for predefining common file path names for INSERT files has been added that is useful for centrally located property files and for exchanging models between users. See Section 2.5.2.

VERSION 4.8 vs. VERSION 4.7

As an aid to users of Version 4.7, the major differences between Version 4.7 and Version 4.8 are summarized in this subsection. Previous models are accepted with little difference in execution.

PUMP Expansions—The PUMP module has been greatly expanded to include full map options (as well as optional use of head and flow coefficients), which permit the import of maps from pump design software. Other expansions include cavitation detection and modeling, viscosity corrections, and built-in calculation of heating rates and hydraulic torque. (See Section 3.5.15.) Note that the VPUMP option has been obsoleted.*

TABULAR Expansions—The TABULAR module has also been greatly expanded to assist in the modeling of a wider variety of components. These expansions include working with pressure ratios and corrected/equivalent mass flows to facilitate compressible flow modeling, including modeling gas labyrinth seals. Either static and total pressures can be specified as table inputs, along with flow area changes from inlet to outlet. Non-monotonic flows are now accepted, as are tables of flows versus heads and OFAC (in addition to the previously existing tables of heads versus flows and OFAC). Finally, options to accept torque and dissipative heating or shaft work have been added to assist with the application of TABULAR connectors either to dynamic seals or turbine/compressor primary flows.† (See Section 3.5.16.)

Rotating Path Options—These options have been expanded to include axial components of velocity, hydraulic torque and energy terms associated with rotation (including heating and shaft power effects appropriate for higher rotation speeds), and relative wall motions. The latter effect enables modeling of flow between a rotating wall and a stationary wall, permitting violation of the solid body rotation assumption that was previously implicit. For tubes and STUBE connectors, these new terms affect not only axial pressure force, but also perpendicular acceleration (which affects two-phase flow regime determination). If the walls move relative to one another, this also affects frictional pressure drop and heat transfer calculations (when the rotating tube or STUBE is associated with an HTN, HTNC, or HTNS tie). (See Section 3.26, which has combined the prior Section 3.4.2.)

Large Thermal Model Solution Improvements—New models tend to use more resolution than prior models for the same structure: mesh densities are increasing over time. Two improvements have therefore been made to specifically address such usage.

First, an energy balance check has been added to closure iterations for TRANSIENT. Without this check, if MATMET=0 then TRANSIENT can converge prematurely with only DRLXCA and ARLXCA as criteria for large models with strong G values (i.e., fine conductive meshes). This check is regulated by the new FBEBALA control constant (Section 4.4.1), and can be circumvented by specifying FBEBALA=0.0. The default value of FBEBALA=0.1 enforces only a modest 10% limit on energy imbalance, but can add significant CPU time to a solution. Nonzero MATMET or reduced resolution is the recommended solution.

---

* As part of these expansions, the PUMP option absorbed all the additional capabilities that the VPUMP option originally provided. Therefore, the VPUMP option has been obsoleted. It is available as an undocumented feature for backward compatibility. Simply change “VPUMP” to “PUMP to be compatible with current usage.

† Future versions will include separate turbine and compressor modules to supplement and/or replace the use of TABULAR connectors for modeling primary flow paths in turbomachines or stages of turbomachines.

Second, an algorithm has been implemented to sparsify thermal solution matrices, resulting in reductions of memory and increased solution speed for large models, especially if radiation is present. This algorithm is regulated by the new SPARSEG control constant (Section 4.4.1), and can be circumvented by specifying SPARSEG=0.0. The primary purpose of the SPARSEG algorithm is to enable nonzero MATMET to be applied to larger models without requiring excessive memory, and thereby avoiding energy imbalance issues such as FBEBALA addresses for iterative solutions.

MISCELLANEOUS IMPROVEMENTS—

(1) A new Solver diagnostic utility, SOLCHECK (Section 7.13.4), is now available. SOLCHECK can be used to detect weak functions or absent connectivity. It provides a "Solver-eye view" of the optimization problem.

(2) A new utility routine is available for calculating the standard atmospheric temperature and pressure: STDATMOS (7.7.13).

(3) A new utility routine, TOTALTP, is available for calculating total (stagnation) temperature and pressure at a fluid lump (Section 7.11.1.10), given a reference path for velocity head.

(4) A new tabulation-style output routine, CHOKETAB, has been added to provide more detailed information on both choking and any isentropic expansions and throat states that might

(5) Improvements have been made to the handling of thermodynamic state initializations using expressions.

(6) Conflicts with RESPAR and RESTAR (if SAVPAR were used or if all data were saved) and Solver, reliability engineering, and parametric sweep options have been eliminated. It is no longer necessary to use the '-R' option to avoid saving registers when performing sweeps or using iterative procedures that involve transients. These routines (SOLVER, SOLCHECK, SAMPLE, DSAMPLE, RELEST, PSWEEP, DVSWEEP, RVSWEEP) make sure that the key variables are restored and that their current values are propagated after a RESTAR or RESPAR call has been made.

(7) Diffusion-limited heat and mass transfer is now automatically incorporated for twinned tanks employing both a condensible/volatile species and one or more noncondensible species (Section 3.25.2. and HTUDIF in Section 7.11.4).

(8) For twinned tanks in duct macros, or where twinned HTN/HTNC ties are also applied to those twinned tanks by the user, then the heat transfer from wall to liquid had previously used FQ=1: all of the twinned tie was apportioned to the liquid side. This treatment tended to underestimate nonboiling heat transfer for the bubbly, slug, and annular regimes. In untwinned tanks, the heat transfer was essentially related to the resistance of the liquid layer: the temperature drop from the wall to interface. With twinned tanks, that resistance should instead be approximately halved since the temperature drop should instead be calculated from the wall to the *middle* of the liquid; a separate resistance (per CULT) takes into account liquid-to-interface temperature drops. Therefore, when ITAC=3 (the default), primary (liq-

uid) ties might now have an FQ of up to 2.0 to better account for this effect, and to better match the overall heat transfer rates of homogeneous tanks at steady states.

(9) For 6000 series fluids, if $P>P_{crit}$ but $T<T_{crit}$ ("cold supercritical" regime), uncertainty exists as to whether the fluid should be compressible (XL=1.0) or incompressible (XL=0.0). The rules governing this uncertainty have been clarified (see Figure 3-41) and internal improvements have been made to better deal with this ambiguity. (This applies only to non-COMPLIQ fluids.)

(10) The default minimum temperature for user-defined fluids, TMIN, has been raised from 10^{-3} to 1 degree (absolute local units) to avoid conflicts between absolute zero values (e.g., "-460.0" in FPROP blocks vs. "-459.67" in GLOBAL DATA).

(11) Range limits have been relaxed for liquids within mixtures, with the rules explained in Section C.4.

(12) A new manual section summarizing output operations and options has been provided (Section 4.6).

(13) Two new natural convection correlations have been provided for heat transfer between concentric spheres and cylinders (Section 7.11.4.8).

(14) New heat transfer correlations are available for flows between stationary and rotating disks (Section 7.11.4.9).

VERSION 5.0 vs. VERSION 4.8

As an aid to users of Version 4.8, the major differences between Version 4.8 and Version 5.0 are summarized in this subsection. Previous models are accepted with little difference in execution.

Turbine Device—A new connector device, the TURBINE, is available for modeling turbines using either performance maps or equations. Power, torque, and efficiency calculations can be modeled, as well as sonic-limited flows. User logic, the Solver, or co-solved ODEs can be used to either balance torques or calculate dynamic shaft speeds for turbopump systems, or for turbines and compressors mounted on the same shaft. (Section 3.5.17)

Compressor Devices—Two new connector devices are available to simulate either variable (COMPRESS, Section 3.5.18) or positive displacement (COMPPD, Section 3.5.19) compressors. Surging and choking may be modeled, along with calculations of power, torque, and mechanical efficiency.

Thermodynamically Compressible Liquids—In all prior versions, liquids (XL=0.0) were assumed to be thermodynamically incompressible, although a tank wall compliance could be used to simulate compressibility for waterhammer-type analyses. Nonetheless, the lack of a fully compressible liquid option resulted in regions of ambiguity (overlap of two equations of state) for cold, high pressure fluids and created modeling problems for systems employing cryogens, or for systems such as Joule-Thomson expanders or transcritical carbon dioxide heat pump and refrigeration cycles that traversed this problem zone. A new COMPLIQ option has been added to 6000 series FPROP blocks that eliminates such problems, and compatible full-range fluids descriptions are now available. (Section 3.21.7)

MISCELLANEOUS IMPROVEMENTS—

(1) For rotating passages with walls that do not co-rotate (RVR<1), new routines are available for calculating torque in disks and cylinders for use with the path INTORQ=YES option (Section 7.11.7).

(2) A new interpolation routine, D1D1DAH, permits interpolations to use two arrays with hysteresis: one for increases in the independent variable, and another for decreases (Section 7.3.3).

(3) Various minor improvements in text output options have been made, including thermal submodel convergence status, and energy balance information (e.g., EBALSC) calculated and output for thermal submodels even if not converged. For fluid submodels, path tabulators TUBTAB and TB2TAB now calculate ACeff ... the effective AC factor instead of the actual (which may be zero). Finally, using a new routine CSTNAMES, unnamed Solver constraints can be listed in Solver output routine CSTTAB to distinguish or identify them. (Section 7.13.2)

(4) Named and unnamed Solver constraints can be temporarily disabled using a new utility, OFFCST. This option is useful for design space prescanning using DSCANLH, which otherwise is limited in the starting point returned if the design space is excessively constrained. (Section 7.13.2)

(5) New PID controller options have been added to permit advanced initializations (PIDINT) and to limit the wind-up (PIDSETW). (Section 7.7.8)

(6) New map-style output routines are available to provide detailed performance information and input echoing for the new turbomachine devices: PUMPMAP, TURBMAP, COMPRMAP, and COMPPDMAP (Section 7.11.8).

(7) PSWEEP has been modified to permit reverse parametric sweeps from larger to smaller values (Section 7.12.7).

(8) The default MCH for all turbomachine connector devices is now zero: choking is presumed to be contained within the performance maps.

VERSION 5.1 vs. VERSION 5.0

As an aid to users of Version 5.0, the major differences between Version 5.1 and Version 5.0 are summarized in this subsection. Previous models are accepted with little difference in execution.

Variable Molecular Weight Fluid Species—6000 series FPROP DATA blocks now support an optional feature: a molecular weight that varies as a function of pressure and temperature. This option facilitates the modeling of complex mixtures, perhaps internally reacting or dissociating/recombining, such as ionized gases and hot products of combustion. It may be used alone or in combination with the chemical reactions described next. (Section 3.21.7.5)

Chemical Reactions—The ability to model finite rate ("reacting flow") chemical reactions has been added. This capability takes in the form of species mass sink/source terms for FLUINT tanks within transients (or within STDSTL, but not STEADY). The corresponding heat of reaction is applied on the basis of heat of formation added to the species fluid description (FPROP DATA block). A reaction rate utility routine (EQRATE) is available to help define chemical equations, and perhaps to drive the reaction to a desired state (as defined by either equilibrium fractions, or by reaction/combustion efficiencies). One of the product species might be a variable molecular weight fluid, eliminating the need to track many products of combustion by using a single equivalent species instead. (Section 3.21.2.5, Section 3.24)

MISCELLANEOUS IMPROVEMENTS—

(1) A new routine is available (TANKTRHO) to change tank (and only tank) temperatures instantaneously, holding density constant. It may also be used to specify density (and therefore mass), perhaps as an initial condition holding temperature constant (or alternatively, specifying a new temperature at the same time). (Section 7.11.1.1)

(2) New PUMP options exist for ISTP: 'TS' and 'ST' to match the options available in TABULAR, TURBINE, COMPRESS, etc.

(3) The output routines TURBMAP and COMPRMAP now print the unitless modifying factors δ, θ, and ε if MREF=3 and NSPD=1.

(4) GFR, the flow rate (corresponding to FR but in G_{table} units) at the start of the last interval, is available as an output variable for TABULAR, PUMP, TURBINE, COMPRESS (but not COMPPD).GFR is also printed in the corresponding tabulation-style output routines for each device type.

(5) Execution of large FLUINT models (either due to number of lumps or number of constituents) has been accelerated by improved handling of internal data. No changes to the user interface have been made: this change is internal.

(6) A new variation of RECESS has been created to handle rate-of-recession as an input. See RECESS_RATE in Section 7.7.11 for more information.

VERSION 5.2 vs. VERSION 5.1

As an aid to users of Version 5.1, the major differences between Version 5.2 and Version 5.1 are summarized in this subsection. Previous models are accepted with little difference in execution.

New Default Thermal Solution Method—The prior default solution method, MATMET=0, was purely iterative: each node was solved individually (as if neighboring nodes were temporarily acting as boundaries), repeating a cycle through all nodes until they were converged (steady state solution, or transient time step). This method has the advantages of being tolerant of network construction errors and incautious user logic. However, it had the disadvantage of being very slow for huge models, with potentially stalled solutions (see for example, FBEBALA). The alternative was to use the Yale Sparse Matrix Package (YSMP) for simultaneous solutions applied to one (MATMET=1) or more (MATMET=2) thermal submodels at a time.

As models grew in size, especially with the increased reliance on FEA solutions, a new method was required to reduce memory requirements and to improve scaling (the fractional increase in solution cost for a fractional increase in model size). The result is a new AMG-CG (algebraic multigrid method with conjugate gradient acceleration). This method (Appendix F) is usually substantially faster than YSMP and almost always uses less memory. In fact, it scales so well that there is rarely any benefit to solving submodel by submodel (MATMET=11). Instead, a much better approach is to use MATMET=12: all submodels designated with MATMET=12 will be solved simultaneously as a single matrix. This approach is now the default method, eliminating concerns about stalled and slowly executing large models. However, a simultaneous solution is also less tolerant of mistakes in network construction and logic: MATMET=0 is still available as a debugging mode.

Because AMG-CG's performance (both speed and memory) is much less sensitive to matrix density than YSMP's performance, the default for SPARSEG has been reduced from 0.01 to 0.001 now that MATMET=12 is the default thermal solution setting.

Flat Front Two-Phase Modeling—Section 3.27 documents a new method for modeling the progression of a liquid or vapor front through a piping network. When modeling the filling of empty lines with liquid, or when modeling the purging of liquid-filled lines with a high pressure gas, it is more convenient and perhaps conservative to represent two-phase flow using an "axially stratified flow regime." The interface between liquid and vapor is assumed to be a well-delimited plane perpendicular to the flow direction.

MISCELLANEOUS IMPROVEMENTS—

(1) A utility has been added to assist in modeling ice or frost accretion and melting (ACCRETE, Section 7.7.12). ACCRETE has many parallels and shared utilities with RECESS.

(2) System-level (effectiveness/NTU) utilities are available for sizing or simulating heat exchangers. "System-level" means treating the hot and cold flow streams as single FLUINT paths between two lumps representing the inlets and outlets. Section 7.11.10 details routines for calculating effectiveness, NTU or UA_{tot}, and the heat capacities C_{min} and C_{max} based on current conditions in a FLUINT path. These values may be used in a central utility, HXMASTER (Section 7.11.10.3), which may be used to either size or simulate a heat

exchanger based on whether effectiveness, transferred power, or an outlet temperature is either known or desired.

(3) Methods for incorporating compact heat exchanger (CHX) style modeling methods of heat exchangers have been developed, and are detailed in Section 7.11.11. This approach takes advantage of a concurrent development in Sinaps and FloCAD: “network logic.”

VERSION 5.3 vs. VERSION 5.2

As an aid to users of Version 5.2, the major differences between Version 5.3 and Version 5.2 are summarized in this subsection. Previous models are often accepted without modification, and with little difference in execution. (See “Argument Checking” below: modifications may be required for older models to compile successfully.)

Some of the changes affect predictions even if no new options are selected. “Extended Duct Macros” may cause differences in results as new terms are considered in Sinaps 5.3 and FloCAD 5.3. ORIFICE elements may predict slightly lower gas flow rates due to consideration of an alternative compressible flow correction. Also, choking calculations involving phase change now automatically check for spinodal (phase decomposition) limits instead of leaving it to the user to check for these, and this new methodology can result in significant differences in results in some models.

Expanded Names and Line Lengths—Submodel and register names may now be up to 32 characters long. Data (but not logic*) input lines may now be up to 1000 characters long, which means file names, register expressions, etc. may now be longer as well. Element IDs may be up to 8 digits long.

Double-precision Registers—Registers may now be real (single precision), integer, or double precision (REAL*8, or 8-bytes). New temporary variables (ATEST_DP through HTEST_DP, and OTEST_DP through ZTEST_DP) are also available in double precision, as well as INTEGER*8 integers (ITEST_DP through NTEST_DP). See Section 2.8.3.

Routine Argument Checking—A source of difficult-to-trap errors has been eliminated: the wrong number of type of argument in subroutines and functions in user logic. However, this means some “laziness” in prior input files (e.g., the use of “1” when “1.0” was expected) is no longer tolerated. It also means that new functionality such as optional or alternative arguments is now possible. See Section 4.1.4.2, Section 4.1.4.6, and Section 4.1.4.7.

Global-level Logic and Output Blocks—FLOGIC, VARIABLES, and OUTPUT CALLS blocks may be created which operate at the “global” (submodel-independent) level, and are therefore independent of BUILD or BUILDF configurations.

Extended Duct Macros—Sinaps and FloCAD auto-detect flow-wise sequences of lumps and paths (tubes or STUBE connectors specifically) can have continuous flow area and so which should be automatically aggregated into an equivalent LINE or HX macro. Sinaps and FloCAD “Pipes” are always considered to be duct macros as well. In Version 5.3, these tools will now combine mixed types of elements (tank vs. junction, tube vs. STUBE, homogeneous vs. twinned path or lump) into a macrocommand: something not possible in text-based input. In fact, Sinaps and FloCAD will also combine one or more Pipes themselves into a larger macro if possible. See Section 3.9.2.2.

This step has been taken in part to allow the user to break a single Pipe into multiple Pipes (perhaps due to an insulation change, etc.) without losing the axial acceleration terms at the ends of each macro, which can be important in two-phase or compressible flows. See also MACINTAB and MACROTAB (Section 3.9.2.1), new output routines to help slow macro-level inputs and responses.

---

* Fortran compilers limit fixed format to 132 columns.

MISCELLANEOUS IMPROVEMENTS—

(1) Root and minimum finders have been added to assist users in co-solving auxiliary equations during steady-state solutions. One example of such an task is finding the rotational speed of a shaft interconnecting a turbine and compressor such that net torque is zero. These routines (MIN_FIND, ROOT_FIND) may be viewed as alternatives or adjuncts to the Solver module. See Section 7.10.

(2) A new film coefficient correlation for mixed forced/natural convection, NCCYLIN, is available. This routine is useful for estimating the heat transfer between fluid and wall when filling or emptying tanks (especially cylindrical ones). See Section 7.11.4.9.

(3) A variation of CHGLMP is available to assist in the simultaneous resetting of multiple species fractions: CHGLMP_MIX. See Section 7.11.1.2.

(4) A new utility, SETTMOD, allows a user to set some or all nodes in a submodel to a new temperature. See Section 7.5.1.

(5) SINGLEPR (SINGLEPRECISION) is now ignored in OPTIONS. See Section 2.7.3.

(6) New duct macro tabulation-style output routines are available: MACINTAB (for echoing inputs) and MACROTAB (for outputs). See Section 3.9.2.1 and Section 7.11.8.

(7) During film or transition boiling in twinned HTN/HTNC ties to twinned tanks, the vapor twin tie is no longer disabled via the FQ factor: heat is added directly to the superheated film.

(8) The current unit identifier, UID, is available as an integer in logic or expressions (UID=0 for ‘ENG’ and UID=1 for ‘SI’) such that the user can write model-independent logic and expressions.

(9) The MIXARRAY check has been expanded to auto-detect and accept bivariate arrays in order to encourage the use of this check.

(10) A new check has been added to auto-detect perpetual oscillations in nodal temperatures in thermal steady-state solutions, automatically invoking heavy damping (ITERXT=0, EXTLIM=0.5) if detected. A warning message will appear if this emergency measure is taken.

(11) Decomposition (spinodal) limits have been added to metastable (nonequilibrium) choking calculations for flow in entrances and contractions to throats. These limits often cause attempts to find metastable throat states to be abandoned in favor of equilibrium states, as if positive MCH had been input instead of negative MCH. See Section 3.18.1.5.

(12) A direct link to NIST’s REFPROP is available for alternative (high fidelity) fluid property descriptions. See Section 3.21.7.9.

(13) For all-gas flows through orifices, an additional check is made against the ASME Y expansion factor to make sure the predicted flows do not exceed the predictions of that correlation. See Section 3.5.12. This extra check may result in slightly lower flow rates in some models.

VERSION 5.4 vs. VERSION 5.3

As an aid to users of Version 5.3, the major differences between Version 5.4 and Version 5.3 are summarized in this subsection. Previous models are normally accepted without modification, and with little difference in execution.

Full Double Precision— Modern CPUs and operating systems no longer penalize the use of double precision (8 byte) real variables over single precision (4 byte) variables. While key parts of the solution have always been double precision, the size of all real (and integer) variables has been extended to exploit the availability of greater accuracy. No changes to models is required, and results are often very similar. (Future version will revise numerical schemes to exploit the availability of greater accuracy.) Double precision registers and double precision variations of various routines (e.g., DIFFDP1) are tolerated but no longer necessary since they are now identical to their single precision analogs (e.g., DIFFEQ1).

"Pool Boiling" (HTP) Ties—In many situations, such as on the insides of vessel walls, convection coefficients can be calculated even though the fluid is quasi-stagnant and forced convection is not appropriate. For those situations, and to make it easier to model pool boiling, a new type of tie has been created. See Section 3.6.3.3 for details.

Controllable Orifices—Orifices can often be used as models of partially opened valves. This fact has been exploited in extensions of the ORIFICE connector device to create orifices that can be commanded to open or close, or to maintain a constant upstream or downstream pressure. These options add more realistic transient behavior for modeling valve events and regulators. See Section 3.5.12.2 for details.

MISCELLANEOUS IMPROVEMENTS—

(1) Due to limitations in the built-in water properties, FID=718 (simple water) is no longer available as a working fluid, and is therefore no longer the default. If no FID has been specified for a fluid submodel, the preprocessor will issue a fatal error. If 718 has been used as a working fluid or species, the preprocessor will issue a fatal error as well.

For two-phase water properties, the user must instead use an INSERT file. These four F-files are now included in the installation (in addition to R-files):

f6070_water.inc. . . . . two-phase water, incompressible liquid
f6070_water_big.inc . . . . . same as above, twice the resolution
f6070CL_water.inc . . . . . two-phase water, compressible liquid
f6070CL_water_big.inc . . . . . same as above, twice the resolution

(2) For FPROP blocks in which WSRK (the modified SRK acentric factor used in liquid mixture density calculations) is a valid input, the actual (true) acentric factor is also now available: WTRUE. This latter value is used for Peng-Robinson spinodal estimations for two-phase choking calculations.

(3) In the POOLBOIL routine, the QMIN and QCHF terms apply the natural convection gravity level (perhaps as altered in a call to NCSET). For an HTP tie, this gravity term is based on the ACCM input and is independent of any NCSET calls.

(4) When using fluid mixtures, the lower pressure limit (PMIN) is now set according to the contents of each lump. This means that if there are two species, one with a high PMIN and one with a low PMIN, then the lump is allowed to go to the lower pressure value providing it only contains the low-PMIN fluid.

(5) A new twinned tank initialization utility, SPLIT_TWIN, is available for use within logic blocks. Whereas GOTWIN enables splitting of twins in the next solution step, SPLIT_TWIN causes the split immediately such that each tank can then be adjusted. This option is primarily used for initializations, and eliminates the prior need to specify NSOL=2 before the BUILDF statement.

(6) Memory requirements are now allocated as needed for handling SINDA conductors.

VERSION 5.5 vs. VERSION 5.4

As an aid to users of Version 5.4, the major differences between Version 5.5 and Version 5.4 are summarized in this subsection. Previous models are normally accepted without modification, and with little difference in execution.

Void Fraction as an Input— The initial void fraction (AL) may now be specified as an input parameter for lumps. If AL is specified, the initial XL is ignored. For twinned tanks, specifying AL corresponds to setting the initial volume of each tank: AL*VOL for the secondary/vapor tank, and (1.0-AL)*VOL for the primary/liquid tank.

Twinned Tank Vapor Temperature Initialization—The secondary (vapor/ullage) side of a twinned tank may now be initialized with a temperature distinct from that of the liquid using the TLV ("TL of Vapor") option. This option also signals the twins to start in the split state, even if the phasic temperatures are the same initially (TL=TLV).

Internal Updates—Internal modules for the thermal AMG-CG simultaneous solution and for the Solver NLP module have been updated. Although execution might be slightly different (hopefully faster and more robust), the main reason for these updates is to maintain the ability to support users should issues arise.

SIGNIFICANT EXTERNAL PROGRAM IMPROVEMENTS—The Thermal Desktop tool suite has received an additional boost from new tools that simplify the preparation and maintenance of thermal models that originate from CAD geometry. These tools include CRTech TD Direct™, an expansion of SpaceClaim®,* which allows import, manipulation, healing and simplification of virtually any CAD part or assembly. Version 5.5 marks a significant new module an advanced mesh generator, which allows users to mark-up geometry with properties, boundary conditions, and so forth such that changes update automatically (including re-meshing) to Thermal Desktop.

MISCELLANEOUS IMPROVEMENTS—

(1) The coefficient of thermal expansion is now available from the NCPROP natural convection utility.

(2) New phase-specific average temperature and density options are available for SUMFLO and SUMDFLO.

* SpaceClaim is a registered trademark of SpaceClaim Corporation.

VERSION 5.6 vs. VERSION 5.5

As an aid to users of Version 5.5, the major differences between Version 5.6 and Version 5.5 are summarized in this subsection. Previous models are normally accepted without modification, and with little difference in execution.

Most of the changes in Version 5.6 are internal: improved speed and accuracy, reduced memory, etc. Most of these changes are related to two-phase flow in general, and to twinned tanks (phasic nonequilibrium options) specifically.

Expanded Twinned Tank "Wrong Phase" (VDRP/VBUB) Options—When liquid appears in the secondary/gas/vapor side of a twinned tank, it is returned gradually to the liquid side as droplets at a velocity VDRP. VBUB similarly prescribes the bubble rise rate of vapor leaving the primary/liquid side and returning to the vapor/gas twin. New meanings of VDRP and VBUB are now available that better handle cases with copious boiling or condensation, or with bubble injection on the liquid side or droplet spraying on the vapor/gas side.

Flat Front Twinned Tank Transitions—Within a duct macro that uses FF=FILL or FF=PURGE and twinned tanks, a significant change has been made in the handling the movement of the liquid/vapor interface. Instead of collapsing one pair of twinned tanks as the void fraction hits the upper limit (for purging) or lower limit (for filling), and then waiting for the next tank downstream to form enough of the entering phase to split, the front is now "passed" downstream. This not only diminishes the discontinuities, it preserves the temperatures on each side of the liquid/vapor interface.

Compressed Solution Results (CSR folders)—An alternative to the binary SAVE *file* is now available: the *CSR folder*. A CSR folder is a compressed folder containing numerous but simple files, one for each parameter (e.g., T, FR, PL, etc.) saved. Each such file is a simple list of saved values (though these values are still in binary format for compression and accuracy). EZXY, Sinaps, and Thermal Desktop all can read either a traditional SAVE file or a CSR folder format for postprocessing, restarting, etc.

The new format attempts to overcome several limitations of the older single-file format, which was difficult for a third party (including an advanced user) to read.* Traditional SAVE files were also difficult to expand: using the CSR format will allow users to add their own variables to the postprocessing set in future versions. The only disadvantage of the CSR format is a slower speed when it used with small models, especially when repeating a run that forces old files to be deleted and re-created. Large models, on the other hand, often execute faster than before.

CSR folders are created by the same routines that create the binary SAVE file format: SAVE, RESAVE, SAVEDB, etc. Therefore, an "RSO file" becomes an "RSO folder." References in the manual to a "SAVE file" may be interpreted as also applying to a "SAVE folder" (a CSR folder created by the SAVE family of routines) and vice versa.

* Documentation of the new format and examples of how to access it are available from CRTech.

Traditional SAVE files will eventually be obsoleted. The CSR format will be the default for all models soon, though the traditional single-file format will still be available as an option. However, the format of traditional SAVE files is now frozen, meaning that no new variables added in future versions will be accessible in that output format. This restriction will prevent the use of the old format for complete restarting; RESTAR will soon operate only with CSR folders.

CSR folders will soon be the default for new models created in either Sinaps or Thermal Desktop. Prior models will continue to use the SAVE file, but should be converted by the user as soon as possible to the new format.

For users not using the GUIs, the choice of file vs. CSR folder format can be invoked (once per run in OPERATIONS) using the utilities USESAVEFILE and USECSR. The default is to use the traditional SAVE file to allow backward compatibility of existing text input files, but the insertion of a USECSR is strongly encouraged.

MISCELLANEOUS IMPROVEMENTS—

(1) TIE2P is a user-callable routine to extract the boiling regime information available in the “2P” column of the TIETAB routine.

(2) Two new timer routines, SINDA_CLOCK and SINDA_CPU, are available for fetching clock and CPU time for the current processor execution in user logic. (Section 7.6.22)

(3) The internal version of NIST’s REFPROP (used in R-files) has been updated to version 9. New F-files are available as well.

(4) New indirect operations have been added to expressions and Network Logic (a feature of Sinaps and Thermal Desktop) to refer to the secondary (twin) lump or path of a tie, path, or FTIE. For example, the #LUMPTWIN of a tie refers to the secondary tank of a twinned pair to which it is connected.

(5) Extensions of ARITNOD are now available to convert more diffusion nodes into “arithmetic” status as a function of either submodel (ARITMOD) or time scale (ARITCSG). ARITMOD is convenient for re-purposing a submodel intended for small time scales to a case focused on larger time scales. ARITCSG can also serve this purpose, and it can also be used for models with parametric adjustments in dimensions, or with fluid ties or radiation conductors that change significantly. (Section 7.6.21)

(6) A new utility, TANK_LINKER (Section 7.11.9.3) allows an IFACE-like connection to be made between two tanks, even if they are not in the same fluid submodel.

(7) Slip flow (ducts with twinned paths) has been reworked with improved algorithms and averaging schemes to yield smoother and more stable responses. No change to the equations or to the user interface have been made, yet predictions can sometimes differ significantly. Often this is a result of a change in flow regime, but these differences ultimately underscore the uncertainty in such estimations.

(8) The use of flat-front (fill/purge) duct modeling has been extended to better handle volatile liquids, including low levels of boiling or flashing in the liquid column, and condensation in the vapor column. See DUCT_MODE (Section 7.11.1.4).

VERSION 5.7 vs VERSION 5.6

As an aid to users of Version 5.6, the major differences between Version 5.7 and Version 5.6 are summarized in this subsection. Previous models are normally accepted without modification, and with little difference in execution.

Most of the changes in Version 5.7 are internal, representing infrastructural improvements intended to enable future expansions. These improvements include restructuring of memory to be almost entirely dynamically allocated.

MISCELLANEOUS IMPROVEMENTS—

(1) New correlations are available for detailed modeling of offset strip fin heat exchangers (see Section 7.11.11.4).

(2) Two new input factors for HTN/C/S ties, RLAM and RTURB, allow the user to specify the Reynolds number transition limits for single-phase heat transfer. See Section 3.6.6.

(3) Another new tie factor is for output postprocessing purposes: NTWOP, an integer flag denoting the current state of two-phase heat transfer (e.g., transition vs. film boiling). See Section 3.6.7.3.

(4) New options in the SUMFLO and SUMDFLO utilities allow the summation of the total energy and mass error rates in the submodel.

(5) When working fluids are combined into a mixture, the smallest valid range of temperatures and pressures must be satisfied when all species are present.

Previously, this limited range was enforced at the submodel level, with certain exceptions applied for local concentrations in any one lump. These exceptions have been expanded such that virtually all enforcement of range limits is now based on local lump states. In other words, if a constituent is not present in a lump, the range limits of that substance are also ignored in that lump. For example, a hot gas can be used at a temperature above the upper limits of other species, so long as those lower-temperature species are not present in the hot regions.

There must exist *some* common temperature/pressure range at which all species can coexist, even if they never actually mix or even if one species is not currently used anywhere within the submodel.

As usual, it is dangerous to introduce a species into a lump whose current temperature or pressure is outside of the newly introduced species' range limits. See Section C.4 for more information.

(6) Flow paths that connect to a large two-phase vessel require additional analytic treatment depending on whether Path inlet/outlet locations ("ports") are above or the liquid surface. The coordinate location of each end of the path (CXI, CYI, CZI and CXJ, CYJ, CZJ) may now be defined independently of the endpoint lump locations (CX, CY, CZ for each lump). This invokes simulation logic including adjustment of phase suction options, twinned tank attachment, body force terms, and choking calculations. See Section 3.5 and especially Section 3.13.4.

(7) An import utility, TankCalc60, has been created for Version 6.0 of the free program TankCalc. TankCalc is used to calculate liquid level as a function of volume in common shapes of two-phase vessels, along with wetted and surface areas. TankCalc60 uses the outputs of TankCalc to update various simulation values such as the elevation of the liquid/vapor surface, the twinned tank liquid/vapor surface area, and twinned tie characteristic dimensions per phase.

(8) The program has been updated to REFPROP 9.1. This affects property R-files. .A new set of corresponding F-files is also available at www.crtech.com

(9) A new utility is available to print the causes of the most severe time step limits in a prior transient run. See SETTL and PRINTTL in Section 7.6.23.

VERSION 5.8 vs VERSION 5.7

As an aid to users of Version 5.7, the major differences between Version 5.8 and Version 5.7 are summarized in this subsection. Previous models are normally accepted without modification, and with little difference in execution.

Almost all of the changes in Version 5.8 are internal, and most of those changes support the new Compartment feature in FloCAD Version 5.8. Refer to the Thermal Desktop User's Manual for more details on Compartments.

MISCELLANEOUS IMPROVEMENTS—

(1) New keywords are available to customize the operation of twinned HTP and HTU ties. For example, vapor/gas (secondary) tie AHT values for ITAC=1 ties can be input separately as "AHT_V." These new "_V" keywords are not SINDA/FLUINT variables: they are not translatable in Fortran logic,* and are not listed in postprocessing trees. Rather, they provide a means of setting (via expression) the vapor-side values independent of liquid-side (primary) values within FLOW DATA. See Section 3.6.3.1 and Section 3.6.3.3 for more details.

These new keywords are intended to be used in conjunction with FloCAD Compartment ties, but they may be used independently as well.

(2) ITAC=1 HTU and HTP twinned ties now continue to both be active (maintained as independent heat transfer routes) even if the attached lump is single-phase or is twinned and currently in the mixed mode. This usage allows independent expressions and logic to be developed for each phase (e.g., UB or DCH for liquid, versus UB or DCH for vapor/gas).

(3) New path output routines, DPTAB and DP2TAB, present flow forces converted into units of pressure difference for ease of comparison. See Section 7 for more details.

(4) Steady-state convergence and transient stability have been improved for pressure-controlling (MODA=-1 or -2) ORIFICE devices. Steady-state convergence now checks for AORI changes, but these changes are now more intelligently damped for faster convergence.

(5) The SINDAWIN run-time utility for PCs has been expanded to include progress plotting and register resetting during the run. See Appendix G.

(6) Ports (path end locations), introduced in Version 5.7, now work as side connections to flat-front pipes.

(7) Reducers and Expanders have been added as connector devices. These can represent sudden expansions and contractions, diffusers, nozzles, and (when used in combination) venturi tubes. See Section 3.5.3 and Section 3.5.4.

---

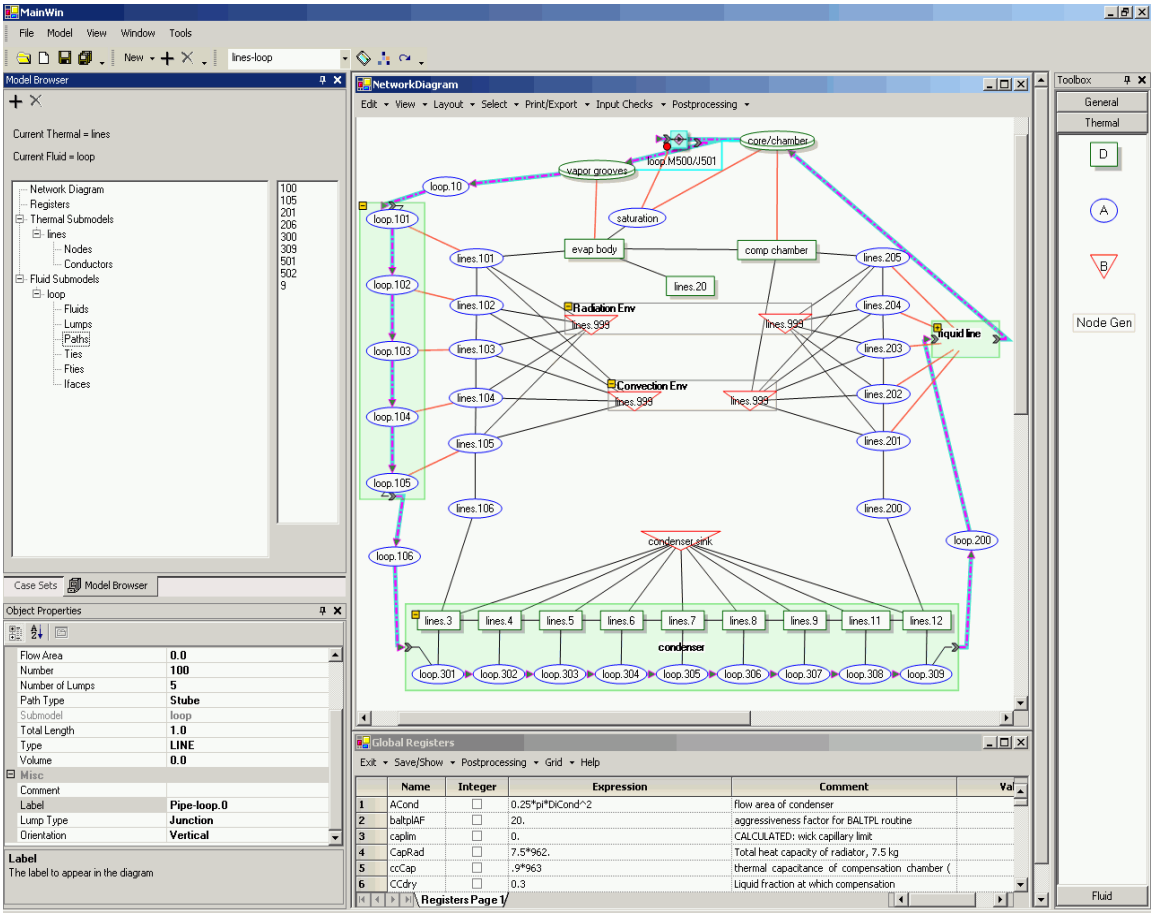
* FloCAD Network Element Logic (NEL) can be used to address each variable uniquely, such as "AHT#this" and "AHT#twin"

(8) A new two-phase nonequilibrium expansion/choking option (MCH=-3) has been added to better support liquids that flash in a nozzle or orifice throat. See Section 3.18. MCH=-3 is now the new default for ORIFICE devices. It is also defaulted for the new REDUCER and EXPANDER devices.

(9) Updates for HXMASTER have been made so ties and paths on outlet lumps are no longer disrupted using MODE_S=1 (the setting used in FloCAD). Similarly, the QL of the exit lumps are left alone and CONSTQ FTIEs are no longer needed or recommended to connect the exit lumps of a single heat exchanger. Temperatures of plena and HLDLMP tanks/junctions are also no longer changed automatically. See Section 7.11.10.3.

Sinaps®: GRAPHICAL (NONGEOMETRIC) MODEL DEVELOPMENT

Sinaps* is a companion program to SINDA/FLUINT that enables users to graphically sketch their models using a mouse- and menu-driven user interface. Forms and editing windows exist to satisfy other nongraphical input requirements. Sinaps then produces complete SINDA/FLUINT input files, or can be used to launch a run. It can also be used to read binary output files. This enables *graphical display of predictions on the same schematic used to create inputs.* In addition to pop-up X-Y, polar, and bar plots, features such as "color by flow rate," "thicken by conductance," and "shade by temperature" are supported. A sample Sinaps screen image is provided below:



Sinaps is a complete model management tool that eliminates the need to write ASCII inputs and interpret results via ASCII files.

Sinaps is documented in a separate User's Manual. Sinaps® is a registered trademark of C&R Technologies.

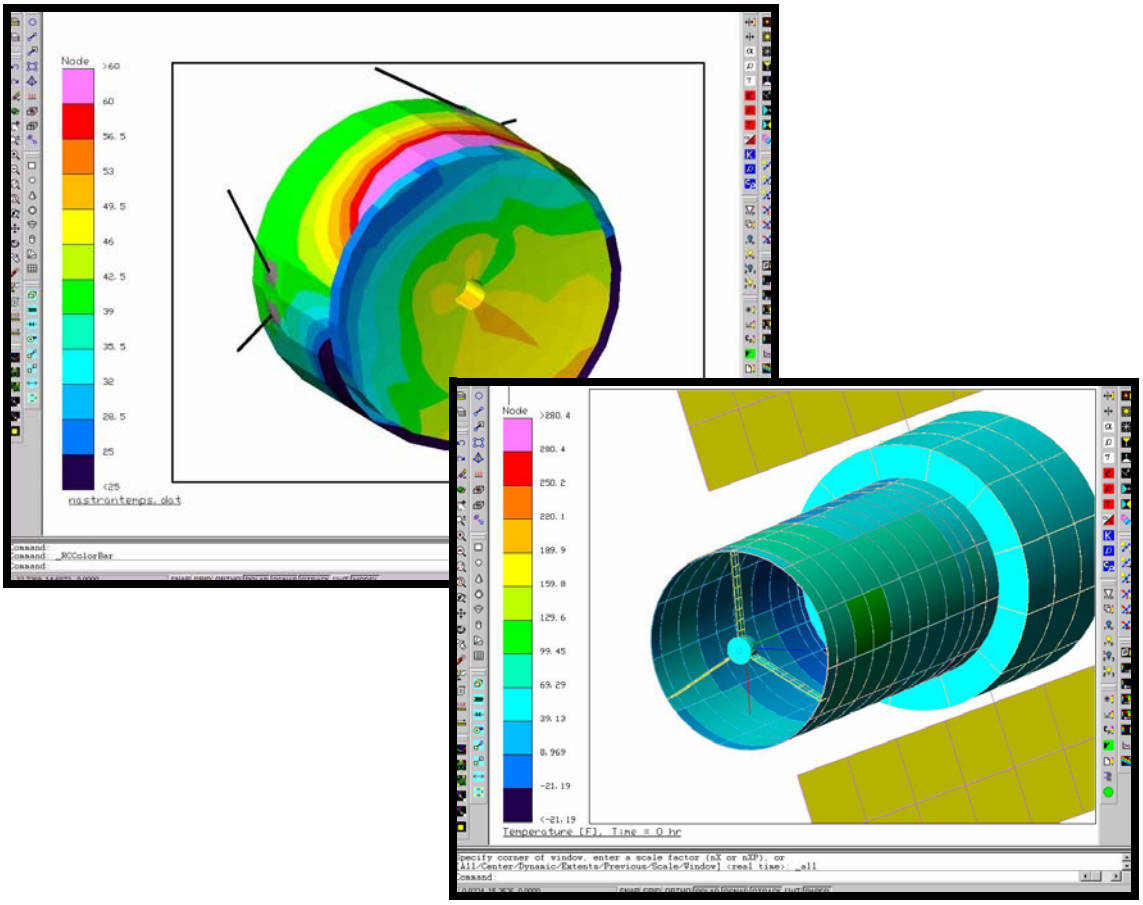
---

* Sinaps® replaced Sinaps*Plus*®.Development of Sinaps has been frozen at Version 5.6. While Sinaps is compatible with SINDA/FLUINT Version 5.7, new users are encouraged to investigate FloCAD®: GEOMETRY-BASED FLUID NETWORK MODELING on page xliv.

Thermal Desktop® and RadCAD®: GRAPHICAL (GEOMETRIC) CONCURRENT ENGINEERING

SINDA/FLUINT does not use nor enforce the use of geometry. Rather, it is an equation solver based not on a geometric description of a system but on an abstract mathematical (circuit or network) description. Radiation exchange, however, normally requires geometry to produce infrared radiation conductances ("RADKs") and absorbed solar fluxes. Also, manual generation of nodal capacitances and linear conductances using finite difference approximations is both tiring and error-prone, and nullifies integration with the design database. SINDA/FLUINT can solve finite element equations if they have been transformed into a network-style formulation, yet structural meshes are often inappropriate for thermal analysis. Thermal Desktop and RadCAD address all of these issues.

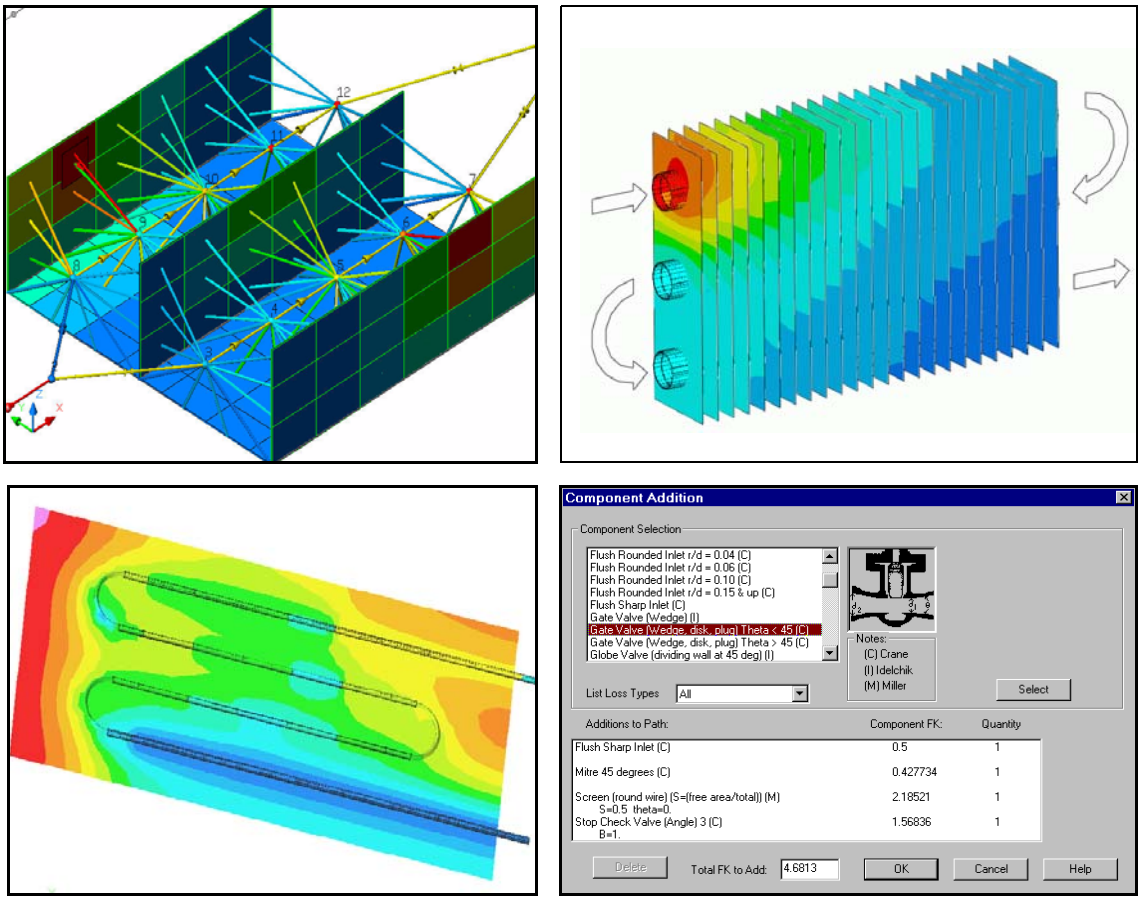
Thermal Desktop allows the thermal engineer to work concurrently with CAD-based designers and FEM-based structural engineers without necessarily using their models directly and without sacrificing good thermal modeling practices. Companion programs (page xlv) provide even tighter connections to CAD geometry, as well as advanced meshers. Thermal Desktop can also be used to postprocess SINDA/FLUINT results, as shown below.



Thermal Desktop is documented in a separate User's Manual. RadCAD is an optional radiation analyzer. Thermal Desktop® and RadCAD® are registered trademarks of C&R Technologies.

FloCAD®: GEOMETRY-BASED FLUID NETWORK MODELING

Thermal Desktop provides calculation of nodal capacitances as well as linear conductances (modeling solid conduction, insulation, contact conduction, etc.). RadCAD provides orbital heating rates and radiation conductances. FloCAD provides a full interface for fluid networks including generation of convective ties to surfaces (upper left in the diagram), and 1D flow modeling of fluid loops and heat pipes (with 1D or 2D pipe walls) attached to such surfaces (upper right and lower left).



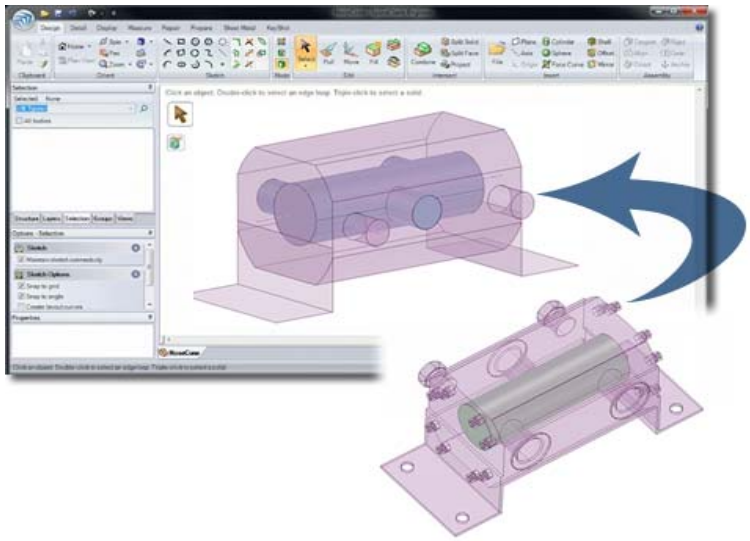
While FloCAD allows full access to FLUINT modeling capabilities, it is designed specifically to make FLUINT easier to use. This includes everything from inclusion of piping schedules and K-factor calculators (lower right in the diagram) to automatic re-generation of wall and duct models given changes in axial discretization, and modeling of rotating paths, partially filled vessels, etc.

FloCAD's line drawing methods can also be applied to simplify and automate the generation of data for SINDA's HEATPIPE and HEATPIPE2 simulation capabilities.

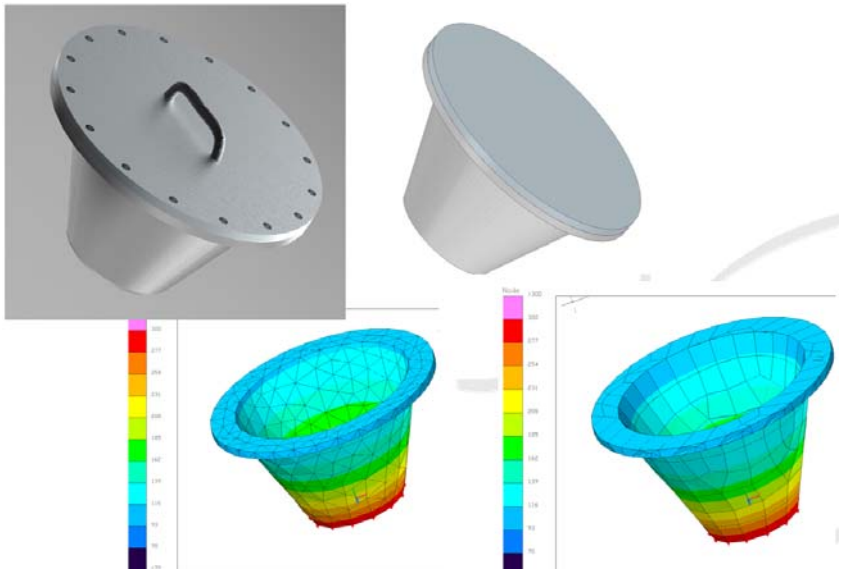
FloCAD is documented in the separate Thermal Desktop User's Manual. FloCAD® is a registered trademark of C&R Technologies.

CRTech TD Direct®: Automating Preparation of CAD-based Models

SpaceClaim is an easy-to-use yet powerful direct modeling CAD system. It imports files from virtually every CAD format, including Creo/Pro™, CATIA™, SolidWorks™, STEP, IGES, and ACIS. Geometry can be healed, defeatured, and modified. Using TD Direct, it can be marked up with thermal information (properties, contacts, convection zones, etc.) and meshed before linking directly with Thermal Desktop.

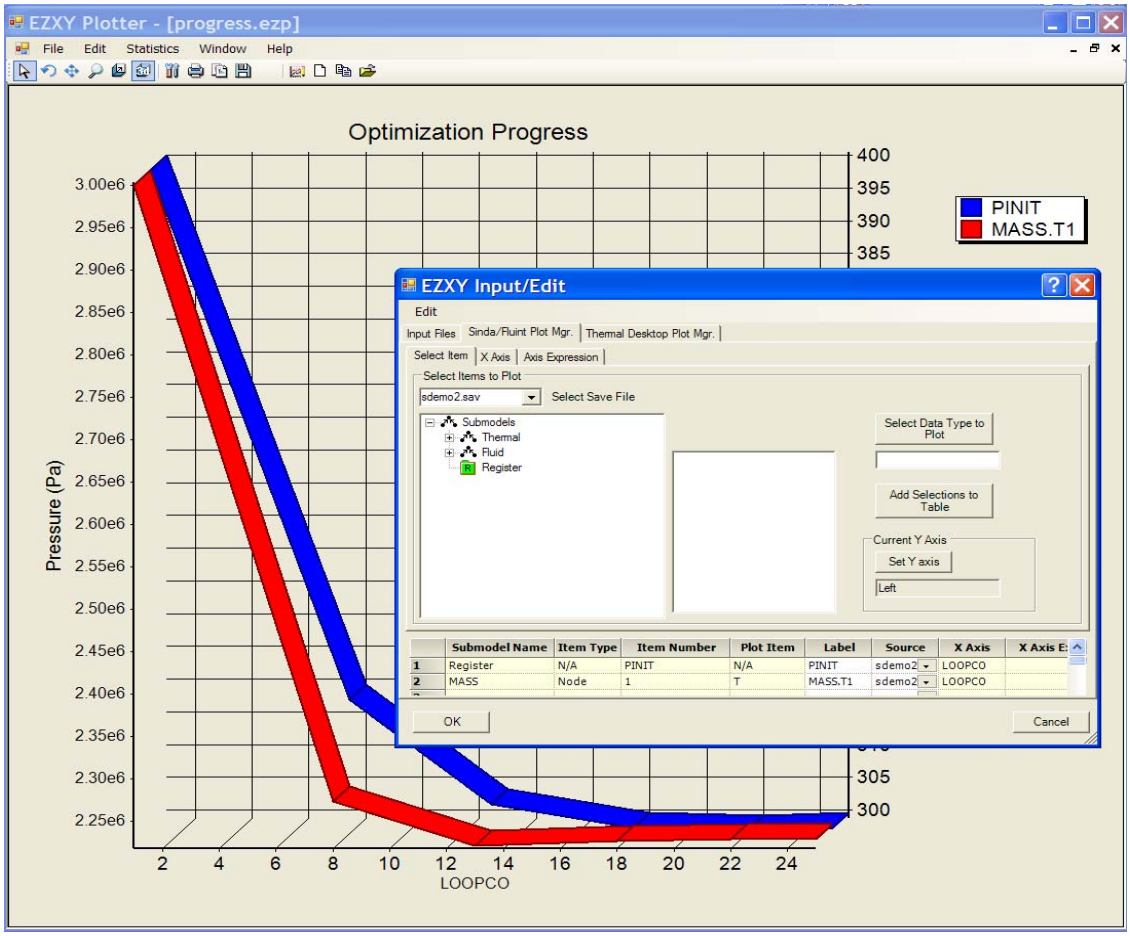


TD Direct performs customizable meshing of assemblies, allowing geometry to change while the Thermal Desktop model can be updated and ready to run with a single button click.



EZXY®:
External Plotting and Results Tabulation

EZXY is an interactive plotting package distributed as part of either Sinaps or Thermal Desktop. EZXY can plot data from several sources simultaneously,* and features extensive customization options, such as those shown below:



EZXY offers the ability to plot on multiple vertical axes, and supports of the Reliability Engineering module (histograms and scatter plots). Being a native Windows application, plots created within it can be pasted directly into software such as MS Word. In fact, as part of the EZXY install set, an MS Excel-based template is available to extract data from results files and produce plots.

EZXY is documented in the form of on-line help included with the code. EZXY® is a registered trademark of C&R Technologies. MS Word® and MS Excel® are registered trademarks of Microsoft.

* Including user text files, RadCAD RADKs, heating rates, and form factors

Resources Available to the User

Important training and software resources available to users are itemized below:

- Primers and tutorials for new users.
- Free online topic-specific mini-classes and videos (less than one hour each)
- An active user forum for tips, commonly asked questions, etc.
- FLUINT-compatible fluid property libraries for cryogens, new refrigerants, etc.
- Thermal Desktop® and RadCAD® and FloCAD®, the geometric concurrent thermal/fluid engineering environment described on page xliii and page xliv.
- CRTech TD Direct®: preparation of CAD models for Thermal Desktop, described on page xlv.
- EZXY®, an interactive plotting package described on page xlvi.
- Microsoft Excel™ templates that can be used for plotting or dynamically launching and controlling a SINDA/FLUINT run.
- Examples of COM-based connections between SINDA/FLUINT and third party programs such as MathWorks' MATLAB®, which can be used as an API (Application Program Interface).
- A translator that converts old SINDA and CINDA input files into the modern SINDA/FLUINT format.
- White papers and conference papers on applications of CRTech codes.
- Access to old (unsupported) versions as well as beta versions of upcoming releases.

For more information on any of these topics, contact C&R Technologies ("CRTech"):

C&R Technologies, Inc.
Boulder, Colorado USA
303-971-0292 (Phone)
303-971-0035 (FAX)
Web page: www.crtech.com

TABLE OF CONTENTS

1 INTRODUCTION AND OVERVIEW 1-1

1.1 How to Use This Manual 1-1
1.2 Learning SINDA/FLUINT 1-2
1.3 SINDA: A Historic Overview. 1-3
1.4 Graphical User Interfaces 1-4
1.5 Introduction to the R-C Network. 1-4
1.5.1 Transiently Heated Bar 1-5
1.5.2 Transiently Heated Bar Using Spreadsheet Options 1-7
1.5.3 Optimizing the Shape of a Heated Bar 1-8
1.5.4 Determining the Reliability of a Heated Bar 1-11
1.5.5 Designing a Reliable Heated Bar: Robust Design 1-13
1.6 System Structure: Preprocessing vs. Processing 1-16
1.7 Prerequisites and Conventions 1-18
1.7.1 Prerequisite Knowledge 1-18
1.7.2 Input Records 1-18
1.7.3 Format Definition Rules 1-19
1.7.4 Fortran Control Statements. 1-20
1.7.5 Terms and Data Conventions 1-21

2 SINDA INPUT FILE 2-1

2.1 Introduction to the Input File 2-1
2.2 Input File Structure 2-2
2.3 Input Block to Manual Section Cross-Reference 2-3
2.4 HEADER Records: Overview. 2-4
2.5 INSERT and INCLUDE Directives 2-5
2.5.1 Format and Examples. 2-5
2.5.2 The paths.txt File 2-6
2.5.3 Suggested Uses and Restrictions. 2-7
2.6 Miscellaneous Macroinstructions 2-8
2.6.1 Comments. 2-8
2.6.2 PSTART and PSTOP Records 2-8
2.6.3 FAC Record 2-8
2.7 OPTIONS DATA 2-10
2.7.1 File Definition 2-12
2.7.2 Other Options 2-13
2.7.3 Single Precision vs. Double Precision 2-14
2.7.4 Old Expressions (Version 3.1 and earlier) 2-15
2.7.5 OPTIONS DATA Formats 2-16

2.8 REGISTER DATA and Data Block Expressions 2-17
2.8.1 Introduction to Expressions and Registers 2-18
2.8.2 Evaluation and Usage of Expressions 2-19
2.8.3 Defining Registers: HEADER REGISTER DATA 2-21
2.8.4 Built-In Constants 2-23
2.8.5 Built-in Functions. 2-24
2.8.6 Logic Blocks and Expressions 2-25
2.8.6.1 Referring to Registers within Logic Blocks 2-25
2.8.6.2 Conflicts Between Logic and Expressions 2-26
2.8.7 Advanced Expressions: Conditional Operators. 2-27
2.8.8 Advanced Expressions: References to Processor Variables . . . 2-29
2.8.8.1 Increment Operator with Processor Variables 2-29
2.8.8.2 Indirect References to Processor Variables 2-30
2.8.8.3 Cautions Regarding Processor Variable References. 2-32
2.8.9 Limitations to Backward Compatibility of Old Expressions 2-33
2.9 Global (Named) USER DATA 2-35
2.9.1 Input Formats 2-35
2.9.2 Referencing Global User Constants in Logic Blocks. 2-36
2.9.3 Registers versus Named User Constants. 2-37
2.10 Submodel-specific (Numbered) USER DATA 2-38
2.10.1 Usage and Referencing 2-38
2.10.2 Input Formats 2-39
2.10.3 Input Examples 2-40
2.10.4 Registers versus Numbered User Constants 2-40
2.11 ARRAY DATA 2-42
2.11.1 Standard Array Data 2-42
2.11.2 Array Data Structures 2-43
2.11.3 Character Data Arrays 2-47
2.11.4 Referencing Array Data in Data and Logic Blocks 2-47
2.11.5 Referencing Character Array Data in Logic Blocks. 2-48
2.12 NODE DATA 2-49
2.12.1 Node Data Input 2-49
2.12.2 Referencing Node Data in Logic Blocks and Expressions 2-60
2.12.3 Backward Compatibility: the Kn and XKn option 2-61
2.12.4 Node Data Updating 2-61
2.13 SOURCE DATA. 2-62
2.13.1 Source Data Input. 2-62
2.13.2 Backward Compatibility: the Kn and XKn option. 2-69
2.13.3 Source Data Referencing 2-69
2.13.4 Source Data Updating. 2-69
2.14 CONDUCTOR DATA. 2-71
2.14.1 Conductor Data Input 2-74
2.14.2 Backward Compatibility: the Kn and XKn option. 2-90
2.14.3 Referencing Conductor Data in Logic Blocks 2-91
2.14.4 Conductor Data Updating 2-92

3 FLUID SYSTEM MODELING 3-1
3.1 Introduction and Overview 3-1
3.1.1 FLUINT Scope and Goals 3-1
3.1.2 The Fluid Submodel Concept 3-3
3.1.3 Networks and Elements 3-4
3.2 FLOW DATA Overview 3-8
3.2.1 General FLOW DATA Input Rules 3-8
3.2.2 HEADER FLOW DATA Record 3-10
3.3 Lumps 3-12
3.3.1 Tanks 3-19
3.3.2 Junctions 3-23
3.3.3 Plena 3-26
3.3.4 Lump Defaults in FLOW DATA 3-29
3.4 Paths 3-31
3.4.1 Tubes 3-32
3.4.1.1 Twinned Tubes: Slip Flow Simulation 3-33
3.4.1.2 Tube (and STUBE Connector) Input Parameters 3-35
3.4.1.3 Tube Subblock Format 3-43
3.4.2 Connectors 3-49
3.4.3 Path Defaults in FLOW DATA 3-56

3.5 Connector Device Models 3-61
3.5.1 NULL Option 3-62
3.5.2 STUBE Option 3-63
3.5.3 REDUCER Option 3-67
3.5.3.1 Sudden Contraction (MODEC=1) 3-68
3.5.3.2 Rounded-edged Contraction (Nozzle, MODEC=2) . . . 3-68
3.5.3.3 Conical Contraction (MODEC=3) 3-69
3.5.3.4 Smooth Contraction (Piping Reducer, MODEC=4) . . 3-70
3.5.3.5 REDUCER Formats in FLOW DATA 3-70
3.5.4 EXPANDER Option 3-72
3.5.4.1 Sudden Expansion (MODEC=1) 3-73
3.5.4.2 Curved Wall Diffuser (MODEC=2) 3-74
3.5.4.3 Conical Diffuser (MODEC=3) 3-75
3.5.4.4 Reducer used as Expander (MODEC=4) 3-76
3.5.4.5 EXPANDER Formats in FLOW DATA 3-77
3.5.5 CAPIL Option 3-79
3.5.6 LOSS Option 3-84
3.5.7 LOSS2 Option 3-86
3.5.8 CHKVLV Option 3-87
3.5.9 CTLVLV Option 3-88
3.5.10 UPRVLV Option (Upstream Pressure Regulator) 3-90
3.5.11 DPRVLV Option (Downstream Pressure Regulator) 3-92
3.5.12 ORIFICE Option 3-94
3.5.12.1 Compressible Flow Effects 3-96
3.5.12.2 Controllable Orifices 3-97
3.5.12.3 FLOW DATA Format 3-99
3.5.13 MFRSET Option 3-102
3.5.14 VFRSET Option 3-103
3.5.15 PUMP Option 3-105
3.5.15.1 Head-Flow Rate Curves and Maps 3-106
3.5.15.2 Defining Pump Efficiencies 3-108
3.5.15.3 QTMK and QTM: Turbomachinery Heat 3-109
3.5.15.4 Hydraulic Torque 3-109
3.5.15.5 Cavitation Detection and Modeling 3-110
3.5.15.6 Viscosity Corrections 3-111
3.5.15.7 Two-Phase Flow Corrections 3-112
3.5.15.8 FLOW DATA Format 3-113
3.5.15.9 Full Map Example and Discussion 3-121
3.5.16 TABULAR Option 3-124
3.5.17 TURBINE Option 3-138
3.5.17.1 Pressure-Flow Curves and Maps 3-139
3.5.17.2 Flow Rate (G) Options 3-141
3.5.17.3 Pressure (H) Options 3-143
3.5.17.4 Defining Turbine Efficiencies or Power 3-143
3.5.17.5 QTMK and QTM: Turbomachinery Power 3-144
3.5.17.6 Hydraulic Torque 3-145
3.5.17.7 Modeling Sonic Limitations (Choking) 3-146
3.5.17.8 FLOW DATA Format 3-146
3.5.17.9 Full Map Example:
GLIST= Method and HLIST1= Submethod 3-154

3.5.17.10 Full Map Example: GLIST= Method and HLIST= Submethod 3-157
3.5.17.11 ELISTR Example 3-159
3.5.18 COMPRESS Option 3-161
3.5.18.1 Pressure-Flow Curves and Maps 3-161
3.5.18.2 Flow Rate (G) Options 3-164
3.5.18.3 Pressure (H) Options 3-166
3.5.18.4 Defining Compressor Efficiencies or Power 3-166
3.5.18.5 QTMK and QTM: Turbomachinery Power 3-167
3.5.18.6 Hydraulic Torque 3-167
3.5.18.7 Specific Speed 3-168
3.5.18.8 Modeling Surging and Choking 3-169
3.5.18.9 FLOW DATA Format 3-171
3.5.18.10 Full Map Example: HLIST= Method 3-177
3.5.19 COMPPD Option. 3-182
3.5.19.1 Pressure Ratio vs. Volumetric Efficiency Maps 3-182
3.5.19.2 Specifying Volumetric Efficiency 3-183
3.5.19.3 Pressure (H) Options 3-184
3.5.19.4 Defining Compressor Efficiencies or Power 3-185
3.5.19.5 QTMK and QTM: Turbomachinery Power 3-186
3.5.19.6 Hydraulic Torque 3-186
3.5.19.7 FLOW DATA Format 3-187
3.5.19.8 Full Map Example: ELISTV= Method 3-191
3.6 Ties (Convection). 3-194
3.6.1 Basic Tie Parameters 3-195
3.6.2 Tie Types and Classifications 3-197
3.6.3 Lump-oriented (Lumped Parameter, Finite Difference) Ties. . . 3-198
3.6.3.1 HTU Ties 3-198
3.6.3.2 HTN and HTNC (Forced Convection) Ties. 3-203
3.6.3.3 HTP (“Pool Boiling” and Quasi-stagnant Convection) Ties 3-215
3.6.4 Path-oriented (Segment) Ties. 3-227
3.6.4.1 HTUS Ties 3-229
3.6.4.2 HTNS Ties 3-232
3.6.5 Effective Fluid Temperature TEF 3-237
3.6.6 Advanced Single-phase Forced Convection Heat Transfer Options 3-239
3.6.7 Advanced Two-phase Forced Convection Heat Transfer Options 3-242
3.6.7.1 Subcooled Boiling and Superheated Condensation 3-242
3.6.7.2 Critical Heat Flux (CHF) and Post-CHF Heat Transfer. 3-243
3.6.7.3 The NTWOP Status Code 3-246
3.6.7.4 Tie Algorithm for Boiling Heat Transfer 3-247
3.6.8 How Ties are Effected. 3-249
3.7 Fties (Fluid to fluid Heat Exchange). 3-250
3.7.1 USER Fties 3-250
3.7.2 CONSTQ Fties 3-251
3.7.3 AXIAL Fties. 3-252

3.8 Interface (iface) Elements and Subvolumes 3-255
3.8.1 Subvolumes 3-256
3.8.2 Interface Equations 3-257
3.8.2.1 Pressure/Volume Relationship 3-257
3.8.2.2 Inertia (Mass) Effects 3-258
3.8.2.3 Conservation of Volume 3-258
3.8.2.4 Volume Limits 3-259
3.8.3 Interface Device Types 3-259
3.8.3.1 FLAT iface 3-259
3.8.3.2 OFFSET iface 3-260
3.8.3.3 SPRING iface 3-260
3.8.3.4 WICK iface 3-262
3.8.3.5 SPHERE iface 3-267
3.8.3.6 NULL iface 3-268
3.8.4 Interface Input Formats in FLOW DATA 3-269
3.8.4.1 Defining Individual ifaces 3-269
3.8.4.2 Generating ifaces 3-271
3.8.5 Interface Duplication Factors 3-272
3.8.6 Usage Notes 3-272
3.8.6.1 COMP and VDOT: Using with ifaces 3-272
3.8.6.2 STEADY Solutions 3-273
3.8.6.3 HLDLMP: Using with ifaces 3-273
3.8.6.4 Using Twinned Tanks with ifaces 3-275
3.8.6.5 Body Forces 3-275
3.8.6.6 Logical References 3-276
3.8.6.7 Output Options 3-276
3.8.6.8 Miscellaneous Cautions 3-276
3.9 Macros: Element Generation and Component Models 3-279
3.9.1 Element Generation Macros 3-280
3.9.2 Duct Macrocommands 3-289
3.9.2.1 Macro-level Output Options 3-300
3.9.2.2 Extended Macros in Sinaps and FloCAD 3-300
3.9.3 Capillary Evaporator Pump Model 3-302
3.10 Sample Fluid Submodel 3-307
3.11 Capillary Models and Phase Suction Options 3-310
3.11.1 Phase Suction Options 3-310
3.11.2 Multiple-constituent Notes 3-312
3.11.3 Capillary Primed vs. Deprimed Decision 3-313
3.12 Symmetry and Duplication Options 3-317
3.12.1 How Path Duplication Factors Work 3-317
3.12.2 How Tie and Ftie Duplication Factors Work 3-322
3.12.3 Referencing Duplication Factors in Logic Blocks 3-322
3.12.4 Important Impacts of Duplication Factors 3-323
3.12.5 Modeling Tricks Using Zero and Noninteger Duplication Factors 3-325

3.13 Gravity and Acceleration Body Forces . . . 3-327
3.13.1 Definitions . . . 3-327
3.13.2 Referencing Body Force Options in Logic Blocks . . . 3-329
3.13.3 Examples . . . 3-329
3.13.3.1 Gravity . . . 3-330
3.13.3.2 General Acceleration . . . 3-330
3.13.4 Path End Locations (“Ports”) . . . 3-332
3.13.4.1 Cautions and Guidance: Ports . . . 3-336
3.14 Variable Volume and Compliant Tanks . . . 3-339
3.14.1 Definitions . . . 3-339
3.14.2 Referencing COMP and VDOT in Logic Blocks . . . 3-340
3.14.3 Modeling with Volume Rates . . . 3-341
3.14.4 Modeling with Compliances . . . 3-341
3.14.4.1 Pipe Wall Flexibility . . . 3-341
3.14.4.2 Gas Compressibility . . . 3-341
3.14.4.3 Liquid Compressibility . . . 3-343
3.14.4.4 Adding Compliances . . . 3-343
3.14.5 Compliances as Smoothness and Safety Measures . . . 3-344
3.15 Spatial Accelerations and the AC Factor . . . 3-346
3.15.1 Background: Modeling Changes in Flow Area . . . 3-347
3.15.2 Modeling with the AC Factor . . . 3-348
3.15.2.1 Area Changes . . . 3-349
3.15.2.2 Density Changes . . . 3-351
3.15.2.3 Combined Area and Density Changes . . . 3-352
3.15.2.4 Irrecoverable Losses Associated with Area Changes . . . 3-352
3.15.2.5 Inlets and Outlets (Entrances and Exhausts) . . . 3-353
3.15.2.6 Radial Blowing (Injection) and Suction (Extraction) . . . 3-354
3.15.3 Automatic Acceleration Calculations in Duct Macros . . . 3-356
3.16 Flow Regime Mapping and Related Pressure Drop . . . 3-359
3.16.1 Flow Regime Descriptions and Distinctions . . . 3-360
3.16.2 Regime Uncertainties: Simplification and Hysteresis . . . 3-362
3.16.3 Interactions with the ACCEL vector, Curved Tubes, and Spinning Flow Passages . . . 3-363
3.16.4 Multiple-constituent Two-phase Heat Transfer . . . 3-363
3.17 Slip Flow Modeling Using Twinned Paths . . . 3-364
3.17.1 An Introduction to Slip Flow . . . 3-364
3.17.2 Twinned Paths: Concept and Usage . . . 3-366
3.17.2.1 Input . . . 3-366
3.17.2.2 Twinned Paths with Twinned Tanks . . . 3-367
3.17.2.3 Output . . . 3-367
3.17.2.4 References in Logic Blocks . . . 3-368
3.17.3 Twinned Paths: Behavior . . . 3-369
3.17.3.1 Single Phase Limits: Homogeneous Mode . . . 3-369
3.17.3.2 Interactions with ACCEL and Rotating Paths . . . 3-369
3.17.3.3 Flow Regimes, Interface Drag, and Wall Friction Distribution . . . 3-370
3.17.3.4 Time Steps . . . 3-371

3.18 Sonic Limits (Choking) 3-372
3.18.1 Background and Computational Methods 3-372
3.18.1.1 AFTH: Throat Area 3-372
3.18.1.2 Two-Phase Sound Speed 3-373
3.18.1.3 Isentropic Expansion and Critical Flow Rate Estimation 3-374
3.18.1.4 Hysteresis in the Choking Calculation 3-374
3.18.1.5 Nonequilibrium Throat States 3-375
3.18.1.6 Alternative Nonequilibrium Method (Dyer, MCH=-3) 3-381
3.18.2 Relevant Path Variables for Choked Flow 3-384
3.18.3 Choked Flow Mode 3-386
3.18.4 Choking in Twinned Paths 3-387
3.19 Multiple-constituent Flows (Mixtures) 3-388
3.19.1 Mixture Property Rules 3-388
3.19.2 Usage Summary 3-389
3.19.2.1 Input Summary 3-389
3.19.2.2 Output Summary 3-389
3.19.2.3 References in Logic and Expressions 3-390
3.19.3 Two-Phase Mixtures 3-390
3.19.3.1 Phase Change 3-390
3.19.3.2 Two-Phase Heat Transfer Calculations in the Absence of Phase Change 3-391
3.19.3.3 Two-Phase Heat Transfer Calculations in the Presence of Phase Change 3-391
3.19.3.4 Thermal Equilibrium 3-392
3.19.3.5 Difficulties with Minuscule Vapor Mass in Two-phase Tanks 3-392
3.19.3.6 Two-Phase Data Requirements 3-393
3.19.4 Species-Specific Suction Options 3-393
3.19.4.1 Setting Species-Specific Suction 3-394
3.19.4.2 STAT Option Reporting and Accessing 3-395
3.19.5 Modeling Considerations and Limitations 3-395
3.19.5.1 Limitations and Restrictions 3-395
3.19.5.2 Minimum Concentration 3-396
3.19.5.3 Mixing Liquids with Different Densities 3-396
3.20 Modeling Tips for Efficient Use 3-397
3.20.1 Resolution 3-397
3.20.2 Initial conditions 3-399
3.20.3 Modeling Practices 3-400
3.20.4 Control Systems and User Logic 3-401

3.21 Alternative Fluid Descriptions: FPROP DATA 3-405
3.21.1 Introduction 3-405
3.21.2 Advanced Properties. 3-407
3.21.2.1 Diffusion Volumes (DIFV). 3-407
3.21.2.2 True Acentric Factor (WTRUE) 3-408
3.21.2.3 Modified Acentric Factor (WSRK) 3-409
3.21.2.4 Association Factor (PHI) 3-409
3.21.2.5 Heat of Formation and Enthalpy at STP. 3-409
3.21.3 FPROP DATA Blocks: General Input Formats 3-411
3.21.3.1 Header Subblocks 3-412
3.21.3.2 Array Subblocks 3-413
3.21.4 Perfect Gas (8000 Series) 3-414
3.21.4.1 Description 3-414
3.21.4.2 Input Formats 3-414
3.21.5 Nonvolatile Liquid (9000 Series) 3-418
3.21.5.1 Description 3-419
3.21.5.2 Input Formats 3-419
3.21.5.3 Miscible vs. Immiscible Liquid Mixtures 3-422
3.21.6 Simplified Two-Phase Fluid (7000 Series) 3-424
3.21.6.1 Description 3-424
3.21.6.2 Input Formats 3-426
3.21.6.3 7000 Series Fluids: Guidance and Restrictions 3-429
3.21.6.4 Example 7000 Series FPROP DATA Blocks 3-430
3.21.7 Advanced Two-Phase Fluid or Real Gas (6000 Series) 3-433
3.21.7.1 Introduction. 3-433
3.21.7.2 Fluid Description Requirements 3-434
3.21.7.3 NEVERLIQ (Real Gas) Variation 3-435
3.21.7.4 COMPLIQ (Compressible Liquid) Variation 3-435
3.21.7.5 Variable Molecular Weight Option 3-437
3.21.7.6 Regimes and Range Limits 3-438
3.21.7.7 Input Formats 3-440
3.21.7.8 Example 6000 Series FPROP DATA Block 3-445
3.21.7.9 Direct Calls to REFPROP: “R files” 3-448
3.21.7.10 REFPROP-generated Tables: “F files”. 3-449
3.22 Fluid Interactions: MIXTURE DATA. 3-451
3.22.1 Solubility Data. 3-451
3.22.2 MIXTURE DATA Input Formats 3-452
3.23 Gas Dissolution/Evolution Modeling Options. 3-457
3.23.1 Overview of Modeling Methods. 3-457
3.23.1.1 Basic Phenomena and Equations 3-457
3.23.1.2 Lumps and Associated Paths. 3-458
3.23.1.3 Scaling and Correlation Variables 3-460
3.23.1.4 Considerations when using Junctions 3-461
3.23.2 Additional Path Variables 3-462
3.23.3 Additional Lump Variables 3-463
3.23.4 Summary of Output Options 3-466
3.24 Chemical Reactions (Reacting Flows). 3-467
3.24.1 Heat of Reaction Correction (QCHEM). 3-469
3.24.2 Vanishing Reactants. 3-470

3.25 Modeling Phasic Nonequilibrium using Twinned Tanks 3-474
3.25.1 Overview 3-475
3.25.2 Defining the Interface 3-476
3.25.3 Input Formats 3-480
3.25.4 Mass Transfer Superpaths 3-483
3.25.5 Execution Control Options 3-485
3.25.6 Initializing Twinned Tanks 3-485
3.25.6.1 Summary of Initialization Utilities 3-486
3.25.6.2 TLV (Vapor/gas Temperature) Input Option 3-486
3.25.7 Anticipation of Phase Change 3-488
3.25.8 Example: Duct Macro 3-490
3.25.8.1 Cautions Regarding Modeling Horizontal, Stratified Lines 3-490
3.25.9 Using Stand-alone (nonmacro) Twinned Pairs 3-491
3.25.9.1 Default Values and Modeling Guidance 3-492
3.25.9.2 Two-phase (Separated Mode) Behavior 3-494
3.25.10 Example: Stand-alone (vs. Duct Macro) Tanks 3-495
3.26 Rotating Flow Passages 3-499
3.26.1 Velocity Components 3-501
3.26.2 Rotational Velocity Ratio (Relative Wall Motion) 3-503
3.26.3 Torque, Efficiency, and Power 3-505
3.26.3.1 Default Mode: Effective Turbomachine (user-defined EFFP) 3-506
3.26.3.2 Alternative Mode: Dissipative Element (user-defined TORQ) 3-507
3.26.3.3 Notes on Rotating Turbomachinery Connectors . . . 3-508
3.26.4 Use with Tubes and STUBE Connectors 3-508
3.26.4.1 Effect on Duct Pressure Drop (RVR<1) 3-508
3.26.4.2 Effect on Convection Ties (RVR<1) 3-510
3.27 Flat-front Two-phase Flow Modeling 3-512
3.27.1 Discretization Requirements 3-514
3.27.2 Flat-front Phase Suction Options 3-516
3.27.3 Axially Stratified Flow Regime 3-517
3.27.4 Brief Input Examples of Flat-front Modeling in Duct Macros . . . 3-518
3.27.5 Notes on Flat-front Modeling 3-519
3.27.5.1 Nonduct Devices 3-519
3.27.5.2 Flat-front at Tees 3-520
3.27.5.3 Computational Noise at Transitions 3-521
3.27.5.4 Flat Front Interactions with Path Ports 3-522

4 EXECUTION, LOGIC, AND CONTROL 4-1

4.1 LOGIC BLOCKS 4-4
4.1.1 Macroinstructions in Logic Blocks 4-6
4.1.1.1 BUILD and BUILDF Instructions 4-6
4.1.1.2 FSTART and FSTOP Instructions 4-8
4.1.1.3 The Spelling Checking Option: SPELLON and SPELLOFF Instructions 4-8
4.1.1.4 DEFMOD Instruction 4-9
4.1.2 Functional Descriptions of Logic Blocks 4-10
4.1.3 Applications Guidelines 4-18
4.1.3.1 Real and Integer Variable Types 4-18
4.1.3.2 OPERATIONS Block 4-18
4.1.3.3 VARIABLES 0 Block 4-20
4.1.3.4 VARIABLES 1 Block 4-21
4.1.3.5 VARIABLES 2 Block 4-23
4.1.3.6 FLOGIC Blocks 4-24
4.1.3.7 OUTPUT CALLS Block 4-26
4.1.3.8 SUBROUTINES Block and CALL COMMON 4-27
4.1.3.9 Program Data Modules 4-29
4.1.3.10 Summary of Translatable Parameters 4-31
4.1.4 Logic Translation 4-33
4.1.4.1 Translatable Statements 4-34
4.1.4.2 Function and Subroutine Argument Checking 4-41
4.1.4.3 F-Type Statements 4-43
4.1.4.4 Internal Storage vs. Logical Reference 4-44
4.1.4.5 Cautions Regarding Reference to Registers in Logic Blocks 4-46
4.1.4.6 Logic used in Versions Older than 5.3 4-46
4.1.4.7 New Functionality or Calling Variations as of Version 5.3 4-47
4.2 Network Solution Routines 4-49
4.2.1 FORWRD: Transient, Explicit 4-51
4.2.2 TRANSIENT (“FWDBCK”): Transient, Implicit Second Order . . . 4-55
4.2.3 STDSTL: Steady State (Fluid Pseudo-transient) 4-59
4.2.4 STEADY (“FASTIC”): Steady State 4-64
4.3 CONTROL DATA: Control Constant Input 4-66
4.3.1 Defining CONTROL Data Values 4-66
4.3.2 CONTROL Data Names and Functions 4-66
4.3.3 CONTROL Data Formats 4-68
4.4 Control Constant Usage 4-70
4.4.1 SINDA Control Constant Summary 4-76
4.4.2 FLUINT Control Constant Summary 4-89
4.4.3 Time and Time Step Control Constants 4-94
4.5 FLUINT Execution and Control 4-96
4.5.1 Fluid Submodel Steady-State Convergence 4-96
4.5.2 Fluid Submodel Time Steps 4-98
4.5.3 FLUINT Trouble Shooting 4-102

4.6 Overview of Processor Output Operations 4-107
4.6.1 Text Output 4-107
4.6.1.1 The OUTPUT File 4-107
4.6.1.2 OUTPUT File Content: Common OUTPUT CALLS Options 4-107
4.6.1.3 Output Frequency 4-108
4.6.1.4 User Files 4-108
4.6.2 Binary Output 4-109
4.6.2.1 The SAVE File or Folder 4-110
4.6.2.2 Different Files with a Common Format 4-110
4.7 Multiple Case/Multiple Run Operations 4-112
4.7.1 Multiple Cases Within a Single Run: SAVPAR, SVPART, RESPAR 4-112
4.7.2 Multiple Cases Within a Single Run: DTIMES 4-113
4.7.3 Restart Operations Between Runs 4-114
4.7.4 Crash Files: Abort Recovery 4-116
4.8 Controlling Submodel States 4-118
4.9 Dynamic Registers: The Underlying Spreadsheet 4-120
4.9.1 Introduction and Importance 4-120
4.9.2 Nomenclature 4-123
4.9.3 Affected Variables 4-123
4.9.3.1 Special SOURCE DATA Rules 4-124
4.9.3.2 Derived Expressions 4-125
4.9.3.3 Formula Defining Plenum States 4-125
4.9.4 Changing Registers in the Processor (e.g., in Logic Blocks) . . 4-126
4.9.5 Automatic Spreadsheet Propagation 4-126
4.9.6 Assuming Control of the Update Process 4-127
4.9.6.1 Sequence and Location of Updates 4-128
4.9.6.2 Disconnecting and Reconnecting Parameters 4-129
4.9.6.3 Discrepancies and Forcing Updates 4-129
4.9.7 RESPAR, RESTAR, and Registers 4-130
4.9.8 Common Misconceptions: Fortran vs. Expressions 4-131

5 ADVANCED DESIGN FEATURES 5-1

5.1 Introduction to the Solver 5-4
5.1.1 Typical Applications 5-4
5.1.2 Thermal Desktop “Dynamic Mode” 5-5
5.1.3 Optimization Terms 5-6
5.1.4 Overview of the Solver 5-7
5.1.5 A Brief Example 5-9
5.2 SOLVER Routine 5-10
5.2.1 SOLOGIC Blocks 5-11
5.2.2 SOLOUTPUT CALLS 5-13
5.3 HEADER PROCEDURE 5-14
5.4 HEADER SOLVER DATA 5-15
5.4.1 Using OBJECT and GOAL 5-19
5.5 HEADER DESIGN DATA 5-21
5.6 HEADER CONSTRAINT DATA 5-24

5.7 Solver Support Subroutines 5-29
5.7.1 Output Routines 5-29
5.7.2 Auxiliary Routines 5-29
5.7.3 Diagnostic Utility 5-30
5.8 How The Solver Works 5-31
5.9 Extended Solver Usage 5-34
5.9.1 Dealing With Discrete Design Variables 5-34
5.9.2 Handling Multiple Objectives 5-34
5.10 Test Data Correlation Methods 5-36
5.10.1 Identifying Key Uncertainties 5-36
5.10.2 Varying the Uncertain Parameters 5-37
5.10.3 Comparison Methods 5-38
5.10.3.1 Least Squares Error 5-38
5.10.3.2 Least Average and Cubic Errors 5-39
5.10.3.3 Minimized Maximum Error (Minimax) 5-39
5.10.3.4 Comparisons at Multiple Times or Multiple Cases . . . 5-42
5.10.3.5 Uneven Weightings 5-44
5.10.4 Test Data Correlation Support Routines 5-44
5.11 Potential Solver Problem Areas 5-46
5.11.1 Large Run Times 5-46
5.11.2 Nonresponsive Solutions: Apparent Stalling 5-47
5.11.2.1 Scaling Problems: Large Changes in OBJECT and Design Variables 5-47
5.11.2.2 Weak Functions 5-49
5.11.2.3 Noisy Functions: Accuracy of the Underlying Procedure 5-49
5.11.2.4 Discontinuous Functions 5-49
5.11.3 Double Checking an Answer 5-50
5.11.4 Solution With Violated Constraints 5-52
5.11.5 Solution Worse Than Before 5-52
5.11.6 Solution on the Edge of Infeasibility 5-52
5.12 Brief Solver Example: Constrained Optimization 5-53
5.13 Brief Solver Example: Goal Seeking 5-56
5.14 Brief Solver Example: Test Data Correlation 5-58
5.14.1 Least Squares (RMSERR and SUMSQR) Examples 5-58
5.14.2 Maximum Error (MAXERR) Example 5-60
5.14.3 MINIMAX Example 5-61
5.15 Design Space Scanning 5-64
5.15.1 Parametric Sweep of a Design Variable 5-64
5.15.2 Latin Hypercube Scanning 5-65
5.15.3 Full Factorial Scanning 5-68
5.15.4 Scanning in the Presence of Equality Constraints 5-71

5.16 Introduction to Reliability Engineering 5-73
5.16.1 Reliability Engineering Terms 5-73
5.16.2 Distribution Functions 5-74
5.16.3 Sampling Techniques 5-75
5.16.4 Reliability Estimation 5-76
5.16.4.1 Based on Sampling 5-76
5.16.4.2 Based on Gradients 5-77
5.16.4.3 Foresight vs. Hindsight 5-78
5.16.5 Usage Overview 5-78
5.16.6 A Brief Example 5-79
5.17 HEADER RANDOM DATA 5-81
5.18 HEADER RELCONSTRAINT DATA 5-84
5.19 SAMPLE Routine 5-88
5.19.1 RELOUTPUT CALLS 5-90
5.20 DSAMPLE Routine 5-92
5.21 RELEST Routine 5-95
5.22 HEADER RELPROCEDURE 5-98
5.23 Database Options and Cumulative Runs 5-100
5.23.1 Reliability Engineering Databases and other Binary Formats 5-100
5.23.2 SAVEDB and RESTDB 5-100
5.23.3 Using REDB Files with SAMPLE and DSAMPLE 5-101
5.23.4 Using REDB Files and Folders with RELEST 5-102
5.23.5 Hindsight: Using Postprocessing 5-102
5.24 Reliability Support Subroutines 5-103
5.24.1 Output Routines 5-103
5.24.2 Auxiliary Subroutines 5-103
5.25 Brief Reliability Engineering Example 5-106
5.26 Parametric Sweep of a Random Variable 5-109
5.27 Robust Design: Reliability-based Optimization 5-110
5.27.1 Problems Using Reliability as OBJECT or Optimization Constraint 5-112
5.27.2 Reliability Constraints vs. Optimization Constraints 5-113
5.27.3 Advanced Hint for Further Reducing Run Times 5-114

6 REFERENCE SUMMARY 6-1

6.1 Basic Conventions - Data Blocks 6-2
6.2 Options Data 6-4
6.3 Register Data 6-5
6.4 Node Data 6-6
6.5 Source Data 6-9
6.6 Conductor Data 6-11
6.7 Control Data 6-16
6.8 User Data 6-18
6.9 Array Data 6-19
6.10 Solver Data 6-20
6.11 Design Data 6-22
6.12 Constraint Data 6-23

6.13 Random Data 6-24
6.14 Reliability Constraint Data 6-25
6.15 Flow Data 6-26
6.15.1 LU (Lump) Subblocks 6-27
6.15.2 PA (Path) Subblocks 6-28
6.15.3 T (Tie) Subblocks 6-33
6.15.4 FT (Ftie) Subblocks 6-34
6.15.5 IF (iface) Subblocks 6-35
6.15.6 M (Macro) Subblocks 6-36
6.16 FPROP Data 6-39
6.16.1 Perfect Gas (8000 Series) 6-39
6.16.2 Incompressible Liquid (9000 Series) 6-40
6.16.3 Simplified Two-Phase (7000 Series) 6-40
6.16.4 Advanced Two-Phase (6000 Series) 6-41
6.17 MIXTURE Data 6-44
6.18 Logic Blocks 6-45
6.18.1 Block Formats 6-45
6.18.2 F-Type FORTRAN Statements 6-46
6.18.3 M-Type FORTRAN Statements 6-46
6.18.4 FLUINT Variables 6-47
6.18.5 Reserved Words 6-50
6.19 Model Size Limits 6-53

7 SUBROUTINE LIBRARY 7-1
7.1 Introduction 7-1
7.1.1 Basic Conventions 7-1
7.1.2 Interpolation Subroutines 7-3
7.1.3 Input and Output Subroutines 7-4
7.1.4 Application and Utility Subroutines 7-4
7.2 Subroutine Name Index 7-5
7.3 Interpolation - Extrapolation Subroutines 7-14
7.3.1 Temperature-Dependent Variable Interpolation 7-16
7.3.2 Time-Dependent Variable Interpolation 7-16
7.3.3 Generalized Linear Interpolation 7-17
7.3.4 Interpolation Using Cyclical Arrays 7-24
7.3.5 Generalized Parabolic Interpolation 7-27
7.3.6 Generalized Lagrangian Interpolation 7-29
7.3.7 Step Interpolation 7-30

7.4 Output Subroutines 7-33
7.4.1 Printout of Thermal Network Attributes 7-33
7.4.2 Printout of Network Connectivity/Heat Flow Map 7-34
7.4.3 Binary Formats for Files and Folders 7-36
7.4.4 Restart Data Save Subroutine 7-37
7.4.5 Variable Save Routine 7-40
7.4.6 Saving Data for Parametric Cases 7-41
7.4.7 Temperature Save Subroutine 7-43
7.4.8 Printing Minimum and Maximum Temperatures 7-44
7.4.9 Array Data Printout 7-45
7.4.10 Numerical Differencing Characteristics Printout 7-46
7.4.11 Sorted Temperature Print 7-47
7.4.12 Save Temperatures for Boundary Temperature Driving 7-47
7.4.13 Print Heater Node Q Values 7-49
7.4.14 Print Routine Controller 7-50
7.4.15 Tabulate Registers and Expressions 7-50
7.4.16 Reset Page Header Information 7-51
7.4.17 Creating Additional User Files 7-52
7.4.18 Model Size Information 7-54
7.4.19 Changing OUTPUT and SAVE Destinations Dynamically 7-54
7.4.20 Sink Temperature Utilities 7-57
7.4.21 Submodel-level Map 7-65
7.5 Input Subroutines 7-68
7.5.1 Resetting Some or all Nodes' Temperature in a Submodel 7-68
7.5.2 Boundary Node Temperature Driving 7-69
7.5.3 Re-initialization for Parametric Cases 7-70
7.5.4 Reading In Parameters from the Restart Folder 7-71

7.6 Utility Subroutines 7-73
7.6.1 Finding User Array Locations 7-73
7.6.2 Finding Conductor Value Locations 7-74
7.6.3 Finding CARRAY (Character Array) Locations 7-75
7.6.4 Finding Submodel Name Locations 7-75
7.6.5 Finding Node Attribute Locations 7-76
7.6.6 Finding User Constant Locations 7-76
7.6.7 Area Integration 7-77
7.6.8 Array Integration 7-77
7.6.9 Least Squares Curve Fitting 7-78
7.6.10 Putting a Submodel in a Boundary State 7-78
7.6.11 Reactivating a Boundary State Submodel 7-79
7.6.12 Calculating Transient Energy Balance 7-79
7.6.13 Net Nodal Heat input. 7-80
7.6.14 Accessing Variables as Double Precision. 7-81
7.6.15 Toggling Double Precision Options On and Off 7-82
7.6.16 Heat Flows and Other Data about a Conductor 7-83
7.6.17 Heat Flows Between Nodes or Groups of Nodes 7-85
7.6.18 Sparse Matrix Memory Utilization Reporting. 7-88
7.6.19 Temporarily Holding a Node's Temperature 7-89
7.6.20 Temporarily Holding Many Nodes' Temperatures 7-90
7.6.21 Temporarily Making Diffusion Nodes Arithmetic 7-93
7.6.22 Execution Time Utilities. 7-100
7.6.23 Print Time Step History List 7-101
7.7 Applications Subroutines 7-103
7.7.1 Heater Node Energy Requirements 7-103
7.7.2 Thermostatic Switch with Hysteresis 7-104
7.7.3 Generalized Heater Controller Simulations. 7-106
7.7.4 Generalized Cooler Controller Simulations. 7-109
7.7.5 SINDA-based Heat Exchanger Calculations. 7-112
7.7.6 Material Conductance and Capacitance 7-117
7.7.7 Phase Change Simulation 7-119
7.7.8 PID Controller Simulation Utilities 7-121
7.7.9 Heat Pipes 7-130
7.7.10 Thermoelectric Coolers (TECs) and other Peltier Devices 7-146
7.7.11 Surface Recession due to Melting or Sublimation 7-157
7.7.11.1 Theory and Modeling Approach 7-157
7.7.11.2 Surface Recession Routines 7-160
7.7.11.3 Example 7-168
7.7.12 Ice Accretion and Melting 7-170
7.7.12.1 Example: Frost Formation on a Pipe 7-174
7.7.12.2 Preliminary Access for Thermal Desktop Users 7-181
7.7.13 Standard Atmospheric Temperature and Pressure 7-183

7.8 Arithmetic Subroutines 7-184
7.8.1 Add Two Arrays 7-185
7.8.2 Add a Constant to Elements of an Array 7-185
7.8.3 Divide Corresponding Elements of Arrays 7-186
7.8.4 Divide a Constant into Elements of an Array 7-186
7.8.5 Multiply Two Arrays 7-187
7.8.6 Multiply the Elements of an Array by a Constant 7-187
7.8.7 Subtract Corresponding Elements Of Arrays 7-188
7.8.8 Subtract a Constant from Elements of an Array 7-188
7.8.9 Sum an Array of Floating Point Values 7-189
7.8.10 Invert Array Elements 7-189
7.8.11 Divide an Array into a Constant 7-190
7.8.12 Inverse of Sum of Inverses 7-190
7.8.13 Array Difference and Multiplication 7-191
7.8.14 Floating Difference and Multiply 7-191
7.8.15 Array Difference and Multiply 7-192
7.8.16 Floating Point Calculation 7-192
7.9 Co-solved (Auxiliary) Differential Equations 7-193
7.9.1 First Order Differential Equations 7-194
7.9.1.1 Example: Pump Start-up 7-197
7.9.2 Second Order Differential Equations 7-199
7.9.2.1 Example: Valve Stem Motion 7-202
7.9.3 Customized Time Step Controls 7-204
7.9.4 Output Tabulation 7-206
7.10 Root and Minimum Finder Utilities 7-207
7.10.1 Co-Solved Minimum Finders 7-209
7.10.2 Co-Solved Root Finders 7-215

7.11 FLUINT Subroutines 7-221
7.11.1 FLUINT Utility Subroutines 7-221
7.11.1.1 Change Utilities 7-221
7.11.1.2 Changing Lump Constituent Mass Fractions 7-228
7.11.1.3 Controlling Slip and Nonequilibrium Modes 7-235
7.11.1.4 Flat Front (Fill/Purge) Duct Modeling Modes 7-241
7.11.1.5 Lump Solution Mode Options 7-243
7.11.1.6 Dynamic Translation Utilities 7-248
7.11.1.7 Capillary Modeling Utilities 7-255
7.11.1.8 Moving a Path or Tie 7-257
7.11.1.9 Auxiliary Calculations 7-260
7.11.1.10 Inquiries 7-262
7.11.2 Property Subroutines 7-272
7.11.2.1 VSOSFV Description 7-274
7.11.2.2 Multiple-constituent (Mixture) Properties 7-275
7.11.2.3 Henry's Coefficient 7-277
7.11.2.4 Psychrometric Utilities 7-278
7.11.2.5 Simplifying Fluid Properties 7-282
7.11.3 Heat Transfer Correlation Functions and Routines. 7-287
7.11.4 Auxiliary Heat Transfer Correlations 7-301
7.11.4.1 Pool Boiling and Limits 7-301
7.11.4.2 External Flow Heat Transfer Correlations 7-306
7.11.4.3 Jet Impingement Cooling 7-312
7.11.4.4 Heat Transfer Enhancement for Rough Walls 7-318
7.11.4.5 Heat Transfer Enhancement for Curved Tubes 7-320
7.11.4.6 Heat Transfer Enhancement for Developing Flows (Entrances) 7-322
7.11.4.7 Heat Transfer Enhancement: Internal Fins or Protrusions 7-325
7.11.4.8 Natural (Free) Convection Correlations 7-328
7.11.4.9 Mixed Convection for Filling and Emptying Cylinders 7-354
7.11.4.10 Heat Transfer Correlations for Flow Between Rotating and Stationary Disks 7-357
7.11.5 Pressure Drop Correlation Functions and Routines 7-360
7.11.6 Auxiliary Pressure Drop Correlations 7-364
7.11.6.1 Friction Correction for Heated or Cooled Ducts 7-364
7.11.6.2 Friction Correction for Mixed and Rarefied Flows . . 7-365
7.11.7 Auxiliary Correlations for Torques in Gaps 7-367
7.11.7.1 Flows Between Rotating and Stationary Disks. 7-367
7.11.7.2 Flows Between Rotating and Stationary Cylinders 7-376
7.11.8 FLUINT Output Routines 7-382
7.11.9 FLUINT Simulation Routines 7-407
7.11.9.1 Compressors and Other Turbomachinery 7-407
7.11.9.2 NCGBUB: Noncondensible Gas Bubble Simulation 7-409
7.11.9.3 IFACE-like Utility for Linking Tanks 7-411
7.11.9.4 Auxiliary Routines for Conduction in Wet Wicks . . . 7-416
7.11.9.5 Choked Flow Detection and Simulation Aids 7-420

7.11.9.6 Modeling Water Hammer and Acoustic Waves 7-423
7.11.9.7 Modeling Wyes (Ys), Tees (Ts), and Manifolds 7-428
7.11.9.8 Two-phase Loop Balancer 7-438
7.11.9.9 Chemical Equilibrium and Reaction Rate Utility 7-441
7.11.9.10 Importing TankCalc Data for Partially Filled Vessels 7-449
7.11.10 Heat Exchanger Design and Simulation: System-Level 7-455
7.11.10.1 Calculating Heat Capacity Rates (Cmin, Cmax). . . . 7-455
7.11.10.2 Effectiveness and NTU (or UAtot) Relationships . . . 7-457
7.11.10.3 Heat Exchanger Simulating or Sizing Utility 7-461
7.11.11 Heat Exchanger Simulation: Detailed Approach 7-466
7.11.11.1 Compact Heat Exchangers: Colburn J Factor 7-466
7.11.11.2 Compact Heat Exchangers: Fanning Friction Factor 7-468
7.11.11.3 Compact Heat Exchanger Example 7-470
7.11.11.4 Offset Strip Fin Correlations 7-472
7.12 Dynamic Register Utility Routines 7-476
7.12.1 Updating the Model 7-476
7.12.2 Disconnecting Automatic Updates 7-484
7.12.3 Reconnecting Automatic Updates. 7-486
7.12.4 Checking a Automatic Update Connection 7-488
7.12.5 Register Utilities 7-489
7.12.6 Changing Nonregister Expressions Dynamically 7-491
7.12.7 Parameter Sweep Utility 7-494
7.13 Solver Output and Utility Routines 7-497
7.13.1 Design Variable Output and Manipulations. 7-497
7.13.2 Constraint Variable Output and Manipulations 7-500
7.13.3 Test Data Correlation Support Routines 7-504
7.13.4 Solver Connectivity and Sensitivity Checker. 7-509
7.14 Reliability Engineering Output and Utility Routines 7-510
7.14.1 Output Options 7-510
7.14.2 Database Options 7-512
7.14.3 Utility Routines 7-515

Appendices

Appendix A Frictional Pressure Drop Correlations A-1
A.1 Single-phase Correlation A-1
A.2 Two-phase Correlations. A-1

Appendix B Forced Convective Heat Transfer Correlations A-4
B.1 Single-phase Flow, Heating or Cooling A-4
B.2 Two-phase Flow, Boiling A-6
B.3 Two-phase Flow, Condensation. A-7
B.4 Two-phase Flow, Mixtures without Phase Change A-7
B.5 Two-phase Flow, Condensation of Mixtures A-8
B.6 Two-phase Flow, Boiling of Mixtures A-10

B.7 Dissolution and Evolution of Gases A-11
B.7.1 Heterogeneous Dissolution and Evolution A-11
B.7.2 Homogeneous Evolution (Nucleation) A-13

Appendix C Available Fluids and Range Limits A-15
C.1 Standard Library Fluids A-15
C.2 User Defined Fluids A-17
C.3 REFPROP 6000 Series fluids A-18
C.4 Range Limits of Mixtures A-18

Appendix D Unit Systems A-21

Appendix E Summary of FLUINT Numerical Methods A-22
E.1 Basic Governing Equations A-24
E.2 Formulation of the Equation Set A-27
E.3 Incompressible Tanks A-29
E.4 Phase Suction Options A-30
E.5 Interface (iface) Implementation A-30
E.6 Time Step Algorithm A-31

Appendix F Sparse Matrix Inversion A-32
F.1 YSMP (MATMET=1,2 and FLUINT solutions) A-32
F.2 AMG-CG (MATMET=11,12) A-33

Appendix G Run-Time Control and Plotting A-35
G.1 SINDAWIN Run-time Control A-35
G.2 Progress Plotter A-36

Appendix H External Programs A-38
H.1 EZXY® Plotting Utility A-38
H.2 RAPPR: Speed Improvements Using Simplified Fluids A-38

List of Tables

Table 1-1 Complete Input File for Sample Problem 1-8
Table 1-2 Complete Input File Using Registers 1-9
Table 1-3 Optimizing Bar Shape 1-12
Table 1-4 Determining Bar Reliability 1-14
Table 2-1 OPTIONS DATA Summary 2-10
Table 2-2 Program File Definitions 2-12
Table 2-3 OPTIONS DATA Block Example 2-16
Table 2-4 Built-in Constants 2-24
Table 2-5 Built-in Functions 2-24
Table 2-6 Indirect Operators for Referencing Processor Variables in Expressions 2-30
Table 2-7 Summary of NODE Data Input Options 2-51
Table 2-8 Summary of SOURCE DATA Input Options 2-62
Table 2-9 Summary of CONDUCTOR Data Input Options 2-75
Table 2-10 SIV and SPV Family of Temperature-varying Conductors 2-83
Table 3-1 Summary of FLUINT Network Elements 3-6
Table 3-2 Lump State Initialization (without “PL!”) 3-13
Table 3-3 Tube (and STUBE Connector) Descriptive Parameters 3-34
Table 3-4 FLUINT Connector Device Models 3-61
Table 3-5 G (“Flow rate”) Units for PUMP, TABULAR, TURBINE, COMPRESS, COMPPD Connectors 3-118
Table 3-6 H (“Head”) Units for PUMP, TABULAR, TURBINE, COMPRESS, COMPPD Connectors 3-119
Table 3-7 Summary of PUMP Input Options 3-121
Table 3-8 Summary of TURBINE Input Options 3-154
Table 3-9 Summary of COMPRESS Input Options 3-178
Table 3-10 Summary of COMPPD Input Options 3-191
Table 3-11 Boiling Heat Transfer Calculation for HTN, HTNC Ties 3-248
Table 3-12 Sample Model Description 3-308
Table 3-13 FLOW DATA Block for Sample System 3-308
Table 3-14 Diffusion Volumes and Elemental Increments 3-408
Table 3-15 Modified Acentric Values (WSRK) for Selected Fluids 3-409
Table 3-16 Summary of Scalable Dissolution/Evolution Variables 3-461
Table 3-17 Summary of Scalable Twinned Tank Interface Variables 3-478
Table 3-18 Sample of Stand-alone Twinned Tanks 3-495
Table 4-1 Complete Input File for Sample Problem 4-2
Table 4-2 Control Constant Definitions 4-67
Table 4-3 CONTROL DATA Block Example 4-69
Table 4-4 User-Specified Control Constants Required by Solution Routines 4-75
Table 4-5 Summary of Time and Time Step Control Parameters 4-95
Table 4-6 Three Node Bar Model Using Dynamic Registers 4-121
Table 5-1 Overview of Advanced Design Features 5-1
Table 5-2 Solver Control Data Definitions 5-15
Table 5-3 Results of Various Least Squares Methods 5-60
Table 5-4 Results of Simplified MAXERR Minimizations 5-61
Table 5-5 Results of Formal MINIMAX Correlations 5-62
Table 5-6 Comparison of Reliability Estimation Routines 5-88

Table 6-1 Reserved Word List 6-51
Table 6-2 Reserved Word List, Cont'd 6-52
Table 6-3 Model Size Limits 6-53
Table 7-1 Excerpt Results from a Sample SUBMAP Call 7-67
Table 7-2 Summary of HEATPIPE Parameters and Their Units 7-133
Table 7-3 Complete Input File for Pipe Frost Accretion 7-176
Table 7-4 Alternate NODE DATA for Arithmetic Ice Stack 7-180
Table 7-5 Illustration of MIN_FIND_I and MIN_FIND Usage with LAGI=5 and LAGN=3 7-212
Table 7-6 Illustration of ROOT_FIND_I, ROOT_FIND Usage with LAGI=4, LAGN=3, XLO ¹ XHI 7-218
Table 7-7 Property Routines 7-273
Table 7-8 Additional Property Routines, Compressible Liquids 7-274
Table A-1 Library Fluids A-15
Table A-2 Range Limits for Standard Library Fluids A-16
Table A-3 REFPROP: Available Fluids and Range Limits A-19
Table A-4 Unit Systems, Assumed Units A-21
Table A-5 FLUINT Solution Algorithm in Pseudo-Code A-24

List of Figures

Figure 1-1 Bar of Metal for Sample Problem 1-5
Figure 1-2 Sample Thermal Network. 1-6
Figure 1-3 Bar of Variable Cross Section 1-10
Figure 1-4 Simplified SINDA/FLUINT Data Flow. 1-16
Figure 1-5 Detailed SINDA/FLUINT Data Flow 1-17
Figure 2-1 Preprocessor Flow Chart 2-11
Figure 2-2 Sample Bivariate Function 2-45
Figure 2-3 Curve of Temperature-Varying Capacitance 2-56
Figure 2-4 Temperature-Varying Heat Rate 2-65
Figure 2-5 Cyclic Heat Source Example 2-68
Figure 2-6 Sample Thermal Network. 2-80
Figure 2-7 Curve of Temperature-Varying Conductance 2-81
Figure 2-8 Example PIV and PIM Network 2-88
Figure 3-1 Hierarchy of FLUINT Network Elements 3-7
Figure 3-2 Examples of Junction Loops 3-23
Figure 3-3 Update Sequence for Tube and STUBE Solutions 3-51
Figure 3-4 : Rounded-edge Reducer (Nozzle) 3-69
Figure 3-5 Basis for Built-in ORIFICE K-factor Correlation (Circular Case Depicted) 3-95
Figure 3-6 Similarity Rules for Dimensional and Nondimensional Pump Curves 3-106
Figure 3-7 Default Pump Degradation Curve for Two-Phase at Inlet 3-113
Figure 3-8 Graphical Examples of Symmetric and Asymmetric Head-Flow Rate Curves 3-125
Figure 3-9 Graphical Example of Default Asymmetric Option for Pressure Ratios 3-126
Figure 3-10 Graphical Examples of Symmetric for Pressure Ratio (HU=201) 3-126
Figure 3-11 Graphical Example of Symmetric for Martin β Factor (HU=202) 3-127
Figure 3-12 Sample Turbine Performance Maps 3-139
Figure 3-13 Sample Compressor Performance Map. 3-162
Figure 3-14 Downstream Discretized Heat Exchanger Sections. 3-204
Figure 3-15 Center Discretized Heat Exchanger Sections 3-205
Figure 3-16 Heat Exchanger Sections Using Segment Ties 3-228
Figure 3-17 Examples of Modeling with a FLAT iface 3-255
Figure 3-18 Deactivization of Redundant ifaces 3-257
Figure 3-19 Bellows Representation of a SPRING iface 3-261
Figure 3-20 Behavioral Modes of a WICK iface with Tank B Containing Vapor . . . 3-264
Figure 3-21 Axial Subdivision of a Depressurising Vessel Using Interfaces 3-274
Figure 3-22 Duct Macro Options (Four Sections Each). 3-290
Figure 3-23 Schematic of FLUINT Sample Model. 3-307
Figure 3-24 Algorithm for Determining Prime/Deprime Status 3-314
Figure 3-25 One Path Duplicated 10 Times 3-318
Figure 3-26 Duplicated Subnetworks 3-319
Figure 3-27 Duplicated Subnetwork with Different End Points 3-319
Figure 3-28 (a) Subnetworks That Cannot Use Duplication Factors 3-320
Figure 3-28 (b) Above Model After Duplication-Enabling Assumption 3-320
Figure 3-29 Symmetry in Thermal Submodels Using One-Way Conductors 3-324

Figure 3-30 Orientation of Sample Loop 3-330
Figure 3-31 Pressures in a Reducer 3-347
Figure 3-32 Pressures in an Expander 3-348
Figure 3-33 A Passage with Uniform Radial Injection/Extraction 3-355
Figure 3-34 Actual vs. Assumed Velocity at the Exit of a Duct Macro 3-358
Figure 3-35 Simplified Two-phase Flow Regimes 3-360
Figure 3-36 Homogeneous vs Slip Flow in Stratified Regime at Same Flow Quality 3-365
Figure 3-37 Vapor and Liquid Spinodals 3-379
Figure 3-38 Spinodals for a Cubic PVT Surface Isotherm 3-380
Figure 3-39 “Frictional” Limit Versus Critical Flow Limit for Choking 3-383
Figure 3-40 Controlling Lines with Plena, Control Valves, and Heater Junctions . . 3-402
Figure 3-41 Property Limits Illustrated for Two Hypothetical Two-Phase Fluids . . . 3-431
Figure 3-42 Lump plus Associated Paths for Modeling Dissolution/Evolution 3-460
Figure 3-43 Twinned (Nonequilibrium) Tanks 3-477
Figure 3-44 A Typical Twinned Tank Network 3-479
Figure 3-45 Temperature Response Given ULT = -1.0 3-498
Figure 3-46 Pressure Response as a Function of ULT 3-498
Figure 3-47 Velocity Angles within the Flow-Tangent Plane (and Radial Flow Example) 3-502
Figure 3-48 Velocity Angles for the Axial Component 3-502
Figure 3-49 Velocity Angle Example: Flow Through a Journal Bearing (Top/Internal View) 3-503
Figure 3-50 Flat-front (Fill or Purge) Discretization Schemes 3-515
Figure 3-51 Axially Stratified Flow Regime for Flat-front Modeling 3-517
Figure 3-52 : Specifying Ports on a Flat-front Segment 3-523
Figure 3-53 : Two Ways to Model a Sight Glass: Flat-front and Port Example 3-525
Figure 4-1 Basic Processor Program Flow 4-10
Figure 4-2 Sample Flow Chart for an OPERATIONS Block 4-11
Figure 4-4 Internal Flow of Subroutine VARBLO 4-12
Figure 4-3 Nested Structure of the Logic Blocks 4-13
Figure 4-5 Flow Chart of Network Solution Routine TRANSIENT (“FWDBCK”) . . . 4-14
Figure 4-7 Flow Chart of Network Solution Routines STDSTL and STEADY (“FASTIC”) 4-16
Figure 4-6 Flow Chart of Network Solution Routine FORWRD 4-17
Figure 4-8 Basic Flow in FORWRD During Each Time Step 4-72
Figure 4-9 Basic Flow in TRANSIENT (“FWDBCK”) During Each Time Step 4-73
Figure 4-10 Basic Flow in STDSTL and STEADY (“FASTIC”) 4-74
Figure 5-1 Interrelationship of Optimization Terms 5-7
Figure 5-2 Flow Chart for SOLVER 5-12
Figure 5-3 Graphical Example of Optimization in Two Dimensions 5-32
Figure 5-4 Graphical Example Showing Importance of RCACTO 5-33
Figure 5-5 Fake Test Data to Correlate Against 5-58
Figure 5-6 Summary of Best Methods 5-63
Figure 5-7 Two Possible Latin Hypercube Samplings of a Two Dimensional Design Space with a Discretization Level of Five 5-67
Figure 5-8 Interrelationship of Reliability Engineering Terms 5-74
Figure 6-1 Basic Input File 6-1
Figure 7-1 GADD Conductor Networks vs. Case Number 7-118

Figure 7-2 Heat Pipe Variations and Typical Network Diagram 7-131
Figure 7-3 Network Changes Invoked by RECESS for Simulating Mass Loss . . . 7-159
Figure 7-4 Frost Thickness versus Time 7-181
Figure 7-5 Geometry for DIFFEQ2 Valve Stem Example 7-202
Figure 7-6 Transverse and Helical Ribs, Crimps, Wires, etc. 7-325
Figure 7-7 Geometry for Subroutine NCFINS 7-344
Figure 7-8 Comparison of FLUINT vs. Method of Characteristics for Waterhammer 7-423
Figure 7-9 TankCalc Input Screen. 7-450
Figure 7-10 TankCalc Compute Screens (two views of the same data) 7-451
Figure 7-11 TankCalc Output as SINDA Array 7-452
Figure 7-12 FloCAD Network Element Logic Example of TankCalc60 Usage 7-452
Figure 7-13 Example Problem for CHX Modeling 7-470
Figure 7-14 Offset Strip Fin with Defining Dimensions 7-473
Figure A-1 SINDAWIN Utility A-35
Figure A-2 Processor Status and Run-time Modifications A-35

1 INTRODUCTION AND OVERVIEW

1.1 How to Use This Manual

SINDA/FLUINT is best described as a general-purpose *language* in which arbitrary thermal/fluid problems may be posed. Like the description of any language, a tabulation of the lexicon without a simultaneous explanation of the rules of grammar risks missing the gestalt. Also like any language, there is a limit to the degree of fluency that can be achieved merely by reading or listening.

Not all concepts, rules, and methods in SINDA/FLUINT can be defined and explained in a linear fashion. Also, few readers start at the beginning of this manual and read it straight through; most use it as a reference source rather than as training material. Therefore, while a general attempt is made to avoid reference to concepts not yet explained, this goal is often sacrificed when it interferes with the usefulness of this manual as a reference. The SINDA novice is therefore advised that several readings may be necessary, and that only by using the code will some of the concepts become clear.

This section provides the new user with background material to help frame an understanding. However, other documents are available that are devoted to introducing new users to SINDA/FLUINT, as described in the preface (page xlvii).

Section 2 describes the basic SINDA (thermal network) input file, focusing on thermal network data as well as providing an introduction to the input file in general terms.

Section 3 describes the FLUINT (flow network analysis) capabilities.

While the above two sections focus on networks and data input, Section 4 describes logical inputs, which govern program execution and network solution. Sections 2 and 4 are required reading for the analyst creating thermal conduction/radiation models. Section 3 is required in addition to those two sections if the model contains fluid flow analyses.

Section 5 introduces higher level analysis operations such as design optimization, goal seeking, and test data correlation using the *Solver*, and statistical design methods using the Reliability Engineering module.

Section 6 is a concise summary of input formats and other useful information. While completely redundant, this section serves as a quick reference guide.

Section 7 documents the program library, a large set of subroutines that augment the user's analysis, and which in many instances will be an integral part of every model (e.g., output routines).

The appendices within this volume are largely devoted to further explanations of FLUINT correlations, properties, and solution methods. Also described are external pre- and post-processors. Sinaps, which renders the post-processing programs obsolete, is documented in a separate manual (see page xlii of the preface).

A separate volume, the Sample Problem Appendix, contains examples intended to help clarify concepts introduced in this volume. Graphical user interfaces are documented separately.

1.2 Learning SINDA/FLUINT

Engineering design tools, and perhaps computer tools in general, seem to represent a spectrum of choices. On one end of the spectrum are tools that are simple to learn and use, but offer limited utility: they must often be abandoned when a completely new design or design problem arises. At the other end of the spectrum are tools that are very general purpose and that are highly customizeable and extensible, but which consequently have many options and take a longer time to learn. SINDA/FLUINT is unashamedly at this latter end of the spectrum. A conscious choice has been made in its design: if the user makes an up-front investment in training, the pay-off is that the user need only learn one tool, and that analyses can be performed of applications that were clearly not foreseen by the authors of the tool.

SINDA/FLUINT takes a toolbox approach. Almost no assumptions are made with respect to the types of system or problem that will be tackled by users, since by making such assumptions its utility will be diminished. There are no “cookbook” methods. *This tool is intended to be used by engineers who must make decisions and assumptions in order to efficiently arrive at an acceptable answer.* The first step, therefore, is to think about the problem at hand and how to phrase an appropriate question to the program, know first what design questions need to be answer. This manual cannot make modeling decisions, it can only present the options available and explain the generic impact of their selection.

The ability of SINDA/FLUINT to be customized to any new application is intrinsic. User logic is not only “accepted” as in some codes, it is required in at least very simple form. This link to Fortran is sometimes a point of frustration to users unaccustomed to Fortran programming. To those users, it should be understood that an arbitrary SINDA-unique language could have been invented that everyone would have had to learn. By using Fortran, which is normally part of most engineering curricula, at least those required to learn rudimentary skills will find other applications for those skills. Many analyses can be performed with such basic skills. However, the ability to guide, augment, or even override the solution procedure has been arguably the single most important explanation for the popularity and longevity of SINDA. Many of the original uses of the Fortran-style, interface, however, have been replaced by the advent of an expression-based internal spreadsheet. In other words, extensive use of the spreadsheet expressions minimizes the need to use extensive Fortran logic.

A set of sample problems is available in a separate volume. These problems are designed not only to lead the reader through a logic sequence of problem definition and resolution, they are designed to illustrate most program options. Therefore, the reader should plan on first scanning this volume (especially Sections 2, 3, and 4), and then carefully reviewing the sample problems. After accruing this general background, the best teacher is experience.

Section 5 describes high-level analysis operations that transcend steady-state and transient analyses that are the subject of Sections 2 through 4. Because Section 5 builds upon a knowledge of basic SINDA/FLUINT operation, it should therefore be learned after acquiring an understanding of the earlier chapters.

Finally, it should be noted that many users of Thermal Desktop®, the CAD-based geometric interface to SINDA/FLUINT, need to know very little SINDA/FLUINT to do most of their tasks. To them, this manual might be perceived as "advanced Thermal Desktop usage." The user is therefore encouraged to consider using that tool as a starting point, perhaps after having learned some of the basics in Sections 1 and 2 of this manual.

1.3 SINDA: A Historic Overview

SINDA is a software system suited for solving lumped parameter representations of physical problems governed by diffusion-type equations. The system was originally designed as a general thermal analyzer that utilizes resistor-capacitor (R-C) network representations of thermal systems (Section 1.5); although, with due attention to units and thermally-oriented peculiarities,* SINDA will accept R-C networks representing other types of systems including finite difference and finite element.

SINDA was originally conceived and written under the name CINDA (Chrysler Integrated Numerical Differencing Analyzer) in the 1960's. Around 1970, the NASA JSC version of the program became SINDA. During the 1970's and into the early 1980's, SINDA was improved by a series of enhancements that include the fluid flow network capability, improved network solution algorithms and the addition of a number of input options.

SINDA/FLUINT, first released as "SINDA '85" in 1985, represented a more extensive change to the program than it had undergone in any one of the previous development phases. The preprocessor was completely rewritten. One of the most significant innovations was the ability to organize models by submodels. Beyond mere organization tools, submodels allow users to combine models without name or logic conflicts, and enable a higher level of network manipulation: submodels can be dynamically added, dropped, or swapped during program execution.

The basic hallmarks of SINDA's design were retained. These include dual execution (Section 1.6): using a preprocessor that accepts the user's definition of the thermal or flow network parameters and converts user supplied "Fortran-like" language into real Fortran. This Fortran is compiled and sent to the system loader. The loader collects the necessary parts of the pre-compiled Fortran subroutines that make up the processor library. Because many of the subroutines are invoked through user-supplied calls, each processor load module is essentially custom-designed by the user for the problem at hand. When the collect/load task is finished the resulting load module is executed. This is the so-called "processor execution."

A pressure/flow analysis of a system containing an arbitrary piping network can be performed simultaneously with the thermal analysis during transient or steady state solutions. This permits the mutual influences of thermal and fluid problems to be included in the analysis. This fluid network analysis capability is called FLUINT. In the 3.0 release in 1994, the ability to handle fluid mixtures instead of just single substances was added to the code. The Version 4.1 and 4.2 releases added complex mixture phenomena such as dissolution, and full nonequilibrium two-phase control vol-

* For example, heat flow by radiation is unique to thermal systems in that it is a function of temperature to the fourth power. By contrast, no flow phenomenon is a function of pressure to the fourth power.

umes. Rotating flow paths and turbomachinery components were added in Version 4.8. Thermodynamically compressible liquids and chemical reactions were added in the 5.0 and 5.1 releases, respectively.

In the 4.0 release in 1997, a new spreadsheet-like capability was introduced along with a top-level method for dealing with complex modeling tasks like design optimization and test data correlation. In Version 4.3, this top-level capability was further extended with statistical design methods intended to measure and improve design reliability. These revolutionary capabilities apply to all parts of the code, whether part of thermal or fluid networks.

In the 5.2 release, a completely new thermal solution was introduced to speed the solution of huge thermal models by orders of magnitude. In the 5.4 release, the program internally converted to all double-precision variables, yet without affecting the user interface (other than rendering certain options no longer necessary). This change opens the path for different numerical schemes and tolerances to enable greater speed and robustness.

Within this manual, SINDA/FLUINT will always be regarded as a thermal and/or fluid network analyzer. The knowledgeable user will, however, recognize the program's ability to solve electrical networks or any other mathematical model that can be represented by a lumped-parameter network.

1.4 Graphical User Interfaces

SINDA/FLUINT was used in a stand-alone fashion for many years, with input files generated in a text editor. While it may still be used in this fashion, almost all new users in the last decade generate models and run SINDA/FLUINT through one of its graphical user interfaces (GUIs): Sinaps® (page xlii in the preface), Thermal Desktop® (page xliii and page xliv in the preface), or a variety of third-party and in-house interfaces. The novice user is strongly encouraged to investigate these more modern methods of using SINDA/FLUINT.

However, **in order to support all SINDA/FLUINT users, this manual is written assuming no GUI is used.** Nonetheless, references to circumstances special to any such software may be mentioned from time to time.

1.5 Introduction to the R-C Network

To introduce the user to the nature of the R-C network input required by SINDA thermal modeling, as well as to illustrate the basic flexibility of the program, a short sample problem, from engineering statement to completed input file, will be presented in the following paragraphs. Note that fluid networks (introduced in Section 3) are fundamentally different from the thermal networks that will be introduced here. Also note that, despite the use of US Customary units in these examples, ***SINDA/FLUINT can employ several unit sets with equal ease.***

The following examples will illustrate the complete process of transforming an engineering problem into a SINDA/FLUINT input file. Only the most elementary features will be employed so that the new user will not be overwhelmed with detail. At the same time, however, a small amount of program logic will be included to expose the new user to the versatility and flexibility that are possible within the SINDA/FLUINT system.

1.5.1 Transiently Heated Bar

Consider a bar of metal as shown in Figure 1-1. It is 0.1 inch thick, 1.0 inch wide, and 3.0 inches long, and is fully insulated except for one end. A 10 watt heater is embedded at the insulated end of the bar, and the uninsulated end radiates to deep space. The bar is initially at a temperature of 70°F. After turning on the heat, it is desired to know the time, to the nearest minute, when the center of the bar reaches a temperature of 200°F, and the temperature distribution in the bar at this time.

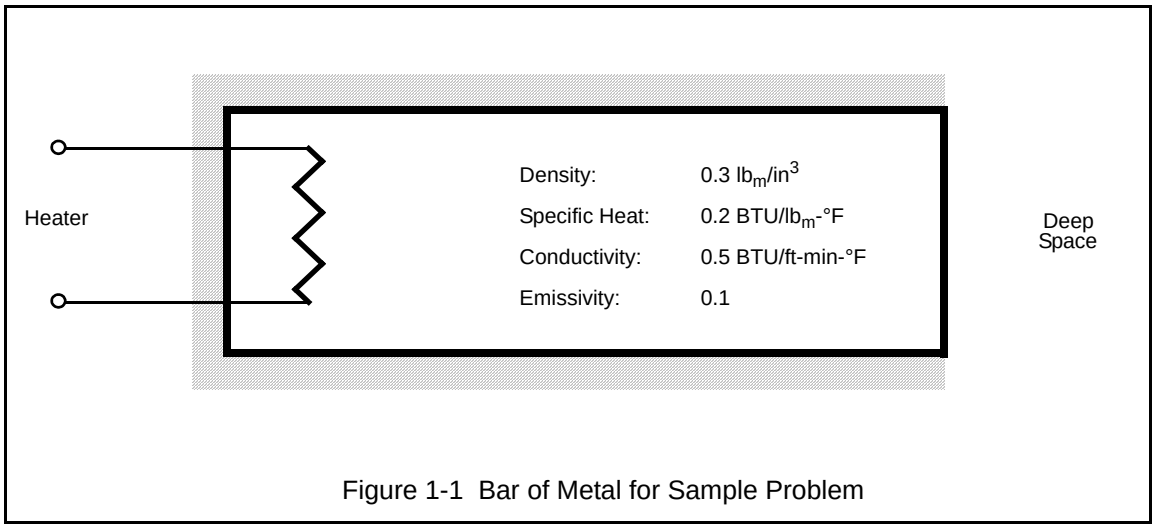


Figure 1-1 Bar of Metal for Sample Problem

Figure 1-2 shows the lumped parameter/R-C network which represents the stated problem. The mass of the bar has been divided into three* portions (shown schematically as capacitors) which are called *nodes*, and each node has been assigned an arbitrary identification number. The heat conduction paths, (shown schematically as resistors) which are called *conductors*, have also been assigned arbitrary identification numbers. A heat *source*, Q, which represents the heater in the insulated end of the bar, is shown entering node 10. Deep space is represented by a constant temperature node at -460°F (shown schematically as a ground).

* The purpose of this short example is to introduce concepts, and not to teach good modeling practices. An odd number of portions was chosen such that one would represent the mass at the center of the bar, which is of particular interest to this problem. Three nodes were selected to keep the demonstration simple, whereas more careful engineering judgment might have resulted in a selection of five or seven nodes. Many other nodalizations are possible and some are likely better (e.g., representing the exposed surface temperature with an arithmetic [massless] node, as will be described later).

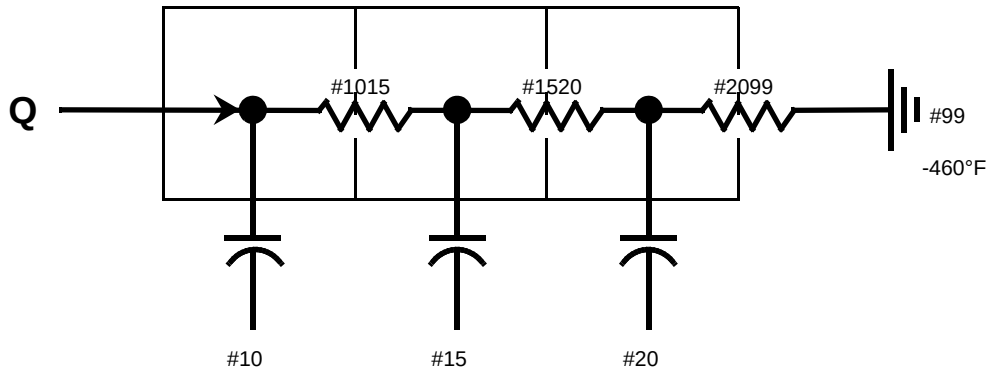


Figure 1-2 Sample Thermal Network

The thermal energy storage capacity (or capacitance) of each bar node is equal to the product of its density, volume, and specific heat (in this case, 0.3*(0.1*1.0*3.0)*0.2/3 = 0.006 BTU/°F). The three bar nodes may be defined by specifying their identification number, initial temperature, and capacitance as follows:

```
10, 70.0, 0.006                    $ record 1
15, 70.0, 0.006                    $ record 2
20, 70.0, 0.006                    $ record 3
```

The node representing deep space may be defined as follows:

```
-99, -460.0, 0.0                   $ record 4
```

The *conductance* of a conductor which represents a heat flow path through a material is equal to the product of the material's thermal conductivity and the cross-sectional area of the flow path, divided by the length of the path. The conductance of bar conductors 1015 and 1520 is, therefore, (0.5/12.0)*(0.1*1.0)/(3.0/3) = 0.00417 BTU/min-°F.

The conductance of a conductor which represents heat flow by radiation is equal to the product of the Stefan-Boltzmann constant and the emissivity, area, and view factor from the surface. In this case, conductor 2099 has a conductance of 0.1*(0.1*1.0)*0.1712e-8/(144*60) = 1.98 x 10^{-15} BTU/min-R^4. In SINDA, the three conductors may be defined by specifying their number, the numbers of their adjoining nodes, and their conductances as follows:

```
1015, 10, 15, 0.00417              $ record 5
1520, 15, 20, 0.00417              $ record 6
-2099, 20, 99, 1.98E-15            $ record 7
```

Since the bar will undoubtedly reach the temperature of interest in less than 1000 minutes and the output results need be accurate only to the nearest minute, the SINDA end time and output time control variables may be set, appropriately, as follows:

```
TIMEND = 1000.0, OUTPUT = 1.0          $ record 8
```

To impress the heat source on node 10, the following record is prepared, where unit conversions into consistent units are written in pseudo-Fortran:

```
10, 10.0*3.413/60.0                    $ record 9
```

Since a transient analysis of the bar is desired, a routine from the SINDA library which performs transient analysis by the Forward-Backward (Crank-Nicholson) finite differencing technique will be called into action, with the following record:

```
CALL TRANSIENT                         $ record 10
```

Finally, to suppress output until the center of the bar reaches 200°F, at which time it is desirable to print the temperatures of the nodes and to end the problem, the following logic statements are required:

```
IF (T15 .GE. 200.0) THEN               $ record 11
  CALL TPRINT('SUB1') $ PRINT TEMPS    $ record 12
  TIMEND = TIMEN $ END PROBLEM NOW     $ record 13
ENDIF                                  $ record 14
```

Table 1-1 shows how the 14 records prepared specifically for the sample problem have been inserted in a "real" input file which contains the additional records necessary to separate, for example, the node data from the conductor data, etc. This list of records constitutes the complete SINDA/FLUINT input required to solve the stated problem.

1.5.2 Transiently Heated Bar Using Spreadsheet Options

With a little forethought, the user should instead have created a section listing important dimensions and properties to be referenced in other parts of the input. For example:

```
HEADER REGISTER DATA
     DENS      = 0.3
     CP        = 0.2
     CON       = 0.5
     EMIS      = 0.1
     THICK     = 0.1
     WIDE      = 1.0
     LONG      = 3.0
     AREA      = THICK*WIDE
     VOLUME    = AREA*LONG
```

Table 1-1 Complete Input File for Sample Problem

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM
      OUTPUT  = BAR.OUT
      MODEL   = TEST
HEADER NODE DATA, SUB1
     10, 70.0, 0.006               $    record 1
     15, 70.0, 0.006               $    record 2
     20, 70.0, 0.006               $    record 3
     -99, -460., 0.0               $    record 4
HEADER CONDUCTOR DATA, SUB1
     1015, 10, 15, 0.00417         $    record 5
     1520, 15, 20, 0.00417         $    record 6
    -2099, 20, 99, 1.98E-15        $    record 7
HEADER CONTROL DATA, GLOBAL
     TIMEND = 1000.0, OUTPUT = 1.0 $    record 8
HEADER SOURCE DATA, SUB1
     10, 10.0*3.413/60.0           $    record 9
HEADER OPERATIONS
BUILD ALL
      CALL TRANSIENT               $    record 10
```

Table 1-2 shows the utility of this approach by showing the previous input file restructured to take advantage of these spreadsheet-like *registers*. In addition to providing centralized control and utilities for fast, consistent changes, registers facilitate model maintenance and improve self-documentation. They also enable rapid parametric and sensitivity analyses, as well as complete design optimization options.

Expressions can be used almost anywhere within SINDA/FLUINT, and they can be arbitrarily complex. They can even refer to model inputs or outputs, and can contain conditional operators. For example, the logic to terminate the run can be placed in the definition of TIMEND (the problem end time):

```
TIMEND  = (sub1.t15 >= 200.0) ? timen : 1000.0
```

1.5.3 Optimizing the Shape of a Heated Bar

If the cross section of the bar were variable at three locations (cross section), as shown in Figure 1-3, then the optimum shape of the bar could be found as a function of some thermal performance criterion. As shown in that figure, the nodalization has changed somewhat: node 1 is located at the edge of the bar and node 3 is located at the opposite edge to take into account the temperature drop across the entire length. Node 2 is located in the middle. The cross sectional thickness at the ends and center of the bar, denoted X_1, X_2, and X_3, are variables to be calculated by the program. X1, X2, and X3 are therefore added as new registers. The masses of the nodes are then calculated as:

```
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
```

Table 1-2 Complete Input File Using Registers

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM, WITH REGISTERS
   OUTPUT = BAR.OUT
   MODEL  = TEST
 HEADER REGISTER DATA
     DENS    = 0.3
     CP      = 0.2
     CON     = 0.5/12.0
     EMIS    = 0.1
     THICK   = 0.1
     WIDE    = 1.0
     LONG    = 3.0
     AREA    = THICK*WIDE
     VOLUME  = AREA*LONG
   HR2MIN  = 60.0
   RESOL   = 3.0
 HEADER NODE DATA, SUB1
   GEN 10,3,5,70.0,(DENS*CP*VOLUME)/RESOL
   -99, -460., 0.0
 HEADER CONDUCTOR DATA, SUB1
   GEN 1015,2,505,10,5,15,5,CON*AREA/(LONG/RESOL)
   -2099, 20, 99, EMIS*AREA*sbcon/(HR2MIN*144.0)
 HEADER CONTROL DATA, GLOBAL
      TIMEND = 1000.0, OUTPUT = 1.0
 HEADER SOURCE DATA, SUB1
      10, 10.0/(btuhrwat*HR2MIN)
 HEADER OPERATIONS
 BUILD ALL
       CALL TRANSIENT
 HEADER OUTPUT CALLS, SUB1
       IF(T15 .GE. 200.0) THEN
           CALL TPRINT('SUB1')
           TIMEND = TIMEN
       ENDIF
 END OF DATA
```

where MPER is a new register defined as the mass per unit thickness, with RESOL being set equal to the current resolution of 3 nodes:

```
MPER    = DENS*WIDE*LONG/RESOL
```

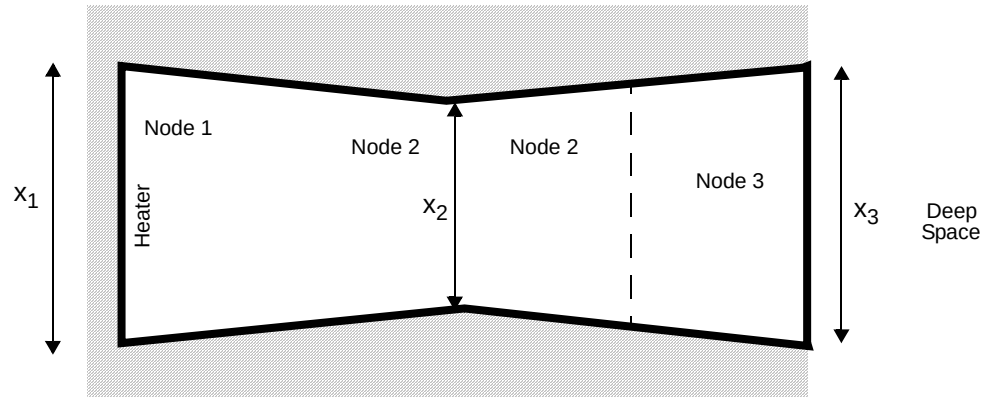


Figure 1-3 Bar of Variable Cross Section

The conductances between node centers can then be input using the formula for heat transfer across a wedge,* and the radiation heat transfer similarly scales with the size of the open-ended face:

```
 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.)
```

A design question is then posed: what is the shape of the bar that minimizes mass while assuring that the temperature of the heater does not exceed 500 degrees under steady state conditions? Furthermore, the cross sectional thickness is not allowed to grow or shrink by more than a factor of 2 between adjacent sections. The declaration of the design variables and the constraints are provides as follows:

```
HEADER DESIGN DATA
       0.0010 <= X1
       0.0011 <= X2 <= 2.0*X1
       0.0012 <= X3 <= 2.0*X2
HEADER CONSTRAINT DATA
       SUB1.T1  <= 500.0
```

* This formula results in an arithmetic fault in the limits of a rectangular bar, as the thickness on each side become equal. Means of dealing with it will be presented in later chapters.

The measurement of the success or failure of each design is then determined by a single steady state solution, with the objective (namely, mass) being updated as part of the procedure by which a design should be evaluated:

```
HEADER PROCEDURE
DEFMOD SUB1
        CALL STEADY
        OBJECT  = (C1+C2+C3)/CP
```

Note that a steady state calculation ignores masses, but that the capacitance of each node presents a convenient measure of the current mass of each design evaluated. In SINDA logic blocks, “C1” refers to the capacitance (mass times specific heat) of node 1.

Table 1-3 presents a complete input listing for the bar optimization problem. SINDA/FLUINT then calculates a minimum mass of 17.9 lb with values of X1=8.5, X2=17 and X3=34. It evaluates 15 different designs (sets of thickness values) during the optimization process.

1.5.4 Determining the Reliability of a Heated Bar

Inputs such as dimensions, properties, dissipations, and environmental conditions are rarely known accurately, or can vary from unit to unit or application to application. Inputs can be specified statistically in SINDA/FLUINT as probability distributions, and then the resulting reliability of the design can be calculated.

In the previous example of the heated bar, assume that the emissivity is not certain: it can assume any value from 0.08 to 0.12 depending on surface treatments, exposure to harsh environments, etc. Furthermore, assume that the width itself assumes a normal (Gaussian) distribution about a mean value of 1.0 due to machining tolerances, with a standard deviation of 0.01. Assume also that the power profile is similarly uncertain, with a standard deviation of 0.5 watts. These are specified to the program using:

```
HEADER RANDOM DATA
        EMIS, UNIFORM, 0.08, 0.12
        WIDE, NORMAL, SD=0.01
        POWER, NORMAL, SD=0.5
```

The procedure to determine the reliability of the design is the same as that used to test the validity of an optimal design, namely a simple steady state:

```
HEADER RELPROCEDURE
        CALL STEADY
```

Table 1-3 Optimizing Bar Shape

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM, USING SOLVER
        OUTPUT          = baropt.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
        X1              = 10
        X2              = 20
        X3              = 30
        LONG            = 3.0
        HR2MIN          = 60.0
        RESOL           = 3.0
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*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, 10.0/(btuhrwat*HR2MIN)
HEADER DESIGN DATA
        0.0010 <= X1
        0.0011 <= X2 <= 2.0*X1
        0.0012 <= X3 <= 2.0*X2
HEADER CONSTRAINT DATA
        SUB1.T1  <= 500.0
HEADER OPERATIONS
BUILD ALL
        NLOOPS         = 100
        CALL SOLVER
        CALL TPRINT('SUB1')
HEADER SOLOUTPUT
        CALL DESTAB
        CALL CSTTAB
HEADER PROCEDURE
DEFMOD SUB1
        CALL STEADY
        OBJECT         = (C1+C2+C3)/CP
END OF DATA
```

For now, assume fixed values of the thicknesses have been selected: X1=8.5, X2=17, and X3=34 (corresponding to the results of the prior optimization). Also assume that the reliability constraint is the same as the optimization constraint. The failure limit is:

```
HEADER RELCONSTRAINT DATA
       SUB1.T1 <= 500.0
```

A descriptive sampling technique, faster than a simple Monte Carlo sampling technique (which is also available), is applied instead of the call for optimization (SOLVER). That, plus some outputs, results in the following in OPERATIONS:

```
CALL DSAMPLE
CALL RANTAB
CALL RCSTTAB
```

A limit of 100 samples (steady state solutions) will be made by default. The final input file is listed in Table 1-4. The results show the bar has only about 50% chance of keeping the heater below 500 degrees. (The actual answers range from 48% to 52% depending on the run.)

1.5.5 Designing a Reliable Heated Bar: Robust Design

The reason the reliability in the above design is so low is that the design was about the same as that yielded by the optimization. More specifically, an optimization constraint of SUB1.T1<500 resulted in an optimized design rested right on that constraint. When an *optimization constraint* is directly reused as a *reliability constraint*, about half the time a perturbation in inputs will cause the design to fall outside of the failure limits.

One alternative is to first optimize the design using a tighter constraint, say SUB1.T1<480, such that it stands a greater chance of not exceeding 500 degrees when inputs are uncertain. But there is no basis for selecting the “480” in the above constraint: the margin applied to the design is artificially selected and it may be too much or too little. (As will be shown below, “480” turns out to be much too little.) Iterative optimizations and reliability estimations might yield a better estimate, but such an approach is excessively cumbersome.

Alternatively, the above two runs (Sections 1.5.3 and 1.5.4) can be combined into a single run such that the optimization takes into account not just a performance constraint but also a reliability constraint: make sure that the optimization returns a design whose reliability is at least 99%. This will of course be a heavier but more robust design.

The optimization constraints become:*

```
HEADER CONSTRAINT DATA
       SUB1.T1  <= 500.0
       0.99 <= REL
```

* The optimization constraint of “SUB.T1 <= 500.0” is now largely meaningless because the reliability constraint will now dominate, but it doesn’t hurt to leave it to be sure this is true.

Table 1-4 Determining Bar Reliability

```
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,NORMAL,SD=0.5
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
        CALL DSMAPLE
        CALL RANTAB
        CALL RCSTTAB
        CALL TPRINT('SUB1')
HEADER RELPROCEDURE
        CALL STEADY
END OF DATA
```

The new variable REL is the reliability of each design (set of values for X1, X2, and X3) attempted by the optimizer, as returned by a special routine (RCGET or RCGETNO). DSAMPLE moves from OPERATIONS to PROCEDURE (and SOLVER assumes its original position from Section 1.5.3).

In other words, *a statistical sampling is performed as part of the evaluation of each design*. Each design attempted is assessed not only for its thermal performance, but also for its anticipated reliability.

However, this might mean 30 different designs are sampled 100 times each, for a total of perhaps 3000 steady state solutions. This would be too expensive for a realistic problem, so a short cut is available instead: a fast approximation technique is used (replacing DSAMPLE with RELEST) to estimate reliability in only 4 steady state solutions, resulting in 120 solutions in this case. This number can be further reduced to between 90 solutions using advanced techniques described in later chapters (Section 5.27).

Using RELEST to evaluate the design yields the following changes to the inputs:

```
HEADER OPERATIONS
BUILD ALL
        NLOOPS  = 100
        CALL SOLVER
        CALL RANTAB
        CALL RCSTTAB
HEADER PROCEDURE
DEFMOD1 SUB1
        CALL STEADY
        OBJECT  = (C1+C2+C3)/CP
        CALL RELEST           $ ESTIMATE RELIABILITY
        CALL RCGETNO(1,x,x,x,x,x,x,REL)
HEADER RELPROCEDURE
        CALL STEADY
```

(The above “x” values are dummy variables since they represent data that are not used in this case.)

Because the optimization is more severely constrained by imposing an additional reliability requirement, the resulting design is much heavier than before. In other words, to avoid more than 1% chance of failing the temperature limit, the mean heater temperature is now only about 350 degrees. The final dimensions are X1=15.5, X2=30.6, and X3=60.5, for a total weight of 32.6 lb.

To summarize, optimization will often yield a design that lies on a constraint. If conditions are variable, then approximately 50% of the time that constraint will be violated. A robust design takes this into account, pulling back farther from the failure limit to assure reliability.

1.6 System Structure: Preprocessing vs. Processing

It should be evident that the SINDA/FLUINT system is a complex applications program. It has, in fact, all of the functions and capabilities of a special purpose computer language. Since most computers in current use in engineering environments already have a Fortran compiler, the SINDA/FLUINT system is designed to augment the existing Fortran system. Hence, the SINDA/FLUINT library serves as an extension to the existing Fortran library, and the program serves as a preprocessor to (i.e., it precedes) the existing Fortran compiler. This augmentation arrangement is illustrated in Figure 1-4.

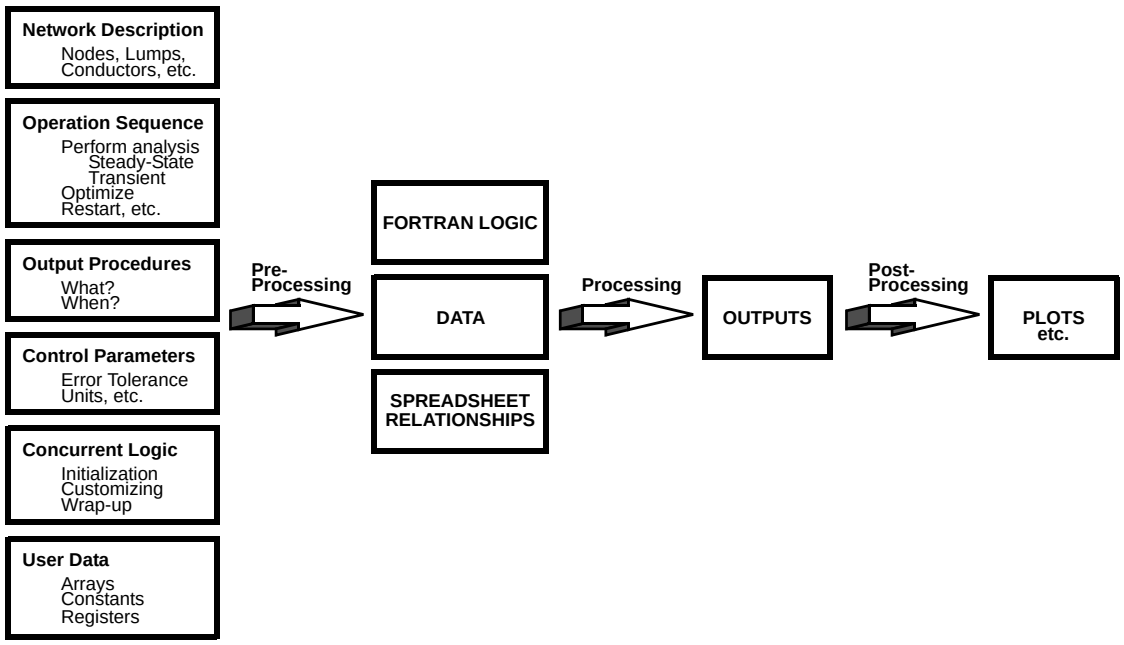


Figure 1-4 Simplified SINDA/FLUINT Data Flow

A more detailed representation is seen in Figure 1-5. SINDA/FLUINT consists of two modules: an executable preprocessor and a library of processor routines. When a run is made, the preprocessor sorts through the user's inputs and generates both data files and Fortran source code files. The Fortran files are compiled and linked with the processor library, forming a dynamically dimensioned and customized executable program for every analysis. When the newly formed program is run, it immediately reads the data files left by the processor and begins to execute the user's instructions.

The distinction between the preprocessor actions and processor actions are important to understand. The preprocessor is a one-time pass through the input file, operating on all blocks but reading in particular *data blocks*. The preprocessor evaluates and perhaps translates *logic blocks*, but does not execute them.

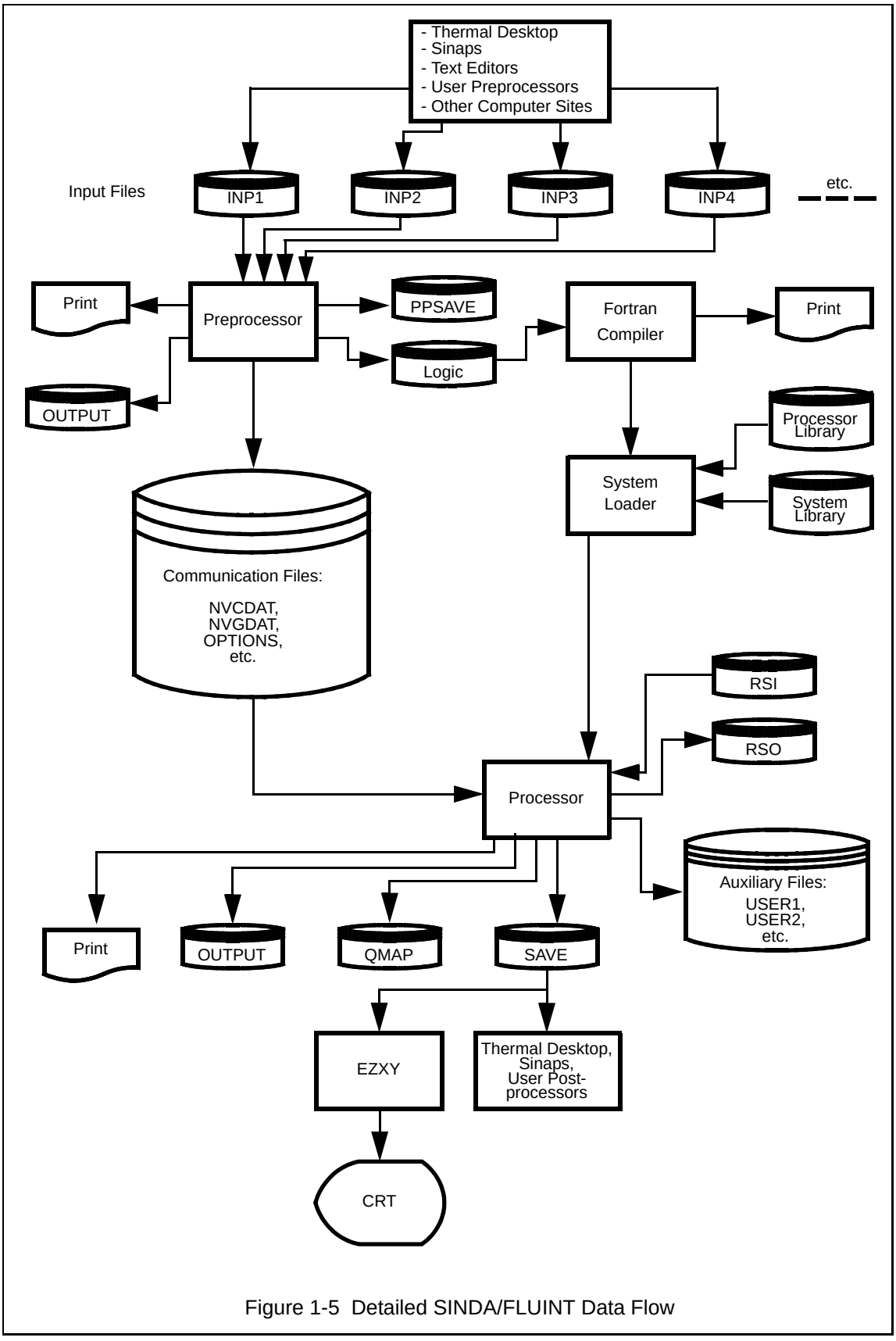


Figure 1-5 Detailed SINDA/FLUINT Data Flow

The processor is executed once, but can consist of any task the user assigns, including repetitive tasks. Even if a single solution routine (steady state or transient) is invoked, the logic blocks used by that routine may be executed many times. Operations performed by the preprocessor are therefore often referred to as “static,” while operations performed by the processor are often referred to as “dynamic.”

Interrelationships between inputs and perhaps between inputs and outputs may be defined using expressions, perhaps using user-named registers as was illustrated in Section 1.5.2. During processor execution, these interrelationships will be resolved in spreadsheet fashion. These updates are performed automatically and frequently by default, but the user can control both the frequency and the content of the update if necessary. Extensive use of this spreadsheet feature eliminates the need for most Fortran-style manipulations in logic blocks.

1.7 Prerequisites and Conventions

1.7.1 Prerequisite Knowledge

In addition to a knowledge of heat transfer and fluid flow, the potential SINDA user will require some familiarity with computers. Specifically, the user is assumed in this manual to have a working knowledge of the following topics:

- Computers
- File Types
- Fortran (briefly)
- Text Editor (e.g., notepad, vi, emacs), or Sinaps

These topics will be referred to but not specifically explained herein. Users whose background includes an introductory course in digital computers and Fortran programming should be familiar with this list. Other users are referred to textbooks on these topics.

1.7.2 Input Records

Input files are normally generated using a text editor that is part of the computer’s collection of “built-in” software (its operating system). In this manual, each line of input will be referred to as a *record*. A record consists of a maximum of 132 characters, including blanks.

While older versions of SINDA required column-dependent input, in SINDA/FLUINT column dependence can be expressed by a single rule: column 1 is reserved for specific key characters and columns 2 through 132 normally constitute a free field. This rule will appear throughout the sections on input definition. The only exceptions to this rule are that (1) in logic blocks columns 1 and 7 have the same significance as in any Fortran code, and (2) a few special characters require two columns to be unique, and therefore require both columns 1 and 2.

In *thermal* data blocks defining nodes and conductors, lines which are too long may be continued with a back slash (\). This \ must be the last character of the line to be continued. Thus, the following represents *one* input record:

```
first input   input \
input         input \
input         last input
```

Back slashes are neither required *nor permitted* in flow data and fluid property blocks, where a subblock input structure is used and inputs are expected to continue over many lines. Neither do back slashes apply to logic blocks, in which the Fortran 6th column rule applies.

1.7.3 Format Definition Rules

This section describes the conventions that will be used when input data are indicated.

First, all input examples will be distinguished from normal text by using a larger typewriter font such as "`FONT`" and "`font`." Furthermore, capital and lower case type will assume a special significance.* When an input record is defined in capital letters, the characters shown must appear in the record exactly as shown. For example, a record might be shown as follows:

```
NAME, RANK, SERIAL NUMBER, Q
```

The actual input record, correctly prepared, would then appear as follows:

```
NAME, RANK, SERIAL NUMBER, Q
```

On the other hand, lower case type is used to indicate a variable input: the user must supply something which corresponds to the item shown in lower case type. For example, an input may be specified as follows:

```
name, rank, serial number, q
```

where: q = length of service, (1 = 1 to 3 years,
2 = 4 to 6 years,
3 = greater than 6 years)

Note that name, rank and serial number were considered self explanatory whereas q was not. The record prepared according to this format might then appear as follows:

```
WILLIAM E. PISKE, CAPTAIN, A03096690, 1
```

The two above examples of input records appeared in the text as non-indented lines. The significance of this is that such a record must begin in column 1, but no other column dependence need be observed. A familiar example of such a record is a comment in a Fortran program. Such comments are also allowed in SINDA/FLUINT.

* This section describes the rules by which valid formats will be presented in this manual. With a few exceptions, SINDA/FLUINT inputs themselves are case insensitive.

If the record appears indented, then it must begin to the right of column 1:

```
      NAME, RANK, SERIAL NUMBER, Q
```

When it is appropriate to identify a particular record format by placing a descriptive comment on the same printed line, this comment will always appear to the right of a dollar sign, $, as illustrated below:

```
ML, name, address      $(mailing list record format)
```

1.7.4 Fortran Control Statements

The purpose of this section is to familiarize the user with the basic types of Fortran statements which are available for implementing program flow control. No attempt will be made to instruct the user in the formal details of utilizing such statements correctly, since such details may be found in any Fortran text or compiler help system. The following list shows the types of Fortran statements which may be used* to effect program control and gives an example of each:

Statement Type	Example	
	1	7
1. CONTINUE	450	CONTINUE
2. unconditional GO TO		GO TO 350
3. computed GO TO	160	GO TO (200,250,300) M
4. assigned GO TO		GO TO N
5. arithmetic IF	200	IF (ITEST+KTEST) 250,300,400
6. logical IF		IF (RTEST.LE.STEST) GOTO 2
7. DO loop		DO 450 M=1,3

The CONTINUE statement causes no action to occur, but is useful as a dummy statement to which a statement number may be assigned. CONTINUE statements are often used to set up a skeleton framework of statement numbers for program control.

The unconditional GO TO statement is used to transfer control directly to the statement prefixed by a certain statement number. This statement number must be known and specified at compile-time and may not be changed at run-time.

The computed GO TO statement permits control to be transferred to the statement prefixed by the Nth statement number in a list of statement numbers, where N is supplied as an integer variable.

The assigned GO TO statement is similar to the unconditional GO TO except that the statement number to which control will be transferred is specified as a variable. This variable is associated with a specific statement number by using an ASSIGN statement.

The arithmetic IF statement is used to transfer control to one of three specified statement numbers, depending on whether a given arithmetic expression is negative, zero, or positive.

* Fortran OPEN statements should not be used: see USRFIL routine instead (Section 7.4.17).

The logical IF statement is used to cause the execution of a given statement if, and only if, a logical expression is true. Logical operators include .NOT., .OR., and .AND. and may operate on logical expressions or variables. Arithmetic expressions connected by relational operators are logical expressions. The following relational operators may be used:

.EQ.	equal to
.NE.	not equal to
.LT.	less than
.GT.	greater than
.LE.	less than or equal to
.GE.	greater than or equal to

The DO loop may be used to cause the execution of a group of statements to be repeated for successively incremented values of a so-called “index variable.”

The statements used as examples thus far in this section are true Fortran statements, using no SINDA/FLUINT parameters. In such statements, a “F” can be placed in column 1 to prevent SINDA/FLUINT from preprocessing or translating that line of code. By default, however, SINDA/FLUINT will scan each line for translatable parameters. For instance, a variant of Example 3 might be:

```
1       7
160     GO TO (200,250,300) YSIDE.K180
```

Here, the simple variable M is replaced by integer user constant number 180 in submodel YSIDE. Since an F does not appear in column 1, the preprocessor will translate the user constant reference into an internal constant number (e.g. 3) and re-write the statement as follows:

```
160     GO TO (200, 250, 300), K(3)
```

Note that the translation only applied to the user constant reference. The remainder of the statement had to conform to Fortran syntax. Lines which must be translated are sometimes referred to in the manual as “translatable statements.”*

The DO loop in Example 7, could also reference a user constant, that is:

```
        7
        DO   450 M = 1, OTHER.K140
```

1.7.5 Terms and Data Conventions

The word *subroutine* is generally used interchangeably with the word *routine*. *Integer* and *fixed point* mean the same thing, as do *real* and *floating point*. *Character* applies to ASCII character strings. The term *data value*, or *literal*, will be taken to mean one element of the set of all integers, floating point numbers, and multi-character strings.

* In the past they were called “M-type” statements because an “M” can be placed in column 1 to signal translation. This “M” is not required and is supported only to be compatible with old versions of SINDA.

Computers encode integer and real and character variables differently. When a computer looks at a string of ones and zeros, it must be told how to interpret them: as integers, as reals, as double precision reals, or as characters. Most software and compilers shield the user from such concerns, but the Fortran upon which SINDA/FLUINT is based does not: it is a weakly-typed language that forces the user to pay attention to how the original data was stored and therefore how it can be accessed. With one important exception,* the user need not be concerned with integer vs. real in SINDA/FLUINT *data* blocks. However, because *logic* blocks are translated into real Fortran and then compiled, users must be careful about references to variables in those blocks. As of Version 5.3, a compiler error will result if either the wrong number or type of arguments is used in a call to a published (SINDA/FLUINT) subroutine or function in user logic.

Integers will be shown in print as a sequence of digits preceded, optionally, by a plus or minus sign. Floating point numbers will appear in print as a sequence of digits with a leading, trailing, or embedded decimal point, prefixed, optionally, by a plus or minus sign, and suffixed, optionally, by an exponent (to the base 10) denoted as the letter E followed by an integer. All real and integer variables are stored in 64 bit (8 byte) precision: ‘double precision’ is redundant.

Character strings will be delimited in print by apostrophes (’), also called *single quotes*. These are necessary because blanks are valid characters and have a specific binary code (i.e., they do not appear on the printed page, but they do appear explicitly in the computer). Examples follow.

INTEGERS:

```
1                       +12345
-1                      -12345
12345                   15
```

FLOATING POINT (SINGLE AND DOUBLE PRECISION) NUMBERS:

```
1.0                     1.5E+6
 10.                    +1.5D6
.10                     -.18E-12
+.15                    -20.E-4
15.                     1467.812
```

CHARACTER STRINGS:

```
’1’                     ’-1’
’ABC’                   ’1.0’
’DOGS AND CATS’         ’(+-+$)’
’   ’                   ’12.6’
```

* The exception is ARRAY DATA (Section 2.11), in which the user must exercise extreme care in either omitting or including the decimal point as required, and note that any expression is treated as a real input.

In contrast to *data values*, another entity, called an *identifier* or *variable*, will be used (in a programming sense). For example, consider the following statement:

```
PI = 3.14
```

In this case, 3.14 is a floating point data value, and PI is an identifier. Note that PI is different from 'PI' which is a character string.

The user must recognize that the four types of data—integer, floating point, double precision, and character—are quite different from the standpoint of the computer. To make this point clear, consider the following addition problem:

```
  100
 +100.0
'+100.0'
```

The answer, in human terms, is three hundred. However, a computer could not perform such a simple problem directly because its internal binary number system maintains many different modes of data representation. It has been the intent of this section to impress upon the reader that some of the same pitfalls awaiting the inexperienced Fortran programmer also await the SINDA/FLUINT user.

2 SINDA INPUT FILE

This section introduces the basic SINDA input file, describing the general structure and detailing input blocks that are common to most if not all input files. *This section does not detail all available inputs, nor even all those inputs required as a minimum.* Rather, it is designed as a prerequisite to the fluid flow modeling capabilities described in the next section and to the optimization and correlation features detailed in Section 5, and as a companion to the execution, control, and logic block descriptions found in Section 4.

2.1 Introduction to the Input File

The SINDA/FLUINT input file is the means by which users specify the structure of their models and the method for solving them. The input file is subdivided into *blocks*, of which there are two types. *Data blocks* are used to pass numerical values to the program, and the input formats for such blocks are specific to the type of block being used. *Logic blocks* are used to pass executable instructions to the program in the form of a Fortran-like language. In fact, logic blocks are translated into real Fortran and then compiled and executed. With one exception noted below, blocks may occur in any order within the input file. The start of a new block is designated by the word “HEADER” starting in column 1, and the words after the HEADER command describe the type of block about to be input (Section 2.4).

To specify the structure of the thermal mathematical model, three data blocks called NODE DATA, SOURCE DATA and CONDUCTOR DATA are provided. When fluid flow is involved, FLOW DATA blocks can be added (in which case the thermal model is optional), as described in Section 3, and working fluid properties may be input in FPROP DATA blocks.

Two fundamental kinds of program control are provided. One type of control uses program variables to control the operation of a pre-programmed SINDA/FLUINT module such as the preprocessor or a processor solution, math utility, or output routine. This parameter-style control is the domain of the OPTIONS DATA block, which is used primarily for preprocessor control, and the CONTROL DATA blocks, which are used primarily for processor control. An additional SOLVER DATA block controls higher-level analysis operations invoked by the Solver routine (Section 5).

The second and most generalized type of program control is done using either spreadsheet-like expressions or using the logic blocks: OPERATIONS, VARIABLES O, VARIABLES 1, VARIABLES 2, FLOGIC 0, FLOGIC 1, FLOGIC 2, OUTPUT CALLS, and SUBROUTINES. These blocks, detailed in Section 4.1, are all written in a SINDA-specialized variant of the Fortran language and provide users with the capability to initialize parameters, specify their choice of solution method and then intervene by changing parameters at will during the solution process. Additional logic blocks used by the Solver and Reliability Engineering modules described in Section 5 include PROCEDURE, RELPROCEDURE, SOLOGIC 0, SOLOGIC 1, SOLOGIC 2, and SOLOUTPUT CALLS.

In addition to model descriptions and program control, another type of input block is provided for the user to enter the data needed to solve the problem. These data consist of single values and arrays of values which can be integers, floating point numbers, expressions, or character strings. The blocks used for this data are REGISTER DATA, USER DATA, ARRAY DATA, CARRAY DATA, FPROP DATA, and MIXTURE DATA.

USER DATA are single floating point or integer numbers declared by the user for later use in logic blocks. In modern usage, REGISTER DATA should be used instead of USER DATA since variables defined in that section can be used in input expressions elsewhere, in addition to being accessible in logic blocks. ARRAY DATA contains arrays of floating point or integer numbers, such as temperature-varying property tables or time-dependent environments. CARRAY DATA are used to store strings of characters. FPROP DATA blocks are used to input alternate fluid property data for fluid submodels. MIXTURE DATA blocks define interactions between working fluids used together within mixtures.

The CARRAY DATA block is reserved for the user to enter character strings of up to 1000 characters in length. This data is useful for output headings, plot titles, file names, and the like. Unlike previous versions of SINDA, there is no machine-dependent size limit on these character strings. The preprocessor automatically reserves the necessary space in core for these strings.

2.2 Input File Structure

The SINDA/FLUINT input file consists of any number of input blocks of the types introduced in the preceding section. They are separated by *header records*. The SINDA/FLUINT header records may contain arguments, so they are really macroinstructions to the preprocessor rather than simply dividers.

There are two blocks which must appear once, and only once, in an input file. These are OPTIONS DATA and OPERATIONS. All the other blocks are optional. **The input file must begin with the OPTIONS DATA block.** It *may* contain one SUBROUTINES, PROCEDURE, RELPROCEDURE, SOLVER DATA, SOLOGIC 0, SOLOGIC 1, SOLOGIC 2, SOLOUTPUT CALLS, and RELOUTPUT CALLS blocks, and one or more REGISTER DATA blocks.

A SINDA/FLUINT model may consist of up to 999 thermal submodels and 999 fluid submodels. This means that the input file may contain up to 999 each NODE DATA, SOURCE DATA, CONDUCTOR DATA, VARIABLES 0, VARIABLES 1 and VARIABLES 2 blocks; up to 999 each FLOW DATA, FLOGIC 0, FLOGIC 1, and FLOGIC 2 blocks; up to 30 FPROP DATA blocks; up to 125 each CONTROL DATA, USER DATA, ARRAY DATA, and OUTPUT CALLS blocks. There is no ambiguity for multiple blocks of any one type because each data value is linked with the submodel name that appears on the header record.

Submodel names may be any *unique* character alphanumeric string of up to 32 characters. They must start with at least one alphabetic character (A through Z). In this case, “unique” precludes the use of reserved SINDA or FLUINT words (such as “T” and “ABSZRO” and “GLOBAL”), and no two submodels may share the same name, even if one is a thermal submodel and the other is a fluid submodel. This restriction also applies to any USER DATA or translated Fortran variable used in

logic blocks (Section 4.1 and Section 6.18.5).

2.3 Input Block to Manual Section Cross-Reference

Descriptions of the SINDA/FLUINT input blocks are contained within the following sections of this manual:

Section 2. . . . . OPTIONS DATA
REGISTER DATA
USER DATA, GLOBAL (“named”)
USER DATA, smn (“numbered”)
ARRAY DATA
CARRAY DATA
NODE DATA
SOURCE DATA
CONDUCTOR DATA

Section 3. . . . . FLOW DATA
FPROP DATA
MIXTURE DATA

Section 4. . . . . OPERATIONS
CONTROL DATA
VARIABLES 0/1/2
FLOGIC 0/1/2
OUTPUT CALLS
SUBROUTINES

Section 5. . . . . SOLVER DATA
PROCEDURE
DESIGN DATA
CONSTRAINT DATA
SOLOGIC 0/1/2
SOLOUTPUT CALLS
RANDOM DATA
RELCONSTRAINT DATA
RELPROCEDURE
RELOUTPUT CALLS

2.4 HEADER Records: Overview

These records are required to separate the input file into blocks. They may also contain multiplying factor data values and other block-specific inputs where appropriate. Header record arguments apply to all the following input records until another header record is encountered or the input file ends. The majority of header records have the form:

```
HEADER type {,smn} [options]
```

where the "H" must begin in column 1, `smn` is the submodel name (which may or may not be required for that type of block), and `type` is one of the following:

```
OPTIONS DATA
REGISTER DATA
CONTROL DATA
USER DATA
ARRAY DATA
CARRAY DATA

NODE DATA
SOURCE DATA
CONDUCTOR DATA

FLOW DATA
FPROP DATA
MIXTURE DATA

OPERATIONS
SUBROUTINES

SOLVER DATA
DESIGN DATA
CONSTRAINT DATA
PROCEDURE
SOLOUTPUT CALLS
SOLOGIC 0
SOLOGIC 1
SOLOGIC 2
RANDOM DATA
RELCONSTRAINT DATA
RELPROCEDURE
RELOUTPUT CALLS
```

Square brackets [ ] indicate optional, defaulted arguments; braces { } indicate options that may or may not be required, depending on other inputs. The submodel name *cannot* be used with OPTIONS DATA, OPERATIONS, SUBROUTINES, REGISTER DATA, PROCEDURE, RELPROCEDURE, SOLVER DATA, SOLOUTPUT CALLS, RELOUTPUT CALLS, SOLOGIC 0/1/2, DESIGN DATA, CONSTRAINT DATA, RANDOM DATA, or RELCONSTRAINT DATA because these blocks only appear once. Nor can it be used with FPROP DATA or MIXTURE DATA, which may appear several times but are not directly associated with any submodel. For other data types the submodel name *is* required.

Logic blocks, which take the format of Fortran-style inputs, omit the word “DATA.” Except for Solver blocks such as SOLOUTPUT CALLS and SOLOGIC 0/1/2, these blocks always require a submodel designation:

```
HEADER type,smn
```

where `type` is:

```
OUTPUT CALLS
VARIABLES 0
VARIABLES 1
VARIABLES 2
FLOGIC 0
FLOGIC 1
FLOGIC 2
```

2.5 INSERT and INCLUDE Directives

The INSERT and INCLUDE directives fetch all or part of another file and place it in the current input file.

The INCLUDE directive is being replaced with the INSERT directive. New users should use only the INSERT directive, and previous users should convert INCLUDEs to INSERTs. The INCLUDE directive is documented in this section as an aid in converting older models.

INCLUDE files cannot be nested (i.e., INCLUDE commands cannot be placed inside of INCLUDEd files), **whereas INSERT files *can* be indefinitely nested** (i.e., INSERT commands *can* be placed inside of INSERTed files). Only INSERT commands can exploit the “paths.txt” feature (Section 2.5.2).

One subtle rule of operation for the INCLUDE directive is that, upon returning from an INCLUDE operation, the program assumes it is still inside the same HEADER block in which the INCLUDE was encountered. This rule, consistent with subroutine-style operation, may cause problems when HEADER blocks are placed inside the INCLUDE file and/or a header record does not immediately follow an INCLUDE directive.

The INSERT directive does not follow this potentially confusing convention. Rather, the INSERTed data is simply collected into a single large file first, and *then* the preprocessing operations begin.

2.5.1 Format and Examples

The general form of the INSERT or INCLUDE directive is as follows:

```
INSERT pfn [,1n1 [:1n2]]
INCLUDE pfn [,1n1 [:1n2]]
```

where the “I” must begin in column 1 and:

```
pfn.....a valid file name (including pathname if needed; see “paths.txt” below)
lnx.....line number range in pfn to include (all of pfn by default)
[ ].....indicates optional input
```

Note that dollar sign style comments cannot be used with INCLUDEs or INSERTs.

As an example of data being inserted into an existing header block, assume that all radiation conductors have been stored in a file called RADK.K. An INSERT directive inside of CONDUCTOR DATA (for example) would add the conductors described in RADK.K to the rest of the block:

```
C EXAMPLE 1
INSERT RADK.K
```

The INCLUDE and INSERT directives can be used to fetch selected portions of a file. The input option “[,ln1 [:1n2]]” specifies the lines to be fetched. If no lines are specified the entire file is read (Examples 1 and 2). If only one number is specified, it is assumed to mean the number of lines to fetch *starting at the beginning of the file* (Example 3). If both line numbers are specified (Example 4), they are assumed to define a range of lines to be fetched (e.g., fetch from line ln1 to ln2, inclusive).

```
C EXAMPLE 2
INSERT COND
C EXAMPLE 3
INSERT COND,10
C EXAMPLE 4
INSERT COND,10:20
```

2.5.2 The paths.txt File

The INSERT command will first try to locate the named file in the current model directory (or relative to that directory) if no path information is given. If it doesn’t find the file there, it will next search the first 20 *absolute* paths listed in a file named “paths.txt” if such a file is found in the current model directory. If a local paths.txt file is not found, or if the file exists but the file named on the INSERT command is not found in the listed directories, then INSERT will next search the /bin directory of the SINDA/FLUINT install location for a global “paths.txt” file and search within it for possible file locations. The first file found that matches the correct name will be inserted, and subsequent matches (i.e., duplicate files of the same name, but in different locations) will be ignored.

Specifically, when searching for an INSERT file for which no path information has been provided (e.g., “INSERT myfile.txt”), the potential file locations is searched in the following order:

1. the local subdirectory (containing the model input file)
2. the absolute directories listed in a file named *paths.txt*, if this file is found in the local model directory

3. the absolute directories listed in a file named *paths.txt*, if this file is found in the /bin subdirectory of the install location

Up to two paths.txt files may exist for each run: a local one and a global one.

The paths.txt files are useful for listing subdirectories containing commonly inserted files, such as those containing material or fluid property information. Because of the potential for the wrong file to be inserted (e.g., an old version left in a forgotten location), it is strongly recommended that only a few such file storage locations be used as central repositories for commonly inserted files.

These files should be saved with text editors (e.g., notepad), and *not* with word processors (e.g., MS Word™). The contents of a paths.txt file might contain the following, as an example (for a PC):

```
C:/My Fluid Props
D:/Program Files/CRTech/Include Files
```

2.5.3 Suggested Uses and Restrictions

Probably the best mode of operation is to use the INCLUDE or INSERT directive to bring information into existing headers. In this mode, large blocks of data (usually conductor or tie data) that make the input file hard to handle can be kept separate while still keeping the structure of the input file clear to a reader.

Alternatively, the INCLUDE or INSERT file can wholly contain one or more header blocks. This is the recommended mode for FPROP DATA blocks.

Caution—INCLUDE directives function somewhat like Fortran subroutine calls; the program does not just blindly collect all files and *then* begin to preprocess the entire input deck (which, conversely, is exactly how the INSERT direction functions). Rather, upon returning from an INCLUDE operation, *the program assumes it is still inside the same header block it was in when the INCLUDE was encountered.* This may cause problems if HEADER records are contained within an INCLUDE file, but a new HEADER is not encountered immediately after returning from an INCLUDE operation. Use the INSERT directive instead if possible.

Restriction—INCLUDE directives cannot be nested (i.e., an INCLUDE file cannot contain another INCLUDE directive to a third file), whereas INSERT commands *can* be nested as long as no infinite loops are created (i.e, an INSERTed file contains an INSERT command referring to the first file).

Caution—Relative path names of files in INSERT commands that encountered within other INSERTed files are treated relative to the original model location, not relative to the current location of the INSERT file. If this is a problem, then the INSERT commands contained within INSERTed files should refer to an absolute path name instead of a relative pathname. The paths listed in “paths.txt” files should always contain absolute path names.

2.6 Miscellaneous Macroinstructions

Macroinstruction records all begin with a key character in Column 1. These records are few in number and powerful in function and should be well understood before attempting to generate SINDA/FLUINT input.

The following is a list of the some of the available macroinstructions records. (Macroinstructions specific to logic blocks are described in Section 4.1.1.) The character in parentheses must be in column 1.

(C)OMMENT	Records
(R)EM	Records
(P)START	Records
(P)STOP	Records
(F)AC	Records

The following subsections present the format and function of these records.

2.6.1 Comments

Records beginning with a C or R in column 1 allow the user to arbitrarily insert comment data into the input file, as in Fortran code. Additionally, a dollar sign, $, signals the end of data on any line, allowing comments to be inserted afterward. However, the dollar sign should not be used on INSERT nor INCLUDE lines nor on OPTION DATA file name designation lines, otherwise it will be added into the file name. Also, the dollar sign should not be used on untranslated logic lines unless the compiler also recognizes this symbol as an in-line comment.

2.6.2 PSTART and PSTOP Records

These records stop and start the echoing and saving of the collected and sorted input file. The printout and save begins by default until a PSTOP is encountered, and a P- instruction stays in effect until another P- instruction is encountered. The various input printout and save options are explained in the OPTIONS DATA, Section 2.7. These instructions have no arguments.

Internally, the program only checks the input file for the character “P” in the first column, and toggles the output mode on and off. Therefore, the full terms “PSTOP” and “PSTART,” while recommended for clarity, are actually equivalent to each other.

2.6.3 FAC Record

One of the options in the NODE and CONDUCTOR DATA header record is `[,fac]`, a multiplier for capacitance and conductance data that is normally used to convert units. A FAC record allows this factor (which defaults to 1.0 upon entering NODE and CONDUCTOR DATA) to be changed within these HEADER blocks. The factor named in the FAC record will then be applied to *all* subsequent nodal capacitance or conductor data. Any number of FAC records may be used within these blocks, but they are not valid in any other block.

The format is:

```
FAC  dv
```

where dv is a positive floating point number or expression.

Expressions used in FAC records are considered to be unit conversion factors or similarly constant factors, and are therefore not “remembered” even if they contain references to registers. They cannot contain references to processor variables. Refer to Section 4.9 for more details.

2.7 OPTIONS DATA

All SINDA/FLUINT input files must begin with a single OPTIONS DATA block. The only mandatory parts are the header record and the one-line problem title. All other variable inputs are optional and defaulted if appropriate. The variables defined in the OPTIONS DATA are flags that (1) control the preprocessor's output, (2) name files for program use, and (3) perform miscellaneous functions like model name definition and spell checker control.

Figure 2-1 and Table 2-1 are a preprocessor execution flow chart and a reference summary of the OPTIONS DATA inputs. The remainder of the text in this section is for user guidance in preprocessor control. (Table 2-3 is an example of an OPTIONS DATA block.)

Table 2-1 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/folder for Rel. Engr. database	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

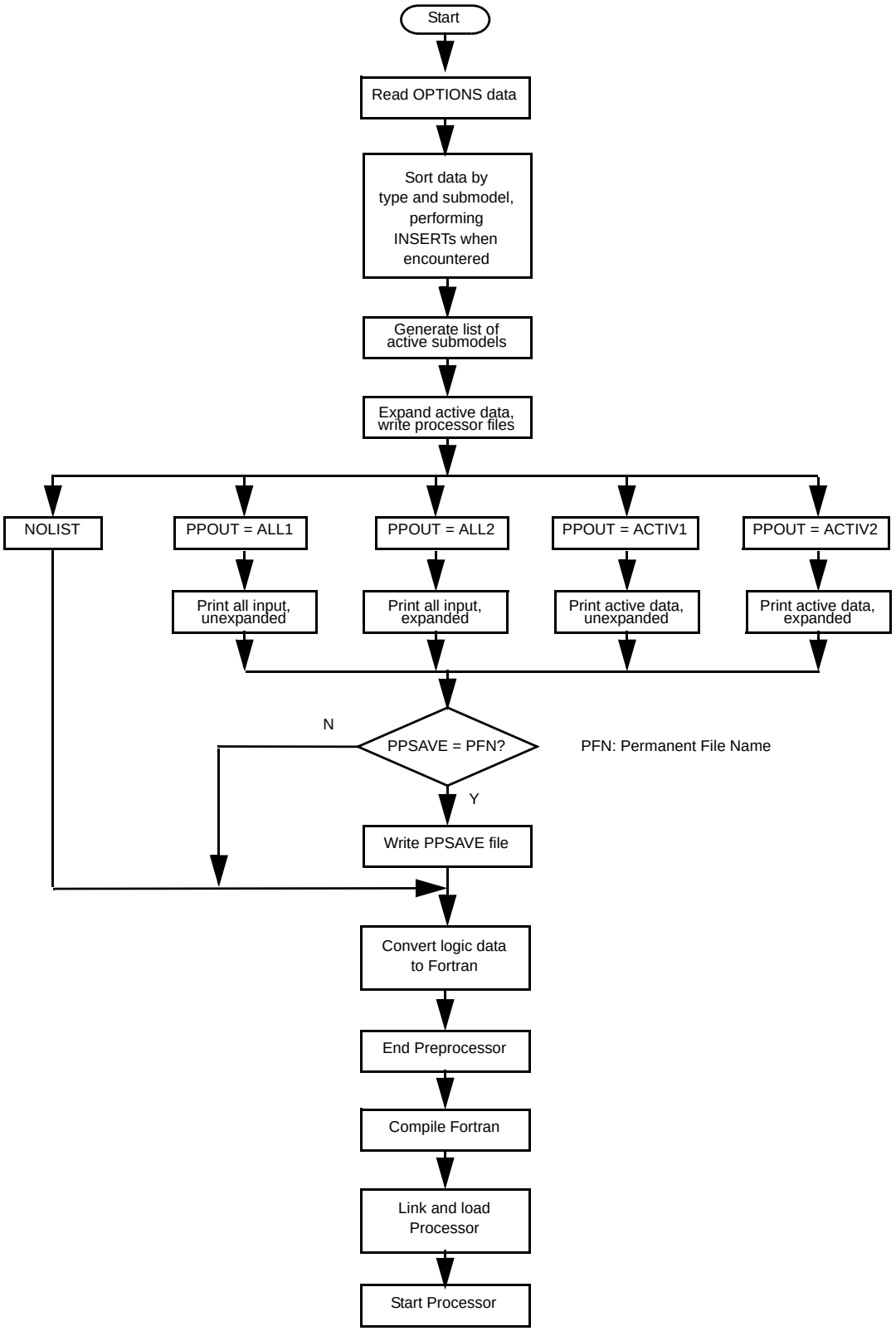


Figure 2-1 Preprocessor Flow Chart

2.7.1 File Definition

Table 2-2 presents the SINDA/FLUINT program file definition and usage. Any of these files may be saved as a file on any host system by entering the generic file mnemonic (e.g., QMAP, SAVE) along with an arbitrary name for the file in OPTIONS DATA. If files are not specifically named, some machines will save output files anyway under system-selected names such as FORT.002. Users are cautioned that some platforms (e.g., HP/Unix) do not have default file names.

Table 2-2 Program File Definitions

File Name	Function/Content	Writing Subroutine	Reading Subroutine	Type
INPUT	OPTIONS DATA, other data blocks at user's option	None	Preprocessor	Formatted
INSERT files	Any number of files containing data called in by INSERT	None	Preprocessor	Formatted
SAVE	File or folder containing all program variables and variable name dictionaries. Used for restart with RESAVE, for boundary temperature driving with BNDDRV, and for post-processor programs such as Sinaps® and EZXY®	SAVE	BNDGET Sinaps EZXY etc.	Binary
RSO	Identical format as SAVE. Allows user to write two different sets of output from same run	RESAVE	RESTAR Sinaps EZXY etc.	Binary
REDB	Identical format as SAVE. Used with Reliability Engineering	SAVEDB	RESTDB EZXY, etc.	Binary
User Defined	A program file generated by BNDGET for use by BNDDRV. This file may be saved as a file if desired	BNDGET	BNDDRV	Formatted
QMAP	File containing QMAP (thermal network values and connectivity) and/or FLOMAP (fluid network values and connectivity) output	QMAP FLOMAP		Formatted
USER1 USER2	Files provided for user output	User logic	User logic	Formatted
OUTPUT	The printed output file	various	None	Formatted

Except for the OUTPUT file, none of these inputs are defaulted, and no read/write action will occur unless the name appears in association with a file name in the OPTIONS DATA block. The default for OUTPUT is to append the processor's printout to the preprocessor printout file, as defined in the runstream (job control). The OUTPUT file can be changed during the run via the CHGOUT utility (Section 7.4.19).

The OPTIONS DATA block is the *only* place in the user's input stream where it is *necessary* to name a file (e.g., RSI, SAVE) and define its use within SINDA/FLUINT. The INSERT directive, if used, will also contain file names. The user should not use the dollar sign ($) style comments on the same line as OPTIONS DATA or INSERT file names. A similar restriction applies to file names input as CARRAY DATA for use in the USRFIL, USRFIL2, and CRASH utilities introduced below.

The user may optionally open two user files USER1 and USER2, and refer to those files as unit numbers NUSER1 and NUSER2, respectively. If more files are needed, they may be opened by calling the USRFIL or USRFIL2 routines (Section 7.4.17).

Another routine that opens a file is the optional CRASH routine, which saves snapshots of the model as frequently as desired without requiring excessive disk space (Section 4.7.4).

2.7.2 Other Options

When the spell check flag is on (the default ***and strongly recommended*** condition), the preprocessor examines each variable name found in the logic blocks to see if it matches one of the user accessible program variables, (e.g., T, Q, C, TL, PL, TIMEND, DRLXCA, etc.) or a user input array or variable name. If not, the preprocessor will raise an error flag and point out the offending variable name. See Section 4.1.1.3 for more information on this feature.

The DIRECTORIES flag turns on printout of the node, conductor, lump, path, tie, array, and constants "directories" or mappings that show the correspondence between the user's identification numbers and the internal (relative) numbering scheme.* The MLINE variable allows the user to define the number of lines per page in the printout. The MODEL name option is for the user's convenience.

The PPOUT variable defines the content of the preprocessor's printout of the input data. Five options are provided:

`NOLIST`	No printout
`PPOUT = ALL1`	Lists input records verbatim from all input files, sorted by data type & submodel, GEN records not expanded.
`PPOUT = ALL2`	Same as ALL1, with GEN records expanded. (Default condition)
`PPOUT = ACTIV1`	Same as ALL1 except only the portions of the input that will be active during the run are printed.
`PPOUT = ACTIV2`	Same as ACTIV1, with GEN records expanded.

If a PPSAVE file is defined, a copy of the printout (as defined by PPOUT) will be written to PPSAVE and saved. This file is intended to be an orderly, sorted file for the user to edit in the system text editor as the model is updated. There is a potential problem here because the file planned for editing may become very large, especially if individual GEN expansion records are present. The text editor may be very slow or simply will not take the entire file. For this reason, the INSERT

* DIRECTORIES is an anachronism preserved only for special model debugging purposes. The user should avoid using knowledge of storage locations when creating logic. LOCATIONS CANNOT BE KNOWN AHEAD OF TIME USING Sinaps NOR Thermal Desktop. Use instead dynamic translation routines such as NODTRN, INTLMP, etc.

macroinstruction, in conjunction with the PSTART & PSTOP macroinstructions, should be used to keep large input files separated into conveniently sized parts. Computer generated radiation conductor files should probably always remain as INSERT files. See Section 2.5 for information on these features. Finally, note that FLUINT macros, the equivalent of thermal GEN options, are *not* expanded into the PPSAVE file because the relationship between generated elements would be lost if they were input separately.

The user also has control over the presentation of output material via the MLINE and TITLE keywords. These values can be altered during processor execution by calling the PAGHED routine (Section 7.6).

The NOUPDATE option turns off the automatic internal calls to UPREG that otherwise occur within the solution routines. (UPREG updates the expressions used in the definition of a model to reflect changes in values of registers and processor variables.) With this flag, propagation of changes in the underlying spreadsheet become entirely the user's responsibility. See also Section 2.8, Section 4.9, and Section 7.12.

The MIXARRAY option overrides internal error checking for ARRAY DATA (Section 2.11), permitting integer and real data types to be mixed within one array. While bivariate arrays are automatically detected and checked, this option *must* be selected when inputting trivariate arrays. However, if MIXARRAY is selected the user must exercise additional precautions to make sure such arrays have been input correctly, and that no other arrays use mixed integer and real data types inadvertently.

In the past, SINDA has accepted an undocumented and discouraged network: a conductor that spans multiple node pairs. (This method was useful in the days when memory was lacking and spreadsheet options were absent.) In order to enable the conductor HR parameter and other features, multiple node pairs are now illegal. To override this error trapping and allow multiple node pairs (but forsake the ability to use conductor HR and perhaps other features), the NODEPAIR command may be used.

The ENDCHAR command allows the user to change the character that designates the end of a line in data or logic blocks.* The default character, $ (dollar sign), signals that the remainder of the line is a comment: an in-line comment designator. The ENDCHAR command may be used to choose a new ending character such as @ (i.e., "ENDCHAR = @"). The user is cautioned not to use characters such as !, #, |, ^, &, or ? since they have meaning within expressions.

2.7.3 Single Precision vs. Double Precision

In older versions, it was possible to direct that the calculation of temperatures and other solution variables be made in single precision (32 bit words). This is no longer possible in OPTIONS DATA: all calculations are made in double precision, and all variables are stored in 64 bit words (Section 4.1.3.1). Any reference to old options such as SINGLEPR or DOUBLEPR are accepted and ignored.

* Logic statements that are preceded by an 'F' are excluded since these are not processed by SINDA/FLUINT.

In user logic, Fortran intrinsics such as SNGL() and DBLE() are not necessary since they do not perform any actions. Also, double precision registers may be used interchangeably with single precision registers, since there is no longer a distinction between them.

2.7.4 Old Expressions (Version 3.1 and earlier)

In Version 3.1 and earlier, simple expressions were allowed in data blocks, and these expressions were evaluated left-to-right with no precedence given to multiple and divide operations over addition and subtraction operations. For example, in SINDA data blocks “2 + 3*4” equaled (2+3)*4=20 in old versions, whereas the current version produces 2+(3*4)=14. Also, any expression is currently treated as a real number, whereas in the prior version expressions consisting entirely of integer values (such as those shown in this paragraph) were treated as integer inputs. Although both occurrences are exceedingly rare, a potential exists for models generated under old rules to produce erroneous results or errors when run without modification under the current version: a backward compatibility issue exists.

To eliminate such concerns, add OLDEXPR to the OPTIONS DATA block of an old model and run it in the new version. All expressions will be evaluated according to the old rules. Check the tail end of the preprocessor output file for cautions regarding ambiguous expressions and integer-only expressions. These cautions are only produced if the OLDEXPR flag has been set.

Correct any such expressions such that cautions are no longer produced. Add parentheses to ambiguous expressions, and convert integer expressions to integer values. Thereafter, delete the OLDEXPR keyword. Otherwise, any complex expression will be flagged as an error, and no registers will be allowed. The OLDEXPR flag is intended to overcome backward compatibility problems only. It is unwise to leave problematic expressions uncorrected in old models.

Refer also to Section 2.8.9.

2.7.5 OPTIONS DATA Formats

Table 2-3 is an example of an OPTIONS DATA block that illustrates common options.

Table 2-3 OPTIONS DATA Block Example

```
HEADER OPTIONS DATA
TITLE    OPTIONS DATA EXAMPLE
         MODEL      =    Example Case
         RSI        =    PFN2.RSI
         RSO        =    PFN3.RSO
         OUTPUT     =    PFN4.OUT
         SAVE       =    PFN5.SAV
         QMAP       =    PFN6.MAP
         USER1      =    PFN8.US1
         USER2      =    PFN9.US2
         MLINE      =    45
         SPELLOFF
         NOUPDATE
         ENDCHAR = @
```

2.8 REGISTER DATA and Data Block Expressions

With few exceptions, almost anywhere a value is input into a SINDA/FLUINT data block, an *expression* can be supplied instead. Expressions can use multiplications (“*”), divisions (“/”), additions (“+”), subtractions (“-”), and exponentiations (“^” or “**”) nested within arbitrary levels of parentheses. Furthermore, the user can use built-in functions (like sine, cosine, and logarithms), built-in conversion constants and physical constants (π, and the Stefan-Boltzmann constant). The user can even define their own variables, called *registers*, making registers arbitrary functions of each other. Expressions can also contain conditional operations (equivalent to IF/THEN/ELSE in Fortran) and may reference any translatable SINDA/FLUINT *processor variable* such as “sub.T22” or “drlxcc” or “pipe.FR20” etc.

An expression which refers to one or more registers or processor variables, and which is used to define a SINDA/FLUINT parameter will be referred to as a *formula*.

Registers and expressions should be used extensively because they

- make a model more self-documenting. An engineer who inherits a model, or who is attempting to read and understand one of his or her own old models, will be better able to understand the sources of inputs when registers are used.
- make a model easier to change and maintain. If a model is built with sufficient forethought, a change to a single value or expression can propagate throughout the input file. It is also easy to store alternate designs and cases within a single model.
- add spreadsheet-like functions to a model. Complex interrelationships can be defined between inputs and outputs to make changes consistently, making it easier to maintain several analysis cases or designs within a single model file.
- allow model building to proceed before dimensions and properties have been finalized.

Registers and values defined by formula can be varied dynamically during processor execution, with their effects propagated automatically throughout a model. This capability greatly facilitates execution of parametric analyses and sensitivity studies (Section 4.9). Furthermore, as detailed in Section 5, real registers can be:

- used as output variables when *goal seeking*, or reversing the normal input/output sequence of SINDA/FLUINT
- used as design variables in optimization
- used as uncertainties or correlating parameters for automated test data calibration
- used as unknowns for worst-case design scenario seeking
- used as probabilistic (random) inputs for reliability estimation

Thus, the user will find it advantageous to make copious use of registers during the construction of a model. Future releases will provide even more functionality available for use with models that are parameterized using registers.

2.8.1 Introduction to Expressions and Registers

As an example of an expression, assume the radial conductivity through a cylinder were being input as a conductance using the formula $2\pi kL/\ln(r_o/r_i)$, where k=165.0, L=10.0, r_o = 3.1 and r_i is 70% of r_o. The user could easily calculate the results using the above formula and input a constant value:

```
2.907E5
```

This value implies nothing about its origin or purpose, much less the values used to calculate it. To make the model a little more self documenting, the user might pre-calculate the "$\ln(r_o/r_i)$" term, specify a value for π, and then input the rest as hard-wired constants:

```
2.0*3.1416*165.0*10.0/0.35667
```

The attentive engineer who inherits the model *might* recognize the form of the equation, but would still have no idea what r_o and r_i were, or whether 165.0 was a length or a conductivity. Using the built-in value of π and the built-in mathematical functions such as natural logarithm ("ln") yields:

```
2.0*pi*165.0*10.0/ln(3.0/2.1)
```

However, it is still not clear to a reader whether 165.0 is a length or a conductivity, and if a dimension or property changes, changes must be made throughout the input file. To solve this problem, users can define their own arbitrarily-named variables, called "registers," and use them throughout the input file.

For example, the user in the above example might create a new input block:

```
HEADER REGISTER DATA
       cond = 165.
C Despite starting with "l", "length" is real
       length = 10.0
C Note that registers can refer to other registers,
C and that they can be input in any order
       rinner = 0.7*router
       router = 3.0
```

and then input the previous expression as:

```
2.0*pi*cond*length/ln(router/rinner)
```

Thus, both the formula used and the values used in that formula are self-evident. Furthermore, *changes to register values and to the expressions defining them are propagated through the input file automatically, perhaps even during processor execution.* Therefore, a change to the value of *router* in one place (REGISTER DATA) will automatically change *rinner* as well as the resulting value of all expressions in which either register was used throughout the input file.

Furthermore, the user may directly refer to ***processor variables*** (either input or response variables such as temperatures, flow rates, control constants, etc.) within almost any expression. For example, the user might choose to refer to the hydraulic diameter of a certain line (tube #111 in fluid submodel "loop", for example) to define *rinner*:

```
RINNER  = 0.5*LOOP.DH111
```

Or, the user may wish to add simple temperature-varying conductivity by referring to the temperature of one or more nodes in the definition of the conductivity COND or the conductance itself.

The rules governing references of processor variables are in general the same as those that apply in logic blocks (Section 4.1.4).

Other commands are available to enable a single expression to be "reused" by indirectly referring to a element or the elements connected to it as a convenience in the generation and maintenance of a model. For example, in defining the source on a node the user can refer to the temperature of the node as "T#this" or "T(#this)." In defining a conductor, the user may refer to the first node as "#nodeA" and the second node as "#nodeB."* Many other examples of indirect referencing exist for other network elements. Using indirect references not only facilitates the use of macrocommands and generation options, but it also makes the model easier to maintain if element identifiers are changed.

Expressions may also contain conditions. For example, the user may wish to use a lower estimate (160.0) for the conductivity in a hot case, and use a lower value (165.0) for a cold case. Using register variables "HOT" and "CASE," the above might be specified as follows:

```
COND    = (CASE==HOT) ? 160.0 : 165.0
```

The remainder of this section describes these features in more detail. See also Table 1-1 versus Table 1-2. Section 4.9 and Section 5 describe advanced operations that can be performed on models exploiting registers in their definitions. Section 4.9.8 attempts to clarify confusion that may exist between register-based arithmetic and updating and seemingly similar operations performed in logic blocks using Fortran.

2.8.2 Evaluation and Usage of Expressions

Expressions may be used any place a real (floating point) value is required within the SINDA input file data blocks: node temperatures, capacitances, SIV factors, tank volumes, tie conductances, tie path factors, etc. "Expressions" can mean:

1) a floating point value (e.g., "1.0", "3.", "3.0e-7");

2) a combination of floating point ("real") *or integer* values, perhaps using internal built-in functions (e.g., "-2.0*4", "11-2*3", "4.5*pi*sin(4.0)") and conditional operations (e.g., "(a>b)?c:d");

3) a register name, or combination of register names (e.g., "fred", "fred+wilma*barney");

* The above capitalization is for legibility: expressions are all case insensitive.

4) a reference to a processor variable (e.g., “t22”, “fred.q214”, “drlxcc”, or “fr#this”)

5) a combination of any or all of the above (e.g., “2.0*pi*cond*TLEN22/ln(router/rinner)”).

Expressions cannot be split across an input line: each expression must be contained within a single line (1000 characters). If an expression cannot fit on one line, create a new register or registers containing subexpressions and substitute the register name(s) in the original expression. In other words, use registers to break long expressions down into smaller more manageable units (see the example at the end of this subsection).

Expressions *cannot* be used any place a specific integer value *must* be defined, such as node or array identifier, or as an increment for such identifiers. If expressions are used in such an instance, a preprocessor error will occur. However, they *are* allowed in places where identifiers are not needed, in which case the real result of an expression will be rounded to the nearest integer.

Expressions containing references to processor variables (which have no meaning in the preprocessor) cannot be used to define certain initial conditions, or a preprocessor error will occur. See Section 2.8.8 for more details and examples.

Expressions are case insensitive: no distinction is made better upper and lower case letters.

With the exception of single integer values (such as “1”, “30293”, etc.), ***all expressions are treated as real (floating point) values***. If an integer value is expected instead (e.g.“ITEST = ...” in USER DATA, GLOBAL, tube IPDC values, integer control constants such as MATMET, etc.), the real value will be rounded to the nearest integer. However, expressions placed in numbered USER DATA and ARRAY DATA are of unknown type, and therefore if an expression is used (including “integer” expressions such as “3+4”), it will be treated as a real value,* and must be handled as such if accessed within logic blocks or routines called from logic blocks.

The resolution of expressions follows normal precedence rules, such as those followed by the Fortran and C languages. However, if there is any question about how precedence is applied, parentheses should be applied.

If there is a problem in any expression, a syntax error will be reported. A syntax error can consist of unbalanced parentheses, misspelled built-in constants, functions, or register names, etc., mathematical problems such as division by zero, logarithm of a number less than 1.0, etc. If the problem cannot be resolved by simple inspection, break the expression down into smaller subexpressions using additional registers. For example:

```
geff=1.0/(ln(roo/rii)/(2.0*pi*len*con) + thick/(area*cslab))
```

becomes (is equivalent to):

```
geff       = 1.0/(1.0/grad + 1.0/gslab)
grad       = 2.0*pi*len*con/ln(roo/rii)
gslab      = area*cslab/thick
```

* Caution: this treatment is unlike versions prior to Version 3.2. See Section 2.8.9.

2.8.3 Defining Registers: HEADER REGISTER DATA

An optional header block is available for defining registers that are to be used elsewhere within the same input file. Note that registers are entirely optional: full expressions may still be used anywhere within the input file even if no registers have been defined.

Input Rules: Each register is defined on a separate, single line that must not exceed 1000 characters. Therefore, expressions themselves are limited to no more than 1000 characters. If the expression is too large to fit within this limit, it should be broken down into subexpressions using multiple registers. The current limit is 15,000 registers.

The generic format for this block is:

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

Registers are internally stored as real (floating point) variables, and expressions are evaluated using floating point math. In Fortran-based logic blocks, however, they may alternatively be chosen to be integer ("INT:") as explained below.*

An example of such a block is as follows:

```
HEADER REGISTER DATA
       WHAM    = sin(90.0*dtor)
       INT:PRELIM= 1      $ "PRELIM" is an integer in logic
       BAM     = 40.5E6-10.0
       ETC     = arcsin(wham)^2
       d_score = one2one
       delta_pressure_filter = main.pl12 - main.pl13
       Big_Accurate_Number = 1.31904144805567D120
```

The definitions of each register should not start in column 1, but otherwise no column restrictions exist. Registers must be called by a unique alphanumeric name that does not conflict with other reserved words used in SINDA/FLUINT (including named user constants), otherwise an error will be produced. *A maximum of 32 characters is allowed per register name.* No distinction is made between upper and lower case. Underscores are allowed (except as leading characters), but other special characters are illegal.

Note that "HEADER REGISTERS" or "HEADER REGISTERS DATA" may be used equivalently. More than one such block is allowed per model, but they are collectively treated as a single block that happened to be input separately. They may be INSERTed, or may contain INSERT commands (Section 2.5).

---

* In prior versions, which distinguished between single and double precisions sizes in user logic, the prefixes SP:, REAL:, and DP: were all valid. The current version makes no distinction between these, so they are all option. However, using DP: will still cause the registers so designated to be sorted and stored at the end of the list (important only for output purposes).

When defining a register expression, references to processor variables may be used as can names of other registers. However, a register should not refer to itself, and circular references similarly should be avoided. Otherwise, nonconvergence may occur (see discussion below).

By default, all registers are of type real, and all calculation procedures are performed using floating point arithmetic.

Real registers are accurate to about 15 significant figures, and can range in exponent from about 10^{-307} to 10^{+308}. (If such extremely large or small values are used for most SINDA/FLUINT inputs, trouble will likely arise, so such exponents should be only used within logic and not within expressions.)

To designate a register as being of type *integer* for use in Fortran-based logic blocks (e.g., VARIABLES 1, FLOGIC 0, etc.), a prefix of "INT:" may be added. This prefix is an indicator only, and is not considered part of the register name.

The Meaning of "INT:" All registers are treated as real (floating point) within the preprocessor. "INT:" *only has meaning once the value has been passed to the processor* (e.g., within logic blocks). Thus, despite defining:

```
INT:FRED = WILMA
```

"INT:FRED" will be treated no differently from "FRED" within the preprocessor. That is, if WILMA is real and is equal to 3.45, then FRED will similarly be equal to 3.45 (not 3.0 nor 4.0) during preprocessor execution and other spreadsheet database updates. However, upon being passed to the processor, FRED will be rounded to 3. It will be passed as a Fortran integer (INTEGER*8) variable of value "3", *and should be treated as an integer by any logic that accesses it.* (In Sinaps, pressing the INT button is equivalent to using "INT:", and Sinaps applies the same rules described above.)

Most variables can and should be left as the default type real. For example, if "NUMGRV" is the number of grooves, then leaving it as a real value of 45.0 does not imply that there are a non-integral number of grooves in the real device, even if 45.0 is represented as 44.99998 or 45.0001 internally. *Rarely does a variable need to be an integer.* Exceptions include variables that are used to control logic (via IF-type checks in logic blocks), or that contain fluid identifiers to be passed to the processor for use in CHGFLD, or that contain node or array numbers for use in logic block operations, etc. In such instances, integer USER DATA, GLOBAL "constants" can be used equivalently.

Initialization and Iterations: Since registers often refer to other registers and perhaps to processor variables, algorithms exist within SINDA/FLUINT to initialize values and to iterate until convergence. Registers and processor variables are evaluated cyclically (in input order, for registers). If a register contains a reference to a processor variable or to another register that has not yet been encountered (and therefore the expression cannot yet be evaluated), then the missing variable is arbitrarily set to a random value in the range of 1.0 (exclusive) to 2.0 (inclusive). If a syntax error or other problem occurs thereafter, an attempt is made to reinitialize all registers over a different interval and try once more. Further encounters of syntax errors are reported as invalid expressions. *Processor variables are completely undefined in the preprocessor, and hence expressions referring*

to them directly or indirectly will always use random values until processor execution begins. See also Section 4.9.

As noted above, if registers refer to each other or to processor variables, iterations may be required to find the final set of values. Iterations continue until convergence (defined as all nonzero registers and expressions changing by less than 1% per iteration), or until 100 passes have been attempted in which case an error condition is reported. The preprocessor tabulates the final values (reported as reals). An optional processor routine (REGTAB, with no arguments) is available to tabulate the values and expressions as passed to the processor, listing real registers first followed by integer registers.

Updates of registers, expressions, and processor variables will happen automatically at many intervals during the processor operation to keep relationships "fresh." This updating can be turned off by the user (using NOUPDATE in OPTIONS DATA, Section 2.7.2) who can then assume complete control of the process (Section 4.9 and Section 7.12). Alternatively, the user may insert "CALL UPREG" at any point in any logic block to force the update to occur, including perhaps iterative resolution of any interdependencies.

2.8.4 Built-In Constants

For convenience, various physical parameters and common unit conversion constants are prestored and available for use in any expression as shown in Table 2-4. If the user creates a register of the same name, then that register will override the internal value for that model but no warning will be issued.

These values are not available as Fortran variables within logic blocks. To "fetch" a value such as pi for use within logic blocks, simply create a register and set it equal to pi (or use perhaps a named USER DATA, GLOBAL variable), such as "pie = pi."

Table 2-4 Built-in Constants

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
wmktobhfr	1.73073	Watt/m-K to BTU/hr-ft-R

2.8.5 Built-in Functions

Common mathematical functions are also available for use anywhere within an expression, as shown in Table 2-5. Their arguments (such as “float” in the following table) can be values, register names, or subexpressions.

Table 2-5 Built-in Functions

Function	Description	Example
abs	absolute value	abs(float)
acos	arccosine (radians)	acos(float)
asin	arcsine (radians)	asin(float)
atan	arctangent (radians)	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[a]	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)

a. Note: In Fortran, “LOG” is a natural logarithm, and “LOG10” is a base-10 logarithm.

These functions, which are only available within expressions, should not be confused with similar Fortran functions used in user logic, since those functions may operate somewhat differently. For example, “max” can only have two arguments in these expressions, whereas any number of arguments may be used in the Fortran intrinsic function of the same name.

Particular caution should be applied to the LOG function, which is the base-10 logarithm within the expression system, but is the natural logarithm inside of Fortran logic.

2.8.6 Logic Blocks and Expressions

[The novice reader should be familiar with the rules of logic translation and references to SINDA/FLUINT variables within logic blocks, Section 4 in general and Section 4.1.4 in particular, before using the information in this section.]

2.8.6.1 Referring to Registers within Logic Blocks

Registers are available for use in Fortran logic blocks, which are discussed in Section 4. A register can be used to define a variable (i.e., validate it with respect to the spell checker), and to make it available for use or inspection in all logic blocks.

Registers are real (floating point) by default, whether or not they start with I through N (contrary to the Fortran convention used in USER DATA, GLOBAL). If accessed in logic, registers should normally be treated as floating point variables. As was noted above (Section 2.8.3), to declare a register as integer, use the “INT:” prefix in HEADER REGISTER DATA, and treat the register as an integer variable in logic. Although it will be treated as a real variable wherever it is used within expressions, its value will be rounded to the nearest integer upon being passed to the processor.

A tabulation of current register names, values, and expressions may be produced by calling the REGTAB output routine from within logic blocks, as described in Section 7.

By default, changes to the values of registers made within logic blocks automatically affect all the expressions in which they were used, including expressions used to define other registers. This information is updated and propagated throughout the model at a various points during the network solutions.

Otherwise, if NOUPDATE has been specified in OPTIONS DATA, propagation of updates is completely within the user's control.* In this case, in order to propagate such changes to both registers *and* to all parameters using these registers in their definitions and vice versa, one or more of the UPREG family of routines must be called (Section 4.9, Section 7.12) if the user has suspended automatic updates.

Any direct (non expression-based) changes to the value of registers in logic will also automatically overwrite their defining expression. For example, if in REGISTER DATA a register had been defined as:

```
VARTWO = pi*Diam^2/4
```

but later, in a logic block, the following statement is executed:

```
VARTWO = ftest $ where ftest is currently 2.3
```

then the defining expression for *VARTWO* would now be "VARTWO=2.3" as if that expression had been input in REGISTER DATA, and any changes to *Diam* would no longer have an effect on the value of *VARTWO*.†

Real registers may also be changed by the Solver or by any of the Reliability Engineering modules if the registers were named as either design variables or random variables, as described in Section 5.

2.8.6.2 Conflicts Between Logic and Expressions

Changes to the model may be invoked either "directly" in logic, via executable pseudo-Fortran statements, or "indirectly" via the expression system. However, *for any one variable, only one or the other method should be used* otherwise conflicts will result that cannot be trapped by the software.

For example, if the temperature of a boundary node (Section 2.12) is defined as equal to a register *Tsource*, then if the value of *Tsource* is ever changed in logic, the temperature of the boundary node would be updated automatically (after the logic block returns, or the next output or solution routine is called, etc.). Alternatively, the user can just change the temperature of the boundary node directly in logic (e.g., "T33 = new_value") in which case this change will be respected as long as *Tsource* is not subsequently changed. If *both* the temperature of the node and *Tsource* are updated in logic, then the actual value of the nodal temperature will be difficult to predict. The user must avoid such conflicts by either sticking to the expression-based system (*Tsource* changes) or to the logic-based system (T33 changes), at least for any single variable. (Logic and expression-based changes *can* otherwise be mixed within any one model.)

Specifically, if a variable has been defined by an expression but then that variable is changed in logic, then either the underlying parameters of the original expression should not be changed, or the DEREG family of routines (Section 7) should be used to disconnect the automatic update. *If the*

* Use of NOUPDATE is strongly discouraged since important routines such as SOLVER and PSWEEP rely on being able to command their own updates.

† Explicit changes to the defining expression for registers can be made via the CHGREG utility, as described in Section 7.

parameter is to be varied in logic, the best solution is not to define it via an expression in the first place. Otherwise, expression-based updates are encouraged whenever possible over logic-based updates.

Another area of potential conflict arises from partial restore operations (as part of restart operations common to parametric transients, for example). Refer to Section 4.9.6.3 and Section 4.9.7 for more details.

Finally, note that a special class of conflict can arise for fluid lump initial states when a condensible or volatile species is present, and Gibb's Phase Rule must be obeyed. Refer to Section 3.3 for more details.

2.8.7 Advanced Expressions: Conditional Operators

It is often convenient to choose between one or more values within an expression. *Conditional operators*, equivalent to IF/THEN/ELSE in Fortran, may be used to accomplish such elections.

For example, simple thermostatic heater control logic could be implemented by defining the source on a node as:

```
(T22 > 25.0)? 0.0 : 50.0
```

Meaning turn the heater off (zero power) if the temperature of node 22 exceeds 25 degrees, else turn it on to 50 units of power if T22 is less than or equal to 25 degrees. The format* of an conditional expression is:

```
(expr1 OP expr2) ? expr3 : expr4
```

where "OP" can be:

== . . . . . equal to
> . . . . . greater than
< . . . . . less than
>= . . . . . greater than or equal to
<= . . . . . less than or equal to
!= . . . . . not equal to

Thus, "(expr1 OP expr2) ? expr3 : expr4" means that if "expr1 OP expr2" is true, the phrase returns expr3 otherwise it returns expr4. For example, in REGISTER DATA:

```
bob = (CASE != HOT)? fred : 2.0*fred
```

* This syntax is basically the same as that used in the C language. However, unlike C, parentheses are required for the first operand, and a single expression cannot be used as the first operand: a comparative operator must be used. Also unlike C, all expressions are evaluated and must be mathematically valid.

means use “fred” if CASE is *not* equal to HOT, otherwise use “2.0*fred” instead. This is equivalent to the Fortran:

```
IF(CASE .NE. HOT)THEN
        BOB     = FRED
ELSE
        BOB     = 2.0*FRED
ENDIF
```

Conditional expressions are useful for many simulation tasks as well as handling multiple cases in one model: alternate boundary or initial conditions, for example. They may even be nested indefinitely: a conditional expression can be substituted for any or all of expr1, expr2, expr3, or expr4 in the above format definition. (Explicit use of parentheses is strongly encouraged when nesting conditional expressions.)

Extending the previous thermostatic heater example to model a heater with a deadband,* the following source might be applied to node 22:

```
(T22 > 25.0)? 0.0 : ((T22 < 20.0)? 50.0 : Q22)
```

The above logic sets the heat rate to be zero if the temperature is greater than 25, sets it to be 50 if the temperature is less than 20, and otherwise within the deadband (20 to 25) the last value of the heat rate is used to model switch hysteresis.

Caution: All target subexpressions (i.e., *both* expr3 *and* expr4 in the above format) must be mathematically valid even if the conditional operator doesn’t select that expression. For example, a conditional expression cannot be used to avoid a mathematical problem such as dividing by zero:

```
y = ( x != 0 )? 1/x : 1.0e30          $ invalid if x=0
```

Logical Operators: Even more elaborate conditional expressions may be constructed using the “&&” (AND) and “||” (OR) logical operators, including parenthetical subexpressions. For example:

```
((expr1 OP1 expr2) && (expr3 OP2 expr4))? expr5:expr6
```

means that if BOTH “expr1 OP1 expr2” AND “expr3 OP2 expr4” are true, then expr5 else expr6. As a specific example, consider:

```
FK = ( (PL1>PSET && XL1!=0) || PL1>PMAX )? KMIN : KMAX
```

In English, this means “If the pressure of lump 1 is greater than PSET and its quality is not zero, or if its pressure exceeds PMAX, then set the K-factor to KMIN, otherwise set it to KMAX.”

* Actually, the HEATER routine (Section 7) is much more convenient to use in this instance.

2.8.8 Advanced Expressions: References to Processor Variables

[The novice reader should be familiar with the rules of logic translation and references to SINDA/FLUINT variables within logic blocks, Section 4 in general and Section 4.1.4 in particular, before using the information in this section.]

In almost all cases, the user can refer to *processor variables* within an expression. A processor variable is a reference such as "smn.T33," meaning the temperature of node #33 in submodel "smn."

Processor variables are unique in that they have no value until the processor starts (see Section 1.6): they have no meaning in the preprocessor (and are assigned random values during that preliminary phase of program operation -- see Section 2.8.8.3) and therefore special rules apply to their use. However, use of processor variables in expressions represents an extremely powerful option that simplifies many modeling tasks, and renders many SINDA/FLUINT options archaic. In essence, ***each input field in SINDA/FLUINT can become almost as powerful as was a logic block in old SINDA.***

Almost any variable that can be referenced in logic blocks (Section 4.1.4) can be referenced within an input expression, including thermal and fluid network inputs (e.g., "G", "DH"), results (e.g., "T", "FR", "VOL"), and status and control parameters (e.g., "LOOPCT", "DRLXCC", etc.). Variables that *cannot* be referenced within expressions include:

1. Global (named) user data variables (HEADER USER DATA, GLOBAL, Section 2.9)
2. Named constraint variables (HEADER CONSTRAINT DATA, Section 5.6)
3. Unsubscripted real array references (Section 2.11). In other words, "A(10)" is illegal since it refers internally to an integer pointer. Such a format is only valid when used as an argument in subroutines expecting a traditional SINDA array. However, the user can refer to the size of an array in an unsubscripted fashion as "NA(10)." Or the user can refer to specific elements within an array as "A(10+3)" or "NA(12+30)."

2.8.8.1 Increment Operator with Processor Variables

This feature has been discontinued, and is documented only to assist in reviewing or updating historical models.

When using macrocommands to generate sets of elements (nodes, lumps, etc.), if the expressions defining these elements refer to processor variables that must differ for each generated element, the user will find it convenient to use the "base#increment" syntax. For example, if 10 nodes numbered 10, 20, 30 ... 100 are being generated and they are to inherit the capacitances from nodes 110 through 200, either the user could input them separately:

```
10, Tinit, C110
20, Tinit, C120
...
100, Tinit, C200
```

or as a single GEN statement using the “base#increment” syntax:

```
GEN 10,10,10, Tinit, C110#10
```

In other words, “110#10” will be “110” the first time it is evaluated, “120” the second time, etc. Parentheses are optional. Negative increments are legal as long as the generated number is positive. For example, “C(110#-10)” becomes the sequence “C(110), C(100), C(90), ...”

The “base#increment” format can only be used in expressions referencing processor variables.

2.8.8.2 Indirect References to Processor Variables

When defining an expression for a network element, that expression will often refer to attributes of that element itself, or of adjacent elements. For example, in the previous example defining the source on node number 22:

```
(T22>25.0)? 0.0 : ((T22<20.0)? 50.0 : Q22)
```

the source was defined on the basis of the temperature of the node itself. This could equivalently (and preferentially) been input using the indirect operation “#this” as:

```
(T#this>25.0)? 0.0 : ((T#this<20.0)? 50.0 : Q#this)
```

Indirect referencing makes it easier to generate multiple elements (as an alternative to the increment operator described above), and to edit multiple elements using Sinaps or Thermal Desktop, but these operators are particularly convenient since the underlying expressions do not change when the model is resequenced or when the element identifiers change for whatever reason.*

Network elements themselves will be defined in later sections of this and the next chapter. However, as a convenient reference, Table 2-6 lists the indirect operators available in SINDA/FLUINT. Note that these operators are only available in NODE, SOURCE, CONDUCTOR, and FLOW DATA.

Table 2-6 Indirect Operators for Referencing Processor Variables in Expressions

Element	Operator	Meaning
Node	#this	the identifier of the current node. Use **#num** instead to exclude submodel prefix, perhaps as needed for performing math, or for passing IDs as subroutine arguments.
Source	#this	the identifier of the current node (to which the source applies). Use **#num** to exclude submodel prefix.
Conductor	#this	the identifier of the current conductor. Use **#num** to exclude submodel prefix.
	#nodeA	the identifier of the first node[a]
	#nodeB	the identifier of the second node[a]
Lump	#this	the identifier of the current lump. Use **#num** to exclude submodel prefix.
	#twin	the identifier of the twinned tank (if applicable)[b]

* Thermal Desktop and Sinaps users should familiarize themselves with “Network Logic,” a powerful Fortran generation option that exploits the use of indirect referencing syntax to automate the creation of user logic.

Table 2-6 Indirect Operators for Referencing Processor Variables in Expressions

Path	#this	the identifier of the current path. Use **#num** to exclude submodel prefix.
	#up	the identifier of the *defined* upstream lump (does not change with flow reversal). This refers to the primary tank if they are twinned. Use **#uptwin** to refer to the secondary tank.
	#down	the identifier of the *defined* downstream lump (does not change with flow reversal). This refers to the primary tank if they are twinned. Use **#downtwin** to refer to the secondary tank.
	#twin	the identifier of the twinned path (if applicable)[b]
Tie	#this	the identifier of the current tie. Use **#num** to exclude submodel prefix.
	#node	the identifier of the attached node[a]
	#lump	the identifier of the attached lump (not valid for HTUS, HTNS). This refers to the primary tank if they are twinned. Use **#lumptwin** to refer to the secondary tank.
	#path	the identifier of the path (not valid for HTU, HTUS). This refers to the primary path if they are twinned. Use **#pathtwin** to refer to the secondary path.
	#path2	the identifier of the second path (only valid for HTNC). This refers to the primary path if they are twinned. Use **#path2twin** to refer to the secondary path.
	#twin	the identifier of the twinned tie (if applicable)[b]
Ftie	#this	the identifier of the current ftie. Use **#num** to exclude submodel prefix.
	#lumpC	the identifier of the first lump. This refers to the primary tank if they are twinned. Use **#lumpCtwin** to refer to the secondary tank.
	#lumpD	the identifier of the second lump. This refers to the primary tank if they are twinned. Use **#lumpDtwin** to refer to the secondary tank.
	#path	the identifier of the path (for AXIAL fties only). This refers to the primary path if they are twinned. Use **#pathtwin** to refer to the secondary path.
	#twin	the identifier of the twinned ftie (if applicable)[b]
Iface	#this	the identifier of the current iface. Use **#num** to exclude submodel prefix.
	#tankA	the identifier of tank A (the reference tank)
	#tankB	the identifier of tank B (the second tank)

a. For all ties and for cross-model conductors, the submodel prefix *cannot* be used in combination with these operators since the correct submodel is already known to the program, and a submodel prefix is therefore redundant.
b. Except in macros, the twinned element number must first be defined on a prior line (using TWIN= .., LTWIN= ... etc.) before #twin can be used in an expression. It cannot be used later in the subblock, or in the same line as #twin.

2.8.8.3 Cautions Regarding Processor Variable References

Caution—Because registers can be used to define network inputs, and because these input variables can in turn be used to define registers and other input variables, the possibility of infinite loops of self-referencing can occur. These infinite loops are trapped in the processor if the system of expressions fails to converge in a reasonable number of iterations.

Caution—Unless the user has specified NOUPDATE in OPTIONS DATA, the internal "spreadsheet" underlying the model will be updated often but at discrete times during the solution, usually after most user logic has been completed for that solution step. Timing conflicts can arise if some parameters are updated in logic blocks and others via expressions. Generally, expression updates are made frequently, and therefore happen last (i.e, after user logic has been executed for that cycle and before the networks are updated). Rarely, the user may need to specify NOUPDATE and assume control of the update process explicitly using the UPREG family of routines (Section 4.9, Section 7.12). *Use of NOUPDATE is strongly discouraged,* however, since it impairs the operation of many features such as the Solver and parametric sweeps (e.g., PSWEEP). Instead, insert "CALL UPREG" at any place in a logic block to command an update of the underlying expressions.

Caution—The SINDA nodal source term, Q, is unique in that it is reset and rebuilt every solution step (steady state iteration or transient time step). Therefore, references to Q terms (e.g., "Q44" or "bar.Q504") in expressions are dangerous since their value can change sporadically at different times during the solution.

Caution—References to K vs. XK and A vs. NA should reflect whether the user originally defined those user constant and array cells as containing real or integer data, as with any reference in user logic.

Caution—By their very name, "processor" variables have no meaning or value in the preprocessor, which is a once-through operation that sorts and prepares data and logic for the creation of the processor (see Section 1.6). Thus, when they are encountered in the preprocessor, processor variables are assigned temporary random values in the range of $1.0 < value <= 2.0$ such that expressions can still be evaluated for valid syntax. *This means that normal checks for valid range are suspended in the preprocessor when a processor variable is used to define a parameter, since the exact value of the resulting expression cannot yet be known until the processor starts.* For example, the checks to prevent a negative flow area are suspended if the flow area were defined on the basis of the diameter (i.e., "AF=π/4*DH#this^2"). The processor, on the other hand, does not perform exhaustive checks of variables for validity in order to prevent excessive computational overhead. Therefore, by using expressions containing processor variables, *the user assumes most of the responsibility of assuring that both the expressions and the inputs to those expressions are correct.*

Caution—Many processor variables cannot be initialized before certain processor analysis operations begin, but these variables are still valid references within input expressions. Examples of such variables include QTIE, DRLXCC, CASL, and TIMEN. Thus, these variables represent a bootstrapping problem: they do not have a valid value (and are normally initialized to zero) until the solution has proceeded far enough, yet input variables requiring those values to be set may have already been encountered and set. In most cases this does not represent a problem. However, consider the example of defining a source based on the value of TIMEN (the current problem time):

```
1, power/timen
```

Until TIMEN has been calculated in the first transient time step, the above expression will divide by zero and hence result in an arithmetic fault. Also, the source applied to that node in the first step will be undefined as long as TIMEN is undefined.

Conditional expressions may be used as a remedy, and in the following case keeping the source zero (or at least a very small number) until TIMEN has been initialized, which will include perhaps any preliminary steady-state runs:

```
1, power/((timen == 0)? 1.0E10 : timen)
```

Restriction—Initial conditions for certain SINDA/FLUINT variables that are almost always output (response) variables cannot be defined using references to processor variables. Preprocessor error messages will be produced if this case is detected.

For example, consider the initial temperature of an arithmetic or diffusion (but not boundary) node (Section 2.12). This initial temperature can be input using a formula that includes a reference to registers. However, only the initial value of the register is used: thereafter, no updates to those registers will affect the node's temperature (Section 4.9) in order to avoid conflicting with SINDA/FLUINT calculations. Because processor variables have no initial value in the preprocessor, however, it is impossible to use them to define the initial temperature of an arithmetic (or diffusion) node. Initial tank and junction (but not plenum) thermodynamic states follow a similar restriction.

Tank volumes (Section 3) represent a special case since they can be both input and output variables. Therefore, the processor will automatically check to see if a potential for a problem exists, and will caution the user if so.

2.8.9 Limitations to Backward Compatibility of Old Expressions

[This section applies only to certain models created for Versions 3.1 and earlier.]

The casual user should note that, despite the length and detail of the following discussion, actual occurrences of problems with backward compatibility are extremely rare, but must be fully documented nonetheless.

Old Precedence vs. New Precedence: The prior (and extremely primitive) expression evaluator in SINDA worked left to right, with no precedence given to * or / operations. Thus, in SINDA data blocks "2 + 3*4" equaled (2+3)*4=20 in old versions, whereas the current version produces 2+(3*4)=14. The previous treatment was confusing since it conflicted with every other language, and therefore many users consider it to be a "bug." Nonetheless, generations of SINDA users had adapted to that strange convention, and some older models might therefore achieve different results when using the new version. (Whether the new results or the old results are correct depends on whether the user understood and correctly adapted for the prior left-to-right convention.) Fortunately, expressions combining +,- operations with *,/ operations are rarely used, and those combining +,- operations *followed by* *,/ operations (which are the only case that generates ambiguity) are even more rare.

The current version issues no warnings when it encounters an expression that is potentially ambiguous (e.g, if this model has been carried over from an older version). However, the older logic and expression handler can be retrieved by adding the “OLDEXPR” keyword into the OPTIONS DATA block (Section 2.7.4). *If there is any doubt as to whether or not an old model contains potentially erroneous expressions, use the OLDEXPR option and check the preprocessor output file for cautions and warnings. If such expressions are found, remove the ambiguity by adding parentheses*.

If OLDEXPR is chosen, none of the new expression handling capabilities will be applied: this option is applied globally. Therefore, once a model has been “checked” and new expressions are used, the OLDEXPR flag should no longer be used.

Integer-only Expressions: In the old expression handler, expressions consisting only of integers (e.g., “11-1”, “3+4”) where preserved as integers, meaning they could be used to define element identifiers (e.g., node numbers), integer user data (e.g., “1 = 3+4”, and therefore accessible in logic blocks as K1, not XK1), integer data within arrays, etc. Although the occurrence of such expressions are extremely rare, the user should note that the new handler treats *all* expressions as floating point values. Only single integer values (e.g., “11”, “3030”) are truly handled as integer inputs. Therefore, integer expressions can no longer be used to define element IDs or increments of those IDs. If the program finds such an expression in an old model (an unlikely occurrence), an error will be issued and the user must change the input file.

Certain data expecting integers (e.g., “ITEST = ...” in USER DATA, GLOBAL, tube IPDC values, integer control constants, etc.) will be rounded to an integer value without issuing a caution, and should therefore not cause the user any concern.

The one area that *is* a concern is the use of integer expressions (not just single integer values) in numbered (submodel-specific) USER DATA and in ARRAY DATA (Section 2.10 and Section 2.11). Such expressions (which again are rarely ever used) are be treated as *real* inputs by the current version. This means that the data values are accessible in logic using A and XK, not NA and K, and that any user routines accessing those values must be rewritten. The code has no means of detecting such an instance, but the user can check for such problems by using the OLDEXPR flag and watching for warnings in the preprocessor output file. (Such warnings were *not* produced in earlier versions, unlike the ambiguous precedence warning described above.)

2.9 Global (Named) USER DATA

Named user constants are a historical feature that have been largely replaced by registers. See Section 2.8 and Section 2.9.3.

User constants ("constants" is a historical misnomer) are all optional and are never referenced by any preprogrammed part of the processor. Rather, they are referenced and used only in other portions of the model, namely as Fortran variables in logic blocks. *Unlike registers, they cannot be used within input expressions: they are intended instead to control processor logic or provide data to options invoked in the processor.*

Users may or may not identify particular constants with a particular submodel, at their option. The "GLOBAL" submodel name is provided for *named constants* (e.g., 'FRED' or 'FISH') that are not referenced using a "real" submodel name as an identifying qualifier. Named USER DATA, described in this section, are primarily used for declaring* user Fortran variables and for making them available in all logic blocks.

Submodel-specific (*numbered)* constants, the subject of the next section (Section 2.10), have predefined names with user-input identifiers. They may be used as alternatives to named (GLOBAL) USER DATA, although their primary purpose is to enable variable inputs in NODE, CONDUCTOR, and SOURCE DATA blocks.

The global block must precede the submodel blocks even though the two types of constants appear unrelated.

2.9.1 Input Formats

Global user constants are single-value program variables that are labeled by any unique symbolic variable name that is not a program reserved name (Section 6.18.5) or register name (Section 2.8). The variable names are supplied along with each constant's initial value in the USER DATA blocks under submodel GLOBAL. The following input rules apply:

1) This block must precede the other USER DATA blocks (Section 2.10)
2) Only symbolic name identifiers of up to 32 characters with an alphabetic lead character are allowed, e.g.: ISTOO, XMAX1. Numeric identifiers are not allowed.
3) Only one GLOBAL block is allowed. Each global constant name must be unique.
4) Integer identifiers must be entered with integer data values in order to initialize the data value as an integer. The same with real identifiers/data values. *Unlike registers, the implicit Fortran identifier rule applies.* That is, identifiers beginning with I, J, K, L, M or N will be treated as integers or fixed point. All others are real (floating point).
5) Character variables are not allowed. CARRAY DATA blocks, described later, are provided for this.

* "Declaring" means that these names will not be flagged by the optional debug feature, described in Section 4.1.1.3, as potential typographical errors.

6) The names ATEST, BTEST,... ZTEST are available by default, although they may also be explicitly set. ITEST through NTEST are integers, the remainder are real.

7) The names ATEST_DP, BTEST_DP, ... ZTEST_DP are also available by default, and are type double-precision floating point, except ITEST_DP through NTEST_DP which are long (8 byte or 64 bit) integers. (As of Version 5.4, the distinction between single and double precision was eliminated since all variables now use 64 bit words. Therefore, the "_DP" variables can be used interchangeably with the original set of 26 variables. They merely represent 26 more ready-made variables, for a total of 52.)

8) The names ATEST_SP, BTEST_SP, ... ZTEST_SP are also available by default, and are type 4 byte (32 bit) floating point, except ITEST_SP through NTEST_SP which are 4 byte (32 bit) integers. (These variables are useful as arguments to old user subroutines for which the internal type was declared as REAL*4 or INTEGER*4.)

The only input format allowed for global user constants is "VN,dv" pairs, separated by commas or equal signs, that may appear anywhere between columns 2 and 1000. Pairs must be complete on a line. An example of a global USER DATA block is as follows:

```
HEADER USER DATA, GLOBAL
       NTEST = 10
       MTEST = 20
       OMEGA = (0.707/2.) + 1.0
       PI = pi
```

Floating point data values may be defined in terms of any valid expression, which may or may not include registers (Section 2.8). Only an integer value is treated as an integer, otherwise all numbers and expressions are treated as floating point. If the constant is an integer and a real value or expression is supplied, the result will be rounded to the nearest integer. If the constant is a real and an integer value is supplied, the result will be converted to type real.

2.9.2 Referencing Global User Constants in Logic Blocks

Global user constants may be referenced by name or redefined in any translatable or F-type statement in the logic block (Section 4.1). Given a global USER DATA block containing the variables PTEST, PTEST_DP (distinct from "PTEST"), TTEST, QTEST, RTEST, ITEST and JTEST (which are all available by default) the following are all examples of references that may appear in any of the OPERATIONS, VARIABLES n, FLOGIC n, OUTPUT or SUBROUTINES blocks:

```
RTEST = POD.G(1201)*QTEST
CALL DA11MC(1.512,TIMEM,A12,A13,PTEST,Q101)
PTEST_DP = VPS(TTEST,FLOW.FI)
DO 111 ITEST = 1, JTEST
```

References to global user constants are restricted to the logic blocks. When constant references are required in NODE, CONDUCTOR or SOURCE DATA blocks, they must be numbered constants associated with a specific submodel (as described in Section 2.10).

Refer also to Section 4.9 for information on how to automate updates using dynamic registers.

2.9.3 Registers versus Named User Constants

Named user constants are a historical feature that have been largely replaced by registers.

Like registers, named constants declare a Fortran variable as a valid name to the spell checker (described in Section 2.7.2 and Section 4.1.1.3), and make these variables available within all logic blocks (e.g., stored in common blocks and data modules). However, unlike registers, named constants cannot be used in expressions or as inputs in data blocks. (Registers, on the other hand, can even be used to define user data!)

Named constants follow a strict scheme for determining real versus integer values, based on their first letter. Any register may be declared as integer, otherwise they are assumed to be real.

Named constants still have a few uses, however. For example, they are never saved in SAVE, RESAVE, CRASH, SAVPAR, or SVPART (binary processor output) files, and are therefore useful for program control between restarted runs.

2.10 Submodel-specific (Numbered) USER DATA

Numbered user constants are a historical feature that have been largely replaced by registers. See Section 2.8 and Section 2.10.4.

User constants (again, “constants” is a historical misnomer) are all optional and are never referenced by any preprogrammed part of the processor. Rather, they are referenced and used only in other portions of the model as supporting data.

Users may or may not identify particular constants with a particular submodel, at their option. The “GLOBAL” submodel name is provided for *named constants* (e.g., ’FRED’ or ’FISH’) that are not referenced using a “real” submodel name as an identifying qualifier. Named USER DATA, described previously in Section 2.9, are primarily used for declaring user Fortran variables and for making them available in all logic blocks.

Submodel-specific (*numbered)* constants, the subject of this section, have predefined names with user-input identifiers. They may be used as alternatives to named (GLOBAL) USER DATA.

CAUTION: *The global block must precede the submodel blocks even though the two types of constants appear unrelated. See* Section 2.9 *for USER DATA, GLOBAL formats.*

2.10.1 Usage and Referencing

Numbered constants are given an arbitrary integer identifier by the user (e.g., “10”, or “80201”). The full identification of a numbered constant includes the submodel name. Thus, the integer identifiers need only be unique within a single submodel block. The same numbered constant entered under a separate and distinct submodel name can still be uniquely referenced. As each constant is read in by the preprocessor, it is assigned an internal sequence number. This sequence encompasses the entire set of numbered constants identified in all submodels. (Similar treatment applies to other network elements to be defined later: nodes, conductors, lumps, paths, ties, and arrays.)

User constant references, outside the USER DATA blocks, are of the forms:

```
[smn.]XKn  or  [smn.]XK(n)      $ real constant
[smn.]Kn   or  [smn.]K(n)       $ integer constant
```

where:

`smn` . . . . . submodel name
`n`. . . . . user’s identification number (six digits maximum)

References of this type are allowed in any of the logic blocks (OPERATIONS, VARIABLES n, FLOGIC n, OUTPUT CALLS and SUBROUTINES as described in Section 4.1). Historically, they could also be used within the NODE, CONDUCTOR and SOURCE DATA blocks as allowed by certain options, but that usage is archaic and no longer documented.

User constants may be initialized with integer or floating point (real) data values or expressions. Character data is input separately in CARRAY DATA blocks, described in later sections.

The largest user constant identification number allowed is 99999999 (i.e., 8 digits).

Most advanced users will eventually need to know how numbered user constants are stored internally in the processor, although such knowledge is not strictly necessary for the casual user. Each submodel's set of numbered user constants are stored contiguously in input order, and all sets are stored contiguously in one array—in fact, integer and real constants share the same array (named K or XK) via a Fortran EQUIVALENCE statement. The preprocessor stores the starting sequence numbers associated with each submodel and the size of each set. These data allow the processor to unambiguously access any numbered user constant.

It is sometimes useful to know that the constants occupy sequential core locations in the same order as they are input. For instance, a user might access or update a series of constants using a Fortran DO loop if the appropriate sequence numbers are known. This information is available for user constants and for other sequentially stored data types in the form of user *directories*, which may be printed using an OPTIONS DATA flag. If the appropriate directory is not available, the user may employ a utility subroutine that produces the internal sequence number, given the user's ID number. (Reference Section 7: NUMTRN.) *Use of this "dynamic translation" routine is recommended not only to avoid memorizing storage sequences, but also to avoid insidious errors caused by changes to the model, which will also change the storage directories.*

Refer also to Section 4.9 for information on how to automate updates using dynamic registers.

2.10.2 Input Formats

Numbered user constants are defined using:

```
HEADER USER DATA, smn
```

where `smn` is a "real" (as opposed to "global") submodel name.

The standard input format for numbered constants is:

```
        id = dv
or      id, dv
```

where:

`id`..... integer identification number
`dv`..... integer or real data value, or expression (real)

Multiple id-dv pairs, separated by commas, may appear within the column 2-1000 data field over several lines, but each pair must be contained on a single line.

If an integer value is detected, the constant may be referred to in logic blocks as "K(id)." Otherwise, a real value should be input (for the options described in NODE, SOURCE, and CONDUCTOR DATA), and the user constant may be referred to in logic blocks as "XK(id)."

CAUTION—The program cannot "know" whether the user constant was intended to be input as a real or integer value, and whether it is accessed properly in logic. (In data blocks, "XK" and "K" forms may be used equivalently. This caution only applies to referencing numbered user constants as numbers within logic blocks.) Specifying "1=1" (integer definition) and then accessing that constant as XK1 (a real reference) will yield an unpredictable number. Similarly, specifying "1=1.0" (real definition) and then accessing that constant as K1 (an integer reference) will also yield an unpredictable number.

Note that "integer expressions" such as "10-9" are treated as if they were of type real, as are *any expression*. Applying the built-in functions *ceil* or *floor* etc. does not result in an integer value, but rather a real value that corresponds to the closest whole number.

The GEN input option allows the user to generate and initialize a group of numbered constants having the same initial value or sequentially incremented values. Two GEN formats are allowed:

```
GEN id#, #k, ik, dv
GEN id#, #k, ik, dvi, idv
```

where:

`id#` . . . . . user identification number of the first constant in the series (integer)
`#k` . . . . . number of constants to be generated (integer)
`ik` . . . . . identification number increment desired (integer)
`dv` . . . . . single data value (integer or real) or expression (real)
`dvi` . . . . . data value or expression to be assigned to initial constant id# (integer or real)
`idv` . . . . . increment to be added to data value to obtain the next consecutive data value in the set (integer or real)

Several GEN groups may appear on the same record, separated by commas. Groups must be complete within columns 2 through 1000 (i.e., not separated over two lines).

2.10.3 Input Examples

The following examples illustrate numbered constant data input:

```
HEADER USER DATA, POD1
       101=(3.485*144.)+14.7, 201 = 864 $ Standard Inputs
C Generates 10 zero value constants
       GEN 901, 10, 1, 0.
C Generates 3 constants with values 100., 150., 200.
       GEN 1001, 3, 1, 100., 50.
```

2.10.4 Registers versus Numbered User Constants

Numbered user constants are a historical feature that have been largely replaced by registers.

Like registers, numbered constants can be accessed and changed as Fortran variables in logic blocks. However, unlike registers, numbered constants cannot be used in expressions or as inputs in most data blocks. (Registers, on the other hand, can even be used to define user data!)

Numbered constants still have a few uses, however. For example, because they are owned by individual submodels rather than being global entities like registers, they can be manipulated at a submodel level. However, even if they are used there are many advantages to defining their values indirectly using register-containing expressions (“formula”).

2.11 ARRAY DATA

ARRAY DATA is analogous in function to numbered USER DATA. The difference is that ARRAY contains multi-element sets of data rather than single data values.

A related block is also available: the CARRAY DATA block, which provide a way to input character data.

2.11.1 Standard Array Data

Standard array data is entered as numbered arrays using:

```
HEADER ARRAY DATA, smn
```

where `smn` is a submodel name. The standard format for inputting arrays is:

$$\texttt{num = dv}_1\texttt{, dv}_2\texttt{, - - - dv}_n$$

where:

`num` . . . . . Array ID number

$\texttt{dv}_n$ . . . . . n^{th} data value in array, which may be an expression (and if so, the value is treated as real, not integer)

The "=" character following the array number signals the end of any previous array.*

Array data records must all be within the columns 2 through 1000 data field and may continue for as many lines as necessary. Lines of array data are automatically terminated with a comma if one is not provided by the user.

Array identification numbers up to 99999999 (i.e., 8 digits) are accepted. Refer to Section 6.19 for limits on the number and sizes of arrays.

Data values may be integer or floating point (real). To avoid difficult-to-detect errors, *mixes of integer and floating point values are not accepted* unless the user has overridden this precaution using the MIXARRAY flag in OPTIONS DATA (Section 2.7.2). Character data must be input in the CARRAY blocks, described in Section 2.11.5.

The user may reserve words in memory within an array using the SPACE input option.† Such words will be initialized to 0.0. Whenever the fragment: "`SPACE,m`" is encountered *within* the data value field of a numbered array, m data values of zero will be inserted. The value of "m" can

---

* Note that there is no END terminator required to signal the end of the array, unlike old versions of SINDA. Thus the "=" cannot be replaced by a comma as in previous versions of SINDA. Arrays from very old SINDA models, containing the END terminators, may be input if the user desires. However, *the user may not mix old and new input forms in the same block*. This is the meaning of the preprocessor error message, "MIXED ARRAY INPUT FORMATS WITHIN ONE HEADER CARD."

† This option *must* be used even with versions that have no limits on sizes of arrays.

be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change allocated space: allocations are one-time events performed by the preprocessor based on initial register values.

An alternative to the SPACE command is the ALLSAME command, which is similar but is used to allocate and fill array cells with a repeated value. Whenever the fragment "`ALLSAME, m, value`" is encountered within the data value field of a numbered array, m data cells will be inserted with the "value" entry (whether that entry is a numeric constant or expression). Again, the value of "m" can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will be ignored.

CAUTION—Because of the use of expressions in ARRAY DATA blocks, *the user **must** use commas between data values. Spaces are unacceptable as delimiters, but are not flagged as errors.* For example, "10.0 -5.0" will be interpreted as a single data value of 5.0 (=10.0-5.0) rather than two values of 10.0 and -5.0. (This represents a deviation from very old SINDA formats, in which arithmetic operations were not possible.)

CAUTION—All data meant to be treated as real numbers must have an explicitly input decimal point, or else they will be treated (and stored) as integers even in the middle of an array of real values. (All expressions are treated as reals, so integer values *within* an expression are treated as real values as well.) Both this mistake and forgetting the comma between values can lead to insidious problems. The program cannot "know" whether the array cell was intended to be input as a real or integer value, and whether it is accessed properly in logic.* Specifying "1=1,2,3" (integer definition) and then accessing the first cell as A(1+1) (a real reference) will yield an unpredictable number. Similarly, specifying "1=1.0,2.0,3.0" (real definition) and then accessing the array cell as NA(1+2) (an integer reference) will also yield an unpredictable number. Note that "integer expressions" such as "10-9" are treated as if they were of type real, as is *any expression*. Applying the built-in functions *ceil* or *floor* etc. does not result in an integer value, but rather a real value that corresponds to the closest whole number.

Because of these difficulties, by default SINDA/FLUINT does not accept arrays containing mixed integer and real data values (except bivariate arrays, which are described below). In the event such usage was intentional and the user accepts responsibility for potential misuses, the MIXARRAY flag can be specified in OPTIONS DATA (Section 2.7.2) to override these precautions.

2.11.2 Array Data Structures

The previous section presented the general rules for entering numbered array data. Specifically structured arrays are required to support a number of program options. These include specifically structured singlet arrays, doublet arrays, bivariate and trivariate arrays. The specific formats for entering these specialized arrays are presented in this section.

* In data blocks, "A" and "NA" forms may be used equivalently. This caution only applies to referencing arrays as numbers or as arguments within logic blocks. See also Section 2.11.4.

Singlet Arrays — Singlet arrays contain a sequence of data values of the same type representing various values of the same parameter. The order of the values within the array is peculiar to its particular usage. For example, the singlet array required for polynomial evaluation of a variable capacitance node (see Section 2.12.1) contains floating point coefficients in the order of increasing powers of the independent variable, as shown below:

$$P_0, P_1, P_2, \ldots, P_n$$

Doublet Arrays — This type of array, most often required for interpolation options and subroutines, contains a sequence of ordered pairs of values usually representing points on a plane curve. The general order for the data values in a doublet array is as follows:

$$X_1, Y_1,\ X_2, Y_2,\ \ldots, X_i, Y_i,\ \ldots,\ X_n, Y_n$$

where:

X . . . . . independent variable
X_i . . . . . ($i = 1,2,\ldots,n$) is strictly increasing with i.
Y . . . . . dependent variable

All interpolation options and subroutines require X and Y to be floating point numbers or expressions. A more memory-efficient method of entering X-Y curve data is available when there are a number of curves defined by dependent variable data values that each correspond to a single set of independent variable data values. This is the situation when a radiation analysis program is used to generate flux vs. time data for a large number of nodes that comprise an orbiting spacecraft. The first data value in each Q-array will be for the first time point in orbit, the second for the next, and so on. In this situation, entering the data as doublet arrays would mean that each time-value would be unnecessarily repeated in each array. When this situation exists, enter the independent variable data once as a singlet array. The dependent variable data is also entered as singlet arrays and interpolation is done with a subroutine that utilizes two singlet array references as arguments (e.g., DA11MC).

Bivariate Arrays — This type of array is used to represent a function of two independent variables: Z = f(X,Y). Data values for bivariate arrays are input in the following order:

$$n,\ X_1,\ X_2, \ldots, X_n$$
$$Y_1, Z_{11}, Z_{12}, \ldots, Z_{1n}$$
$$Y_2, Z_{21}, Z_{22}, \ldots, Z_{2n}$$
$$\vdots \qquad\qquad \vdots$$
$$Y_m, Z_{m1}, Z_{m2}, \ldots, Z_{mn}$$

where:

n . . . . . Number of X values (*integer*[*])
m . . . . . Number of Y values (this value is not input explicitly)
Z_{ji} . . . . . f (X_i, Y_j)
X,Y,Z . . . . . floating point values
X_i . . . . . (i = 1,2,...,n) is strictly increasing in i.
Y_j . . . . . (j = 1,2,...,m) is strictly increasing in j.

The value of m is not input explicitly because the value of n (input as the first data value) and the value of the integer count (generated by the preprocessor) are sufficient to define the location of any element in the array. The following ARRAY DATA cards define bivariate array number 6 which contains values for the function Z = X + Y, taken from the graph in Figure 2-2:

```
6 = 3,  1.0,3.0,5.0
        2.0,3.0,5.0,7.0
        4.0,5.0,7.0,9.0
```

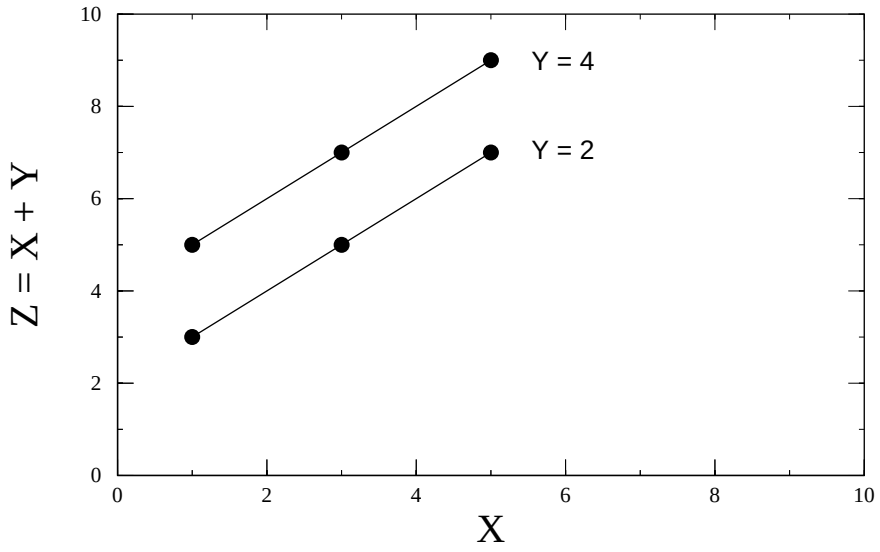


Figure 2-2 Sample Bivariate Function

Although the sample cards above are more readable, the array could just as well have been input as follows:

```
6 = 3,1.0,3.0,5.0,2.0,3.0,5.0,7.0,4.0,5.0,7.0,9.0
```

Note that there is no need for a line-ending comma.

* *Despite* containing mixed real and integer values, The MIXARRAY option is *not* required in OPTIONS DATA for this common type of array (bivariate). However, note that warnings may arise if the number of entries is wrong and the array therefore is not recognized as a bivariate array.

Trivariate Array — This type of array may be thought of as two or more bivariate arrays, where each bivariate array is associated with a third independent variable. Trivariate arrays are used to represent functions of the form F = f(X,Y,Z) for the purpose of evaluating such functions by interpolation. The data values in a trivariate array are input in the following order:

$NX_1, NY_1, Z_1, X_1, X_2, \ldots, X_n$
$Y_1, F_{11}, F_{12}, \ldots, F_{1n}$
$Y_2, F_{21}, F_{22}, \ldots, F_{2n}$ $ bivariate "sheet" for Z_1
$Y_m, F_{m1}, F_{m2}, \ldots, F_{mn}$
$NX_2, NY_2, Z_2, X_1, X_2, \ldots, X_j$
$Y1, F_{11}, F_{12}, \ldots, F_{1j}$
$Y_2, F_{21}, F_{22}, \ldots, F_{2j}$
$Y_k, F_{k1}, F_{k2}, \ldots, F_{kj}$
$NX_3, NY_3, Z_3, \ldots, \ldots, \ldots, \ldots$

A trivariate array may contain as many bivariate "sheets" as desired. The number of X and Y values in each sheet *must be specified as integers** NX and NY, respectively. NX and NY need not be the same for all sheets: All values X_i, Y_i, Z_i and F_{ij} must be floating point numbers or expressions.

The following ARRAY DATA cards define a trivariate array which represents the function F = X + Y + Z over limited ranges of the independent variables:

```
10 =    3,2,1.0,1.0,2.0,3.0     $ (array number 10)
               2.0,4.0,5.0,6.0
               3.0,5.0,6.0,7.0
        2,3,2.0,2.0,3.0
               2.0,6.0,7.0
               3.0,7.0,8.0
               4.0,8.0,9.0
        4,2,3.0,2.0,3.0,4.0,5.0
               0.0,5.0,6.0,7.0,8.0
               1.0,6.0,7.0,8.0,9.0
```

* ***Because this type of array always mixes real and integer numbers, the MIXARRAY option in OPTIONS DATA must be specified.*** Watch for preprocessor cautions regarding other arrays when MIXARRAY is selected. Also, consider a error checking run (preprocessor only, perhaps using a RETURN as the first line in OPERATIONS) in which a decimal point is temporarily placed on the integers of trivariate arrays.

2.11.3 Character Data Arrays

Character data are entered using:

```
HEADER CARRAY DATA, smn
```

Character arrays are numbered "arrays," each containing a single data value. Such data values consist of up to 1000 alphanumeric characters. The following is an example of CARRAY input:

```
HEADER CARRAY DATA, GEORGE
        120 = SAMPLE CHARACTER ARRAY
```

The data value consists of all characters following the equal sign (=) on that line. *Only one CARRAY value can be input on each line.* All character arrays are stored as 1024 characters, blank-filled on the right.

The contents of the character string cannot contain apparent references to SINDA/FLUINT translatable variables, such as "160 = A13404" (which would be confused with array data) or "444 = THREE PLUS PL2303" (which would be confused with a lump pressure). This restriction is due to preprocessor checks, and can be avoided by setting the desired string in a logic block (perhaps in OPERATIONS, see Section 2.11.5).

2.11.4 Referencing Array Data in Data and Logic Blocks

All standard numerical array data, regardless of input structure, are stored in a single Fortran array named "A." A is equivalenced to an array named "NA" so that either integer or real data values may be accessed in core without data translation problems peculiar to Fortran. An array defined in an input record as:

$$n = v_1, v_2, \ . \ . \ . \ v_m$$

resides in memory within the A array as

$$m, v_1, v_2 \ . \ . \ . \ v_m$$

where m is the integer count (i.e., the total number of v values in that array).

Array references are required by some options in the NODE, CONDUCTOR, SOURCE, and FLOW DATA blocks and are often used in the logic blocks (see Section 4.1) as well. Two types of references are required. The reference may be to the array as an entity or merely to a single element in the array. The two reference forms are

```
A(n) or An      $(Array as an entity)
```

and

```
A(n+e)          $(Accesses a data value)
```

where:

n. . . . . array identification number (integer)
e. . . . . numerical position of a specific element in the array (integer)

The A(n) form “points” to the array’s integer count in the A array. This is sufficient data for a subroutine to use the array, if the arrays structure matches the subroutine’s expectations. The A(n+e) form simply points to an element in the A array. Note that this procedure is really no different from accessing a user constant.

The following are examples of array data access forms that may appear in logic data:

```
C - - - Example 1: Subroutine call in VARIABLES 0
        CALL DA11MC(PER,TIMEM,A(120),A(131),1.0,Q(980))
```

The subroutine is supplied with references to a time array, array 120, and a Q-source array, array 131.

```
C - - - Example 2: Referencing a floating point value
        TIMEP = TIMEP + POD.A(131+6)
```

Element no. 6 of array 131 of submodel POD is accessed.

```
C - - - Example 3: Referencing an integer data value
        ISTEP = M+POD.NA(200+2)
C - - - Example 4: Referencing an integer count
        IC = POD.NA(300)
```

The integer count of array 300 will appear in IC. Note: IC = POD.A(300) is an error.

Refer also to Section 4.9 for information on how to automate updates using dynamic registers.

2.11.5 Referencing Character Array Data in Logic Blocks

Each line of character array (CARRAY) data is stored as an entry into the UCA array. The reference form in logic blocks (see Section 4.1) is:

```
UCA(n) or UCAn
```

where n is the array identification number (integer)

Each UCA entry is a left justified 1024 character string. This string can be used for any operations in which a character string is required.

2.12 NODE DATA

Nodes are fundamental SINDA network elements that describe how energy is to be stored or otherwise conserved. Four types of nodes may be defined in the NODE DATA block: *diffusion*, *arithmetic*, *boundary*, and *heater*.

Diffusion nodes have finite thermal mass or *capacitance*, and thus store and release energy. Three locations in core memory are reserved for each diffusion node in an active submodel: one location to store the temperature of the node, one to store its capacitance, and one for the heat source (if any) impressed on the node. Diffusion node data input options are provided to accommodate capacitance values which are not constant (i.e., vary with temperature, etc.)

Arithmetic nodes have zero capacitance. That is, arithmetic nodes are continually brought into a steady state heat balance according to the source applied and the heat exchange with neighboring nodes. Since there is no capacitance value to store, only two core locations are reserved for each arithmetic node: one for the temperature, and one for the impressed heat source (if any).

Boundary nodes have infinite capacitance and may not receive an impressed heat source. A single core location is reserved to store the temperature of each boundary node. The temperatures of boundary nodes are not altered by the network solution routines, but may be modified as desired by the user.

Heater nodes are used for the purpose of establishing heater power requirements. They are the same as boundary nodes except that they report the amount of heat removed or added to maintain the given temperature. This energy is calculated and reported using calls to the subroutine HNQPNT (Section 7.4.13), or just calculated (without printing) using HNQCAL (Section 7.7.1). Diffusion and arithmetic nodes may be temporarily held as heater nodes using the HTRNOD routine (Section 7.2).

2.12.1 Node Data Input

All NODE DATA records appear after:

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

where:

`smn` ..... submodel name
`fac` ..... capacitance multiplying factor
`tcon` ..... temperature conversion flag. Note that `fac` must be input in order to input a value for `tcon`. If `fac` is not otherwise needed, supply a value of 1.0, such as: HEADER NODE DATA, MYMOD, 1.0, FTC

NWORK . . . . . sparse matrix package workspace size for this submodel. *Ignore this input* unless directed to specify it by a processor error message, or unless memory requirements become excessive for large models (see Appendix F). Otherwise, the processor will abort if this value is input and is too small. NWORK specifications must be at the end of the line after `fac` and `tcon` inputs, if any.

`[ ]` . . . . . indicates optional arguments

The submodel name is any identifying name, up to 32 characters. It must start with an alphabetic character (A through Z). `tcon` is a code word used to convert the temperature on the NODE DATA records from one units base to another, if desired. Allowable `tcon` options are:

```
FTC ......Fahrenheit to Centigrade
CTF ......Centigrade to Fahrenheit
FTR ......Fahrenheit to Rankine
RTF ......Rankine to Fahrenheit
CTK ......Centigrade to Kelvin
KTC ......Kelvin to Centigrade
FTK ......Fahrenheit to Kelvin
KTF ......Kelvin to Fahrenheit
CTR ......Centigrade to Rankine
RTC ......Rankine to Centigrade
RTK ......Rankine to Kelvin
KTR ......Kelvin to Rankine
```

Please note that it is the user's responsibility to ensure that all submodels are consistent as to temperature and other dimensional parameters. Temperature units are controlled by `tcon` and control constant ABSZRO (Section 4.3: Control Constants).

Examples:

```
HEADER NODE DATA, MYMOD
HEADER NODE DATA, MYMOD, NWORK = 5000
HEADER NODE DATA, OTHMOD, 2.0          $ FAC = 2.0
HEADER NODE DATA, YETMO, 1.0, RTC
HEADER NODE DATA, AUNMAS, 1.0, RTC, NWORK = 5000
```

There are several signals or codes which may be used on NODE DATA records. The codes provide options which enable the user to input a large variety of nodes in a simple, convenient manner. For all options, the following points apply:

1) Diffusion nodes must be given a positive node number and a positive capacitance.

2) Arithmetic nodes must be given a positive node number and a negative (*not zero*) capacitance. (The negative capacitance value acts as a signal that the node is of the arithmetic type.)

3) Boundary nodes must be given a negative node number and a non-negative capacitance (e.g., 0.0). The capacitance value is ignored but is present for the sake of consistency, and the negative sign on the node number serves as a flag to signify that the node is of the boundary type.

4) Heater nodes must be given a negative node number and a negative capacitance.

5) An initial temperature must be specified for all nodes, regardless of type.

6) Node ID numbers may be as large as eight digits (e.g., 99999999).

7) Whenever expressions are used to define capacitances, the sign of the *current* value (during preprocessor execution) of the expression is used as a flag for determining node type.

8) References to processor variables within expressions may refer to "#this" as the node identifier.

Table 2-7 summarizes the NODE DATA input options which are available to the user. Before proceeding to a detailed discussion of each option, two points should be clarified: (1) *impressed heat sources are not input with node data*; they are input in the SOURCE DATA block, in which case they are transferred to the source locations automatically when needed; and (2) nodal capacitances and heat sources are always accessible to the user in the logic blocks, and hence, a "constant capacitance value," from the standpoint of NODE DATA input, need not be held constant during the entire course of a solution.

Table 2-7 Summary of NODE Data Input Options

OPTION (CODE)	NODE TYPE				DESCRIPTION
	D	A	B	H	
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

While `tcon` and `fac` may use registers in their defining expressions, these values are treated as permanent constants and are not updated by the dynamic register options described in Section 4.9. This does not imply that the temperatures of boundary nodes nor the capacitances of diffusions nodes defined using these factors are constant, just the factors themselves. Furthermore, expressions defining these factors cannot reference processor variables, which have no meaning during preprocessing.

Standard Option ("3 blank" card code) — The simplest NODE DATA record utilizes the blank code. Records using this code must contain three data values for each node, as follows:

```
N#,Ti,C                          $(basic format)
5,70.0,0.75*2.0                  $ example 1
6,80.0,-1.0                      $ example 2
-9,90.0,0.0                      $ example 3
-7,70.0,0.0,8,85.0,-1.0          $ example 4
```

where:

`N#` . . . . . Node ID number assigned by the user (integer data value)
`Ti` . . . . . Initial temperature of the node (floating point data value or expression)
`C`. . . . . Capacitance of the node (floating point data value or expression)

The order and type of the three data values must always be exactly as shown. The rules determining the type of input node are as follows: if N# and C are positive, then the triplet defines a diffusion node; if N# is positive and C is negative, an arithmetic node is defined; if N# is negative and C is zero, then a boundary node is defined. More than one node may be defined on a single card by placing more than one set of three data values on it.

Defining the C value using formula (expressions containing registers and/or processor variables, such as "CP*DENSITY*VOLUME") makes the model readily adjustable.

Caution: Changing the values of registers in expressions defining capacitances can inadvertently change the *type* of node (during preprocessing only) if the sign of the resulting value toggles from positive to negative or vice versa, or if the resulting value toggles from zero to nonzero or vice versa.

Example 1 creates a diffusion node with a node number of 5, an initial temperature of 70.0°, and a capacitance of 1.5. Example 2 creates arithmetic node number 6 with an initial temperature of 80.0°. Example 3 creates boundary node number 9 *(not -9: the negative sign serves only as a flag!)* with an initial temperature of 90.0°. Example 4 shows how the data values for two nodes may be placed on the same record. The two nodes created in example 4 are as follows: boundary node number 7 at a temperature of 70.0°, and arithmetic node number 8 at a temperature of 85.0°.

Caution: If an expression containing a register is used to define the initial temperature ("Ti") of diffusion or arithmetic nodes, changes to theses registers will not affect the temperature of the node during processor execution, since the temperature was presumed to be only an initial condition. Furthermore, processor variables cannot be used to define the initial temperature of arithmetic or

diffusion nodes. Temperatures of boundary and heater nodes are fundamentally different since they are not SINDA predictions, but rather user inputs. Not only can processor variables be used in the definitions of their initial temperatures, but changes to the registers used in their definition will also affect the temperature of boundary and heater nodes.

GEN Option — The GEN option is used to generate and input a group of nodes, each having the same initial temperature and the same capacitance. This option requires five data values for each group of similar nodes, as follows:

```
GEN N#,#N,IN,Ti,C              $ (basic format)
GEN -20,3,5,70.0,1.2           $ example 5
```

where:

`N#`..... ID number of the first node to be generated (integer)

`#N`..... Total number of nodes to be generated (integer). The value of #N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: node generation is a one-time event performed by the preprocessor based on initial register values.

`IN`..... Increment to be added to the current node number to form the node number of the next node (non-zero integer)

`Ti`..... Initial temperature of all nodes (floating point or expression)

`C`..... Capacitance of all nodes (floating point or expression)

If C and N# are positive, a group of diffusion nodes will be generated. If C is negative a group of arithmetic nodes is produced. If N# is negative, then the nodes will all be of the boundary type.

It is important to remember that the negative sign on the N# serves only as a flag to identify a boundary node during input. For reference purposes, the negative sign is deleted. Hence, example 5 will generate three boundary nodes having a temperature of 70.0°, with the node numbers being 20, 25, and 30 (not -20, -15, and -10). The minus sign on the initial N# signifies that the entire group will be boundary nodes. A minus sign in the initial N# and the initial C signifies that the entire group will be heater nodes.

Defining the C value using formula (expressions containing registers and/or processor variables, such as “CP*DENSITY*VOLUME”) makes the model readily adjustable.

Caution: Changing the values of registers in expressions defining capacitances can inadvertently change the *type* of node (during preprocessing only) if the sign of the resulting value toggles from positive to negative or vice versa, or if the resulting value toggles from zero to nonzero or vice versa.

The eight-data-value GEN option, like the CAL option, is obsolete. It is available but not documented.

Caution: If an expression containing a register is used to define the initial temperature ("Ti") of diffusion or arithmetic nodes, changes to theses registers will not affect the temperature of the node during processor execution, since the temperature was presumed to be only an initial condition. Furthermore, processor variables cannot be used to define the initial temperature of arithmetic or diffusion nodes. Temperatures of boundary and heater nodes are fundamentally different since they are not SINDA predictions, but rather user inputs. Not only can processor variables be used in the definitions of their initial temperatures, but changes to the registers used in their definition will also affect the temperature of boundary and heater nodes.

Automated Variable Capacitance Options — Temperature-dependent capacitance can be encoded into the expression defining the capacitance itself. For example, if the Cp of node 22 obeyed a linear relationship with temperature, Cp(T) = Cpdt*(T-Tref) + Cpref:

```
22, Tinit, density*volume*(Cpdt*(T#this-Tref)+Cpref))
```

However, more convenient means are available for describing the temperature-dependence of a single material and then reapplying that relationship to multiple nodes. The rest of this section describes these features.

As will be explained in Section 4.1.2, the network solution routines call for the execution of the operations in the VARIABLES 1 block prior to performing each iteration on the heat transfer equations. VARIABLES 0 is called after each time step update in a transient solution. Using this feature, the user can set time or temperature variable capacitances by specifying VARIABLES 0 or VARIABLES 1 operations which would update values in the node capacitance table. The new capacitance could be calculated using the appropriate algorithm, or interpolation subroutine and independent variable. Though straightforward, this approach tends to be rather cumbersome, especially for those users with limited programming experience.

The automated options cause temperature-dependent capacitances to be calculated wholly within the network solution routines, at the end of the VARIABLES 1 operations, and therefore relieve the user of the task of preparing and inputting capacitance computation algorithms. As specified by the user, one of two methods is used to update the current value of a temperature-dependent capacitance: (1) interpolation, and (2) polynomial evaluation. For interpolation, the user must input, in an ARRAY DATA block, a doublet array of (temperature, capacitance) ordered pairs representing points on the curve of T vs. C. The capacitance is calculated by performing linear interpolation on this array using the current temperature of the node as the independent variable. For polynomial evaluation, the user must input a singlet array of coefficients (P_0, P_1, P_2, ..., P_n) in an ARRAY DATA block. The capacitance is then calculated by evaluating the n^{th} order polynomial in T (i.e., P_0+P_1*T+P_2*T**2+...+P_n*T**n). These two methods enable the automated options to be applicable to a vast majority of the variable capacitance nodes encountered in modeling real systems.

To enable the same array of points or coefficients to be used for all nodes made of the same material, all automated options cause the computed capacitance to be multiplied by a factor before being inserted in the capacitance table. In this way, for example, the array can be used to compute $\rho*C_p$, and the multiplying factor can be used to specify the volume. In another application, the multiplying factor might be used to scale the capacitance to some set of consistent units.

The interpolation options (except BIV*) require a doublet array of temperature vs. capacitance to be input in an ARRAY DATA block. (Capacitance is used loosely in this context. The array could contain, for example, values of temperature vs. specific heat, as long as the associated multiplying factor had units of mass.) The array must contain an even number of floating point data values, as follows:

$$T_1, C_1,\ T_2, C_2,\ \ldots,\ T_i, C_i,\ \ldots,\ T_n, C_n$$

The temperature values, T_i, must be strictly increasing with i. If the node temperature at some time is less than T_1, then C_1 will be used in lieu of performing an extrapolation; if the temperature is greater than T_n, then C_n will be used.

The polynomial options require that an array of coefficients be input in an ARRAY DATA block. The array must consist solely of n+1 floating point values, where n is the order of the polynomial. The coefficients should be entered in the array in the order of increasing powers of temperature (e.g., the first data value is the constant term, P_O, and the $(n+1)^{th}$ data value is the coefficient, P_n, of T**n). For both classes of options, the arrays are referenced by using the form An or smn.An where n is the array number.

SIV and SPV Options — These options allow the user to define single nodes having temperature varying capacitances. The SIV option assumes that the referenced array is to be used for linear interpolation, whereas the SPV option assumes that the array contains polynomial coefficients. Both options require four data values for each node, as follows:

```
SIV N#,Ti,AP,F                  $ (basic format)
SPV N#,Ti,AC,F                  $ (basic format)
SPV 25,70.0,A8,0.0001           $ example 6
SIV 25,70.0,A9,0.0001           $ example 7
SIV 25,70.0,A9,area*length      $ example 8
```

where:

- N#..... Node ID number (integer)
- Ti..... Initial temperature of node (floating point or expression)
- AP,AC..... Reference to an array of T vs. C points, or polynomial coefficients, respectively
- F..... Multiplying factor input as a floating point data value or expression

The following example is presented to clarify the usage of these two options. Consider a temperature varying capacitance given by the equation:

$$C = (0.0001) \times (0.005 \times T^2 + 1.0)$$

* The BIV option requires a bivariate array of time and temperature vs. capacitance.

This equation represents the curve shown in Figure 2-3. To use the SPV option to define node 25, which has a capacitance given by the equation above, the user would prepare a NODE DATA record as shown in example 6. The referenced array of polynomial coefficients, user array number 8, would be entered in an ARRAY DATA block, as follows:

```
8 = 1.0,0.0,0.005
```

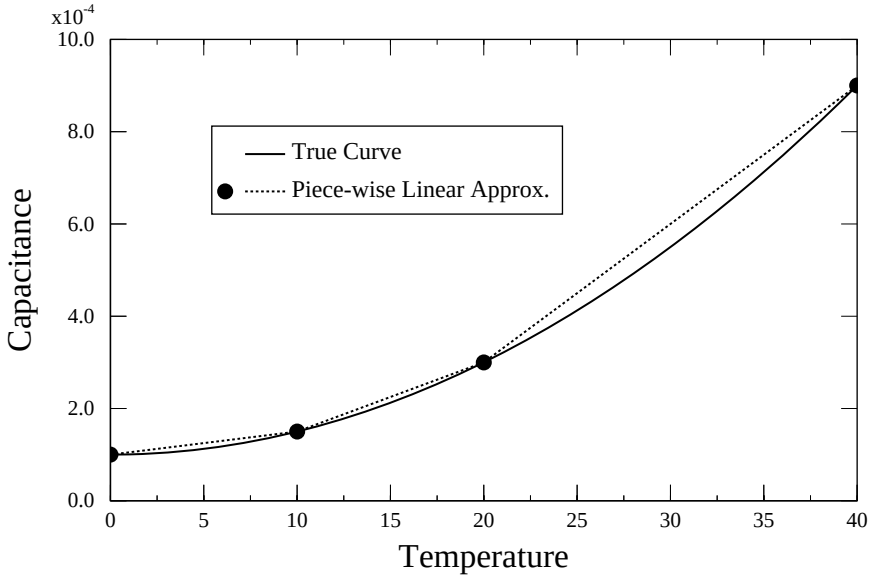


Figure 2-3 Curve of Temperature-Varying Capacitance

To use the interpolation option, SIV, to define this node, the user would prepare a node data record as shown in example 7. The referenced array of T vs. C points, A9, represents a piecewise linear approximation of the curve in Figure 2-3, and would be entered in the appropriate ARRAY DATA block as follows:

```
9 = 0.0,1.0,10.0,1.5,20.0,3.0,40.0,9.0
```

Examples 6 and 7 show that the multiplying factor, 0.0001, was entered directly on the NODE DATA record as a floating point data value. This factor could also have been defined using registers or register-containing expressions (“formulas”), as shown in example 8.

A common usage of these options is to input an array of $\rho * C_p$ vs. T (i.e., density times specific heat versus temperature) *once* for each material used in the model, and then to specify the volume of each node as the multiplying factor in each node declaration. Defining this volume using formulas makes the model readily adjustable.

SIM and SPM Options — These two options combine the features of the SIV and SPV options with those of the GEN option. That is, they allow the user to generate and input a group of nodes all having the same initial temperature and the same temperature varying capacitance. The SIM option generates nodes whose capacitance will be evaluated by interpolation, and the SPM option generates nodes whose capacitance will be evaluated from a polynomial. Six data values are required to define each group of nodes, as follows:

```
SIM N#,#N,IN,Ti,AP,F          $ (basic format)
SPM N#,#N,IN,Ti,AC,F          $ (basic format)
SIM 25,3,5,70.0,A9,area*length $ example 10
```

where:

N#..... ID number of the first node (integer)

#N..... Total number of nodes to be generated (integer). The value of #N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: node generation is a one-time event performed by the preprocessor based on initial register values.

IN..... Increment to be added to the current node number to form the number of the next node (non-zero integer)

Ti..... Initial temperature of all nodes (floating point or expression)

AP,AC..... Reference to an array of points of T vs. C, or polynomial coefficients, respectively. An or smn.An form, where n is the array number)

F..... Multiplying factor (floating point data value or expression)

Example 10 defines three nodes, numbers 25, 30, and 35, whose capacitance will be computed by interpolating on array 9, with the results multiplied by the expression “area*length.”.

DIV and DPV Options — These options allow the user to define a node which is made of two materials, each having different temperature varying capacitance properties. Both require an array and a multiplying factor for each of the two materials. For the DIV option, the result of interpolating on the first array, times the first multiplying factor, is added to the result of doing likewise with the second array and the second multiplying factor. The action of the DPV option is similar, except that the arrays are assumed to contain polynomial coefficients. The formats are shown below:

```
DIV N#,Ti,AP1,F1,AP2,F2       $(basic format)
DPV N#,Ti,AC1,F1,AC2,F2       $(basic format)
DIV 40,70.0,A8,factr,A6,2.0   $ example 11
```

where:

N# . . . . . Node ID number (integer)
Ti . . . . . Initial temperature (floating point or expression)
AP1,AP2. . . . References to arrays of T vs. C points (An or smn.An form, where n is the array number)
AC1,AC2. . . . References to arrays of polynomial coefficients (An or smn.An form)
F1,F2 . . . . . Multiplying factors (floating point data value or expression)

Example 11 defines node 40, whose capacitance will be computed as the sum of interpolation on array 8 times “factr” plus interpolation on array 6 times 2.0.

If one of the materials does not have a temperature varying capacitance, the user may substitute a floating point data value or expression for one of the array references of the NODE DATA card. This substitute value will then be used in place of the result of interpolation (or polynomial evaluation) on the corresponding array. This feature makes the following alternate record formats possible:

```
DIV N#,Ti,SUB,F1,AP2,F2       $(alternate format)
DPV N#,Ti,SUB,F1,AC2,F2       $(alternate format)
DIV N#,Ti,AP1,F1,SUB,F2       $(alternate format)
DPV N#,Ti,AC1,F1,SUB,F        $(alternate format)
DIV 35,70.0,1.0,factr,A9,0.003$ example 12
```

where SUB is a floating point data value or expression.

Example 12 illustrates the case where one of the two materials has a constant capacitance. Since the capacitance of the first material will be calculated as 1.0*factr, it would be quite acceptable to alter the value of the register “factr” at some point in an operations block, and thereby effect a change in the capacitance of this “constant capacitance” material.

DIM and DPM Options — These two options combine the features of the DIV and DPV options with those of the GEN options. They are used to generate and input a group of nodes, all of which have the same initial temperature and all of which are of the same two materials having temperature varying capacitances. The DIM option causes the capacitance for the two materials to be evaluated by interpolating on the referenced arrays, and the DPM option computes the capacitances from polynomial coefficients supplied in the arrays. As for the other dual material options, multiplying factors are applied to the results of the calculations on the arrays. Eight data values are required to define each group of nodes, as follows:

```
DIM N#,#N,IN,Ti,AP1,F1,AP2,F2         $(basic format)
DPM N#,#N,IN,Ti,AC1,F1,AC2,F2         $(basic format)
DPM 30,3,5,70.0,A8,0.1,A6,area*length$ example 13
```

where:

N#..... ID number of the first node (integer)
#N..... Total number of nodes to be generated (integer)
IN..... Increment to be added to the current node number to form the node number of the next node (non-zero integer)
Ti..... Initial temperature of all nodes (floating point or expression)
AP1,AP2 ... References to arrays of points of T vs. C (An or smn.An form, where n is the array number)
AC1,AC2 ... References to arrays of polynomial coefficients (An or smn.An form)
F1,F2..... Multiplying factors (floating point data value or expression)

Example 13 defines 3 nodes, numbers 30, 35, and 40, each made of two materials which have a temperature varying capacitance. Polynomial coefficients for the first material are supplied in array 8, and coefficients for the second material are supplied in array 6.

As for the DIV and DPV options, if one of the materials has a constant capacitance, the array reference for that material may be replaced by a floating point data value or expression. The value so supplied or referenced will be used in place of the result of interpolations or polynomial evaluation on the corresponding array. This feature permits the following formats:

```
DIM N#,#N,IN,Ti,SUB,F1,AP2,F2          $(alt format)
DPM N#,#N,IN,Ti,SUB,F1,AC2,F2          $(alt format)
DIM N#,#N,IN,Ti,AP1,F1,SUB,F2          $(alt format)
DPM N#,#N,IN,Ti,AC1,F1,SUB,F2          $(alt format)
```

where SUB is a floating point data value or expression.

BIV Option — The BIV options enables the user to define a node having a capacitance which is a function of both time and temperature. The format for such a node is similar to that for the SIV option, and requires four data values for each node, as follows:

```
BIV N#,Ti,AB,F                  $ (basic format)
BIV 55,70.0,A2,volume/10.0      $ example 14
```

where:

N#..... Node ID number (integer)
Ti..... Initial temperature of the node (floating point or expression)
AB..... Reference to a bivariate array of temperature and time vs. capacitance. (An or smn.An form)
F..... Multiplying factor (Floating point data value or expression)

Examples 14 defines node 55 whose specific heat, as a function of time and temperature, is supplied in bivariate array number 2. Section 2.11.2 contains a description of the structure of a bivariate array. In this application, the X independent variable should be temperature, the Y independent variable should be mean time, and the Z dependent variable should be capacitance. When

evaluating a node defined with the BIV option, the network solution routines will perform a bivariate interpolation on the referenced array. Using the current temperature of the node and the mean time for the current computation interval (control constant TIMEM) as the values of the independent variables.

2.12.2 Referencing Node Data in Logic Blocks and Expressions

Each node is assigned an arbitrary submodel name and identification number by the user in the NODE DATA blocks. As each node is accepted by the preprocessor, it is renumbered in sequence. The user assigned identification number is called the *actual*, or user node identification number, and the preprocessor assigned sequence number is called the *relative* number. There is no restriction as to what order the four node types appear in the input file. After sorting by the preprocessor, they will be in diffusion, arithmetic, heater, and boundary order within each submodel. This is done for efficiency in solution routine execution and memory utilization.

Although the Fortran subroutines resulting from the translation of the SINDA logic blocks (see Section 4.1) use the relative numbering system to access the node tables, it would be tedious and very error prone for the user to do so directly. Hence, the preprocessor maintains a table of actual vs. relative node numbers, and performs the conversion from the former to the latter whenever a temperature, capacitance or source table reference is encountered in the logic blocks. These tables may be accessed within logic blocks using the NODTRN routine. The node tables are referenced by the following forms:

```
smn.Tn = temperature
smn.Qn = source
smn.Cn = capacitance
```

where n is the actual (user specified identification) node number

By making the user ID numbering system available outside the NODE DATA block, the preprocessor relieves the user of the burden of keeping track of the relative input order of the nodes. For example, the temperature of node number 6 in submodel SUBN is referenced by SUBN.T6, its capacitance by SUBN.C6, and its source by SUBN.Q6, regardless of the relative position of these quantities in their respective tables.

The smn.X(n) format is the “long” or complete reference form. This may be shortened to simply Xn when the submodel name is being supplied by an argument in the preceding header record. If the reference is to a T, Q or C outside the submodel specified in the header record, then the long form is required. The OPERATIONS block is used for initialization and its logic relates to the entire model (as opposed to a single submodel) thus, the long form is the only one allowed in OPERATIONS and similar global blocks (PROCEDURE, SOLOUTPUT, SOLOGIC, SUBROUTINE), unless DEFMOD is used.

2.12.3 Backward Compatibility: the Kn and XKn option

[This section is intended only for users of older versions of the code.]

NODE DATA accepts numbered user constants (K and XK) as part of the definition of the capacitance term in some options. **This usage is historic and has been replaced by the use of registers and expressions, as explained in Section 2.8 and Section 4.9. Usage of this feature is discouraged: older models should be updated in the event that this usage becomes unsupported.** To support this former usage, if no arithmetic expressions are used in combination with user constants (e.g., only the Kn or XKn appear as the input), then the old fashioned methods are used to assure backward compatibility. However, if any multiplier is used (e.g., "3.0*XK3") or any other operation, then the modern dynamic expression handling methods are used instead.

2.12.4 Node Data Updating

In addition to updates dictated by the use of registers and processor variables used within expressions, the time- and temperature-dependent capacitance data inputs defined in the NODE DATA blocks are actually applied (that is, calculated and stored in the capacitance locations) at the end of the VARIABLES 1 logic block for that submodel (See Section 4.1.2). These updates are performed by calls to subroutine CVTEMP. The preprocessor normally inserts this call subsequent to any user-supplied logic in the VARIABLES 1 block. This gives the user an opportunity to update any registers or arrays that may affect the node data calculations before they are performed. If user intervention is required *after* the CVTEMP call, the user may supply a CALL CVTEMP('smn') statement anywhere in the VARIABLES 1 input and it will not be overridden by a preprocessor-inserted call.

The SOURCE and CONDUCTOR DATA options, which will be described next, have equivalent time- and temperature-dependent update procedures using routines QVTIME, QVTEMP, GVTIME, and GVTEMP after VARIABLES 0 as well as VARIABLES 1. However, unlike those options, NODE DATA has no purely time-dependent options, and therefore has no analog to QVTIME and GVTIME—there is no "CVTIME."

Refer also to Section 4.9 for information on how to automate updates using dynamic registers.

2.13 SOURCE DATA

SOURCE DATA blocks provide the user with a convenient means for defining heat sources or sinks which are to be impressed on the nodes. Source must always be input in units of *energy per unit time*. A positive source means energy *input* to the node.

All SOURCE DATA records appear following a header record:

```
HEADER SOURCE DATA, smn
```

where `smn` is a submodel name.

The user should note that SOURCE DATA is really defining a node attribute, so there must be strict correspondence between the full node identification (submodel name plus node number) in all records within all the NODE DATA and SOURCE DATA blocks.

2.13.1 Source Data Input

The SOURCE DATA input options and their associated 3-character codes are summarized in Table 2-8.

Table 2-8 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.

References to processor variables within expressions may refer to “#this” as the node identifier to which the source applies (see also Section 2.13.3, including the warning in that section regarding the resetting of nodal Q terms).

Standard Option (3 blanks) *— This option impresses a constant source on a single node. The input format is as follows:*

```
N#,Q                          $ (basic format)
4,1.8, 6, 4.0                 $ (example 1)
4,fred, 6, barney+fred        $ (example 2)
```

where:

N#..... ID number of a node defined in a NODE DATA block (integer).
Q..... The value of the source (floating point data value or expression)

Example 1 will cause a heat rate of 1.8 to be impressed on node number 4, and the heat rate of 4.0 to be impressed on node number 6. Example 2 portrays more modern usage wherein the sources are defined indirectly as a function of register values (and perhaps processor variables).

In order to enforce the correlation between NODE DATA and SOURCE DATA, the form smn.N (i.e., a reference to a node in another submodel) is not allowed in any SOURCE DATA format.

GEN Option *— The GEN options allows the user to impress the same heat source on a group of several nodes. The format is as follows:*

```
GEN N#,#N,IN,Q          $(basic format)
GEN 6,3,1,4.3           $(example 2)
```

where:

N#..... ID number of the first node to receive the source (integer)
#N..... The total number of nodes to receive this source (integer)
IN..... The increment to be added to the current node number to form the next node number (non-zero integer)
Q..... The value of the source (floating point data value or expression)

Example 2 will cause a heat rate of 4.3 to be impressed on nodes 6, 7, and 8.

Automated Variable Source Options *— Time and temperature-dependent sources can be imposed on a node by encoding the dependency into the expression defining the source itself. For example, if the source on node 22 where to be zero at times earlier than 1.0, and 50.0 units after 1.0, then using the SINDA control constant "timen" as the value of time ("time now"):*

```
22, (timen>1.0)? 50.0 : 0.0
```

However, more convenient means are available for describing the time- or temperature-dependence of a single environment and reapplying that relationship to multiple nodes. The rest of this section describes these features.

The automated options provide for operations which will automatically evaluate the current value of a heat source which varies with time or temperature (or both), and place this value in the appropriate source location.

All options cause the source to be evaluated by performing linear interpolation on arrays of values which represent points on a curve of time or temperature vs. heat rate. If the heat source is temperature dependent, the values are entered in the ARRAY DATA block in the following doublet array format:

$x_1,y_1,\ x_2,y_2,\ \ldots\ xi,yi,\ \ldots\ x_n,y_n$

where:

y_i . . . . . Dependent variable - heat rate

x_i . . . . . Independent variable - temperature; x_i (i = 1, 2, ..., n) is strictly increasing with i

x_i,y_i . . . . . Floating point numbers or expressions

and are referenced in the SOURCE DATA block by using the form: Am or smn.Am, where m is the array identification number.

For temperature-dependent sources, the value of the independent variable used for interpolation will be the current temperature of the node receiving the source. If the value of the independent variable is less than x_1, then y_1 will be used in lieu of performing an extrapolation. Similarly, if the value is greater than x_n, then y_n will be used. To provide further versatility, the value of the heat rate found from interpolation will be multiplied by a factor before being entered in the source table. This multiplying factor can assume any significance that is convenient for the user. If the multiplying factor has any units, the product of these units and those of the dependent variables in the array must be units of heat rate.

SIV Option — *The SIV option allows the user to specify a temperature varying source. The record format is as follows:*

```
SIV N#,A,F              $(basic format)
SIV 8,A4,1.0            $(example 3)
```

where:

N# . . . . . ID number of the node which will receive this source (integer)

A. . . . . Reference to doublet array of temperature vs. heat rate, of the form: An or smn.An, where n is the array number

F. . . . . Multiplying factor (floating point data value or expression)

The following discussion, and example 3 above, illustrates the use of the SIV option.

Assume that the heat rate defined by the graph in Figure 2-4 is to be impressed on node number 8. This graph can be represented by the following doublet array, which would be entered in an ARRAY DATA block:

```
4 = 0.0,0.0,10.0,0.1,20.0,0.05,30.0,0.05,40.0,0.0
```

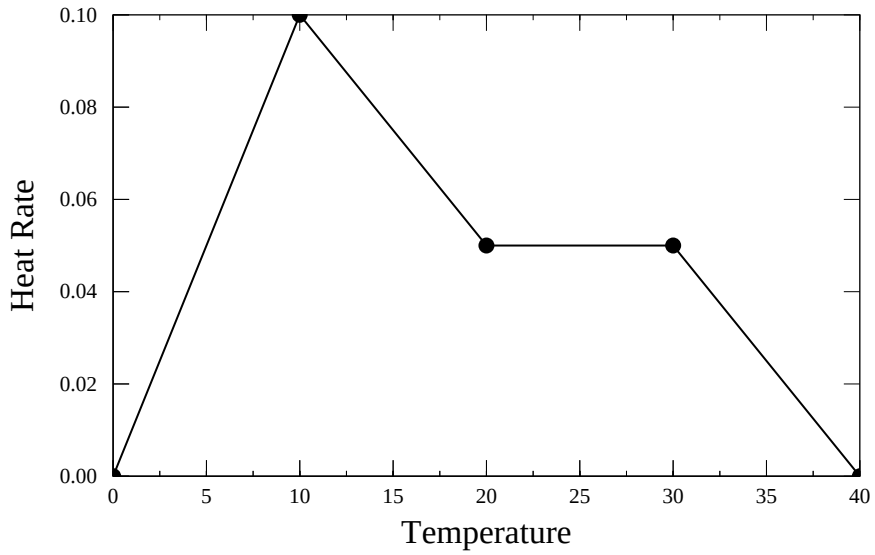


Figure 2-4 Temperature-Varying Heat Rate

NOTE: The required heat source is defined in the source data by the record shown in example 3. Since the referenced array (array number 4) does not need to be scaled or modified in any way, the multiplying factor is merely given as the floating point data value 1.0.

TVS Option — *The TVS option allows the user to apply a time varying source to a node. The difference between the TVS and SIV options is that the array referenced with the TVS option should be entered in the ARRAY DATA block and must be in the singlet form. There is a restriction that both the time array and q-arrays must be in the same data block. The ARRAY DATA format is as follows:*

```
nt     =  t1,t2,t3 - - - - tn
nq     =  q1,q2,q3 - - - - qn
```

where:

`nt`..... Time array number
`ti`..... Time values (independent variable, values strictly increasing with i)
`nq`..... Heat rate array number
`qi`..... Heat rate values (dependent variable)
`ti,qi`..... Floating point numbers or expression

The TVS SOURCE DATA input format is as follows:

```
TVS N#, At, Aq, F              $(basic format)
TVS 16,A1,A9,shunt             $(example 4)
```

where:

N# . . . . . .ID number of the node which will receive this source (integer)
At . . . . . .Reference to singlet time array of the form: An or smn.An, where n is the array number
Aq . . . . . .Reference to singlet heat source array same form
F. . . . . .Multiplying factor (floating point data value or expression)

Example 4 causes the results of interpolating on arrays 1 and 9, (using the mean time for the computation interval (TIMEM) as the independent variable) multiplied by the value in register "shunt" to be inserted in the source location for node number 16 (Q16).

TVD Option — *The TVD option is used to specify a heat source which is the sum of two separate, time-varying sources. The user must supply two singlet time vs. heat rate arrays, in an ARRAY block, and a multiplying factor for each of the two sources, as follows:*

```
TVD N#,At,Aq,FA,Bt,Bq,FB       $(basic format)
TVD 3,A1,A5,K2,A1,A6,2*factor  $(example 5)
```

where:

N# . . . . . .ID number of the node which will receive this source (integer)
At,Aq
Bt,Bq . . . . . .References to singlet arrays of time vs. heat rate for source A and source B, respectively of the form: An or smn.An, where n is the array number
FA,FB . . . . . .Multiplying factors to be applied to the results of interpolation. (floating point data values or expressions)

Example 5 defines a source to be impressed on node 3, which will be evaluated by taking the sum of the interpolation result from array 5 times twice the value of register "factor", and the interpolation result from array 6 times the value of user constant 4. Both interpolations use time array 1 for the independent variable data.

DTV Option — *This option enables the user to specify a source which is the sum of a time varying source and a temperature varying source. The DTV option requires separate doublet arrays for time vs. heat rate and temperature vs. heat rate, and represents the summation Q = f(t) + g(T). The record format is as follows:*

```
DTV N#,At,Ft,AT,FT                    $(basic format)
DTV 26,A14,timen/3600.0,A16,pi*diam   $(example 6)
```

where:

N#..... ID number of the node which will receive this source (integer)
At..... Reference to a doublet array of time vs. heat rate, (form: An or smn.An, where n is the array number)
Ft..... Multiplying factor to be applied to the result of interpolation on array At (floating point data value or expression)
AT..... Reference to a doublet array of temperature vs. heat rate, form: An or smn.An, where n is the array number
FT..... Multiplying factor to be applied to the result of interpolation on array AT (floating point data value or expression)

Example 6 defines a source, to be impressed on node 26, which will be evaluated at each iteration, as the sum of a time varying component (calculated from array 14 and "timen/3600.0") and a temperature varying component (calculated from array 16 and "pi*diam").

The DTV option may be used to define a source which is the sum of (1) a time varying source and a constant source, or (2) a constant source and a temperature varying source. (Case 1 is equivalent in effect to the alternate form for the TVD option, but is also made available to the DTV option for the sake of consistency.) Case 1 is accomplished by replacing the array reference, At, with a floating point data value. Case 2 is accomplished by replacing the array reference, AT, with a floating point data value. In either case, the value of the constant source is taken as the product of the data value or expression and its associated multiplying factor. The alternate formats are as follows:

```
DTV N#,SUB,Ft,AT,FT          $(case 1)
DTV N#,At,Ft,SUB,FT          $(case 2)
```

where SUB is a floating point data value or expression.

PER Option — *The PER option in the SOURCE DATA block provides for automated cyclic interpolation of time varying heat rates. The record format is as follows:*

```
PER N#,At,Aq,F,D,P
PER 12,A1,A5,1.0,.5,1.0
PER 13,A1,A5,1.0,.75,1.0      $(example 7)
PER 14,A1,A5,1.0,0.,1.0
PER 15,A1,A5,1.0,.25,1.0
```

where:

N# . . . . . Node ID number (positive integer)

At,Aq . . . . . Reference to singlet arrays of mean time vs. heat rate points (integer) The time array must start at zero. Reference form is An or smn.An where n is the array number

F. . . . . Multiplying factor (floating point data value or expression)

D. . . . . Phase displacement as a decimal fraction of the total period. (Must be a floating point literal or expression with a value between 0.0 and 1.0, floating point value or 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 At 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.

Example 7 defines cyclic sources to be impressed on nodes 12, 13, 14, and 15 of a cylindrical spacecraft rotating clockwise normal to the sun at one revolution per hour. The heating rates to the four small surfaces shown in Figure 2-5 can be obtained as follows.

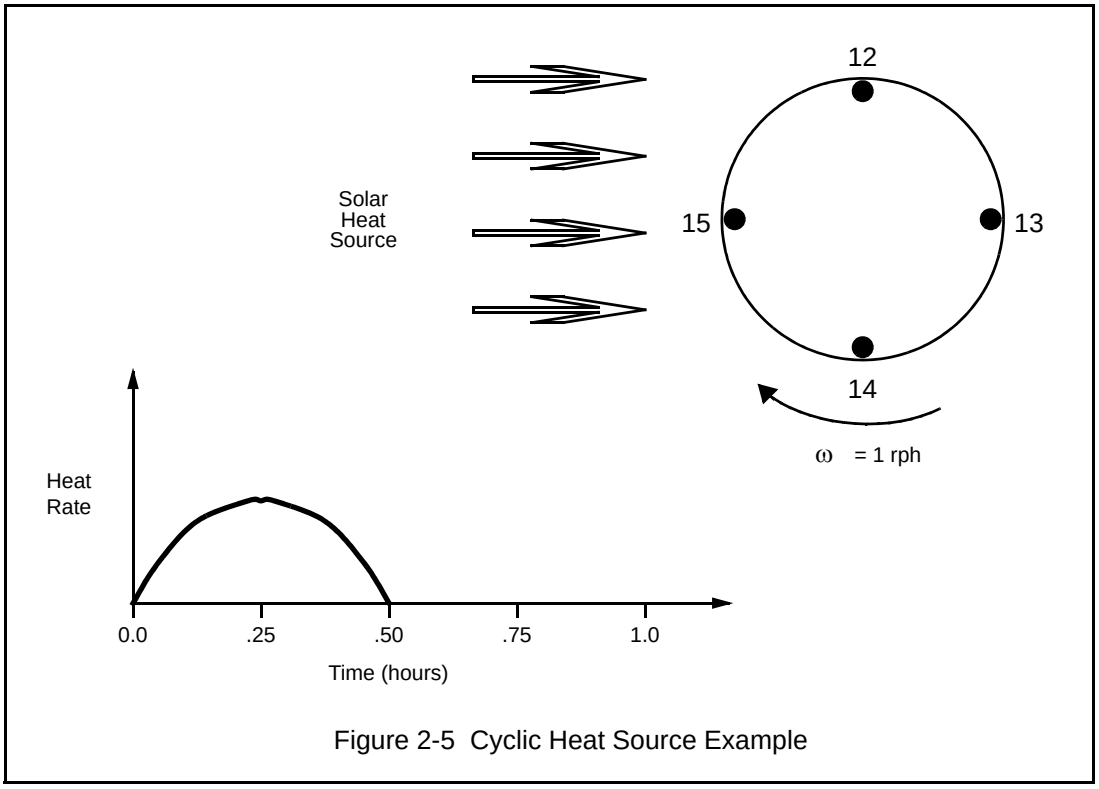


Figure 2-5 Cyclic Heat Source Example

The heating data is entered in an ARRAY DATA block in the usual tabular form. For this case, data is assumed to be entered into arrays 1 and 5. With the cylindrical spacecraft rotating clockwise, and with the above orientation of each surface with respect to the sun at time zero, the phase displacements are as follows:

NODE	PHASE DISPLACEMENT
14	0.0
15	0.25
12	0.50
13	0.75

2.13.2 Backward Compatibility: the Kn and XKn option

[This section is intended only for users of older versions.]

SOURCE DATA accepts numbered user constants (K and XK) as part of the definition of the source term in some options. **This usage is historic and has been replaced by the use of registers and expressions, as explained in Section 2.8 and Section 4.9. Usage of this feature is discouraged: older models should be updated in the event that this usage becomes unsupported.** To support this former usage, if no arithmetic expressions are used in combination with user constants (e.g., only the Kn or XKn appear as the input), then the old fashioned methods are used to assure backward compatibility. However, if any multiplier is used (e.g., "3.0*XK3) or any other operation, then the modern dynamic expression handling methods are used instead.

2.13.3 Source Data Referencing

All references to source data in the logic blocks (see Section 4.1) use the forms: Qn or smn.Qn where smn is a submodel name and n is the node number (see Section 2.12.2). As usual, the "smn." prefix is required only in the OPERATIONS, PROCEDURE, SOLOUTPUT, SOLOGIC, or SUBROUTINES blocks and for cross-references to a node outside the submodel stated in the header or DEFMOD record currently in effect. With this accessing capability, the user may update source data values at will with statements in any logic block.

***CAUTION**: Q values for nodes are zeroed and recalculated every solution step (i.e., every steady state iteration and every transient time step). This means the value of "smn.Qn" as used in any logic block or data expression can change drastically during the course of a solution. It is recommended that such references either be avoided or used with extreme caution.*

2.13.4 Source Data Updating

In addition to updates dictated by the use of registers and processor variables used within expressions, the time and temperature dependent SOURCE DATA inputs defined in the SOURCE DATA blocks are actually applied (that is, stored in the source locations) at the end of VARIABLES 0 and VARIABLES 1 for time and temperature dependent operations, respectively (Section 4.1.2).

The DTV option (time and temperature dependent) is done in VARIABLES 1 prior to each temperature iteration. Three-blanks and GEN options that are dependent on user constants are updated after VARIABLES 1.

These updates are done at the end of the VARIABLES 0 and VARIABLES 1 subroutines through calls to program subroutines QVTIME and QVTEMP. The preprocessor normally supplies these calls subsequent to any user-supplied logic in the VARIABLES 0 or VARIABLES 1 blocks. This gives the user an opportunity to update any registers or arrays that may affect the SOURCE DATA calculations before they are performed. If user intervention is required *after* the QVTIME/QVTEMP calls, the user may supply a "CALL QVTIME" statement anywhere in the VARIABLES 0 input and it will not be overridden by a preprocessor-supplied call. This is also true for QVTEMP in VARIABLES 1.

To conform with previous SINDA practice, ***all time and temperature dependent source terms are set to zero (or their "constant" SOURCE DATA value) at the beginning of VARIABLES 0.*** All constant value sources input in the SOURCE DATA block are reset to their initial values at the beginning of VARIABLES 0. Sources will only be zeroed at that point if no source term had been input or if the constant source value were zero. *In steady-state runs, where VARIABLES 0 is called only once, source data values are reset to their post-VARIABLES 0 (and QVTIME) values at the start of each iteration step.*

Heat transfer ties to fluid submodels (described later) affect the nodal Q-source value. The effect of these ties appears after the VARIABLES 0 (and QVTIME) updates, but before the VARIABLES 1 (and QVTEMP) updates *in all solution routines.* FLUINT ties are therefore treated like time-dependent sources in transient solutions. In steady-state solutions, on the other hand, tied lumps are treated like boundary nodes from the nodes' perspectives.

CAUTION FOR "3-BLANKS" SOURCE UPDATES: There is a significant difference in the timing of internal updates between setting a Q source to be a numeric constant (e.g., "3.0" or "pi*148") versus an expression containing a register or processor variable (e.g., "power" or "factor*(T14-Tref)"). In the former (constant) case, the Q-value will be set *before VARIABLES 0*, whereas in the latter (variable) case, the Q-value will be set *after VARIABLES 1* because of the potential for the underlying variables to be temperature-dependent (whether or not they actually do depend on temperature).

Refer also to Section 4.9 for information on automate updates using dynamic registers.

2.14 CONDUCTOR DATA

Conductors are SINDA network elements that describe how energy is to be transported between two nodes. Conductors are heat transfer paths. Two basic types of conductors may be defined by the user in CONDUCTOR DATA blocks: *linear* and *radiation*.

Linear Conductors *—The conductance of a linear conductor is input in units of energy per unit time per unit degree,* and the heat rate through such a conductor is calculated in the network solution routines as:*

$$HR = G \bullet (T_i - T_j)$$

where:

HR. . . . . Heat rate (energy/time)
G . . . . . Linear conductance
T . . . . . Temperature of end-point nodes i and j

Several types of physical heat transfer mechanisms can be modeled as linear conductors. For heat transfer by conduction, the conductance should be computed as:

$$G = \frac{k \bullet A}{L}$$

where:

k . . . . . Thermal conductivity (energy/length-time-deg)
A . . . . . Cross-sectional area of the conduction path (length^2)
L . . . . . Length of the conduction path (length)

For heat transfer by convection, the conductance should be computed as:

$$G = h \bullet A$$

where:

h . . . . . Convective film coefficient (energy/length^2-time-deg)
A . . . . . Surface area (length^2)

* Temperature can be in degrees F, C, R, or K as long as all data (including mass, length, and time) contributing to the G-value constitute a consistent set. Consistency must also exist among all the capacitance and temperature values in the set of active submodels. The fac and tcon arguments on HEADER records and the FAC record are provided for this. The temperatures must also be consistent with the value ABSZRO. When fluid models are active, the convention simplified by the UID control flag must also be followed.

For heat transfer by mass flow, the conductance should be computed as:

$$G = \dot{m} \bullet Cp$$

where:

$\dot{m}$ . . . . . Mass flow rate (mass/time)
Cp. . . . . Specific heat of the flowing material (energy/mass-deg)

Radiation Conductors — *The conductance of a radiation conductor is input in units* of energy per unit time per degree^4, and the heat rate through such a conductor is calculated in the network solution routines as:*

$$HR = G \bullet (T_i^4 - T_j^4)$$

where:

HR . . . . . Heat rate (energy/time)
G . . . . . Radiation conductance
T. . . . . *Absolute* temperature of endpoint nodes i and j

The value input as the conductance of a radiation conductor should be computed as:

$$G_{input} = \sigma \bullet F \bullet A$$

where:

σ. . . . . Stefan-Boltzmann constant (energy/length^2-time-deg^4, e.g., 0.1714E-8 BTU/ft^2-hr-R^4 or 5.667E-8 W/m^2-K^4). This parameter is available within expressions as sbcon or sbconsi. All radiation conductions in a model will be multiplied by the control constant SIGMA (see Section 4.4.1), which defaults to unity. SIGMA therefore represents a convenient location for containing the Stefan-Boltzmann constant, but users are cautioned to avoid mistakenly applying the Stefan-Boltzmann constant twice: once in CONDUCTOR DATA and once in CONTROL DATA, GLOBAL as SIGMA.
F. . . . . Gray-body factor
A . . . . . Surface area (length^2)

Radiation conductors (also known as radiation exchange factors, and “RADKs”) are only possible to calculate by hand in simple cases. Usually, they are supplied along with solar heating loads by programs such as Thermal Desktop/RadCAD® (see Preface, page xliii).

* In addition to the units consistency required for linear conductors, radiation conductors must also be consistent with the SIGMA value specified in CONTROL Data. The built-in values sbcon and sbconsi are convenient for defining SIGMA—see Section 2.8.4 and Section 4.4.1.

Preliminary Input Rules *— All conductor input options require that the pair of nodes to which a conductor is attached be specified by their ID numbers. The user should recall that the negative sign preceding boundary node numbers on node data lines is not part of the real node number. On conductor data cards, negative signs prefixing node numbers are not flagged as errors because they are used to signify one way conductors (see below).*

If *both* nodes connected by a conductor are in the submodel named in the preceding HEADER CONDUCTOR DATA record, just the node number (N#) will suffice. Since inter-submodel heat transfer is allowed, the form smn.N# is available when one of the pair is in another submodel. The submodel format for upstream nodes in one way conductors is smn.-N#. Note that the second submodel name is not echoed in the preprocessor printback for conductor generation options for reasons of internal execution economy.

A conductor may also appear with both node numbers identified with submodels outside the conductor's submodel. In other words, it is possible to build conductor-only submodels (although an empty NODE DATA block is required.) This allows the user to define one or more heat transfer paths between submodels to be changed or eliminated simply by using the BUILD macroinstruction. The user should keep in mind that the physical constituents of his or her model are primarily diffusion and arithmetic nodes, secondarily conductors and boundary nodes.

One-Way Conductors *— Historically, to facilitate the modeling of fluid loops without a fluid submodel, SINDA allows any conductor to be specified as a one-way conductor. One-way conductors permit the modeling of heat transport by fluid (mass) flow, in which case their conductances are always a function of* $\dot{m} \cdot Cp$. *Such a conductor is defined by prefixing the node number of one of the adjoining nodes with a minus sign. The node so designated*[*] *will not be allowed to lose or gain heat through the conductor, even though the temperature of the node will be used to calculate a heat flow. In other words, the solution subroutines will compute the heat transferred through a one-way conductor as though it were an ordinary conductor. This heat will not, however, be allowed to enter (or leave) the node prefixed by the minus sign; it will be allowed to leave (or enter) the unsigned node only.*

In other words, the "downstream" (positive) node solution includes the effects of the "upstream" (negative-flagged) node, *but the upstream node is completely unaware of the existence of the downstream node. This has significant modeling repercussions beyond simple fluid flow modeling, which is better accommodated using fluid submodels.* For example, one-way radiation conductors are perfectly valid.

One important use of one-way conductors is to reduce model sizes by exploiting symmetries. Dual, opposing one-way conductors (see Figure 3-29) can be used in a manner analogous to the FLUINT path and tie DUP options described later. For example, if a model contains N identical subnetworks all attached to the same main node, then one one-way conductor of value G could be placed from the main node to the "duplicated" subnetwork node, and a second of value N*G would be placed from the subnetwork node to the main node. The main node would then "see" N subnetwork nodes, whereas the subnetwork node would only "see" one main node.

* If fluid flows from node A to node B, then node A (the upstream node) should be flagged with a minus sign to signal a one-way conductor. If the upstream node is outside of the current submodel, the correct designation is: smn.-N#

N need not be an integer. For example, consider an insulated box with X square foot of surface area containing M widgets. A model of one square foot of insulation could be developed independent of a model of one widget, and the two could later be combined (perhaps as separate submodels) using dual one-way conductors. The widget model would “see” X/M square feet of insulation, whereas the insulation model would “see” M/X widgets. No further modifications would be necessary to produce a model of the combined system, assuming that the unit systems matched or that one could be converted using FAC records. Such interpretation and usage is also handy when reducing a detailed component model to a simpler, faster system-level model, as illustrated by Sample Problem G in the Sample Problem Appendix.

The use of one-way conductors has effects on the model energy balance, and on QMAP and NODMAP output files. In QMAP and NODMAP files, downstream nodes do not appear in the listings for the upstream nodes since they have no effect on those nodes. Because of this lack of one-to-one correspondence, energy can appear to be created or destroyed at these locations, which can disrupt the system-level energy balance calculation required for steady-state convergence (as governed by the EBALSA control constant). In this event, SINDA will converge, but will print a warning that the net energy flows are unbalanced. This should not be misinterpreted as signaling an incorrect answer if one-way conductors are used. However, large conductors can also cause this message if the default iterative methods are used, in which case the program is attempting to warn the user that little to no progress is being made toward a solution, and that either tighter convergence criteria or changes/corrections to the model are needed, including perhaps selection of matrix methods.

2.14.1 Conductor Data Input

All conductor data records appear following a header record:

```
HEADER CONDUCTOR DATA, smn [,fac]
```

where:

`smn` . . . . . a submodel name
`fac` . . . . . conductance multiplying factor
`[ ]` . . . . . denotes optional argument

The following general format rules apply for all conductor input options:

1. Linear conductors must have a positive conductor number.
2. Radiation conductors must have a negative conductor number (analogous to boundary node designations, the negative sign is used purely as a signal and has no other meaning).
3. Multiple conductors, or groups of same-valued conductors, may be defined on a single input record using the Standard (3 blanks) option code. All other option codes appear only once per record, together with their data values.
4. Conductor identification numbers may consist of up to 8 digits (e.g., 99999999).

5. Whenever expressions are used to define inputs, the sign of the *current* value (during preprocessor execution) of the expression is used as a flag for determining conductor type (or other suboptions).

6. References to processor variables within expressions may refer to “#this” as the conductor identifier, to “#nodeA” as the identifier of the first node, and to “#nodeB” as the identifier of the second node. When using the last two operators, submodel prefixes are not allowed.

Table 2-9 summarizes the available conductor data input options. Like all other model parameters, conductance values may be referenced and changed in the logic blocks. Hence, a “constant” value input in conductor data need not remain constant during the entire problem solution.

Table 2-9 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 variable (e.g., pressure) that is a function of time. 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

While `fac` may use registers in its defining expression, this value is treated as a permanent constant and is not updated by the dynamic register options described in Section 4.9, and it cannot reference processor variables in its definition. This does not imply that the conductances defined using these factors are constant, just the factors themselves.

Standard Option (3 blanks card code) *— The simplest conductor data card utilizes the 3 blank option and is formatted as follows:*

```
G#,NA,NB,G              $(basic format)
-4,7,9,3.5E-13          $(example 1)
18,31,POD2.8,1.8        $(example 2)
14,8,-9 , emdot*Cp      $(example 3)
```

where:

G# . . . . . Conductor ID number (integer)
NA,NB . . . . . ID numbers of the nodes to which this conductor is connected (integer data value or smn.integer)
G. . . . . Conductance (floating point data value or expression)

Defining the G value using formulas (register containing expressions, such as “COND*AREA/LENGTH”) makes the model readily adjustable.

Example 1 defines conductor number 4 as a radiation conductor connected between nodes 7 and 9 and having a conductance value of 3.5×10^{-13}. Example 2 defines conductor number 18 as a linear conductor connected between nodes 31 and node 8 of submodel POD2, having a conductance value of 1.8.

Example 3 defines a linear conductor, having a value of “emdot*Cp” (defined indirectly using registers) connected as a one-way conductor between nodes 9 and 8. Heat will flow to and from node 8, but will be prevented from entering or leaving node 9 (through this convection path).

Multiple node pairs: A note to users of old versions—The use of a single conductor between multiple node pairs is an anachronism that should be avoided. It is supported but no longer documented. Such usage is incompatible with Sinaps, is incompatible with “#nodeA” and “#nodeB” indirect references in expressions, can make the model more difficult to understand for maintenance purposes, and is no longer needed with the advent of dynamic registers.

GEN Option*—The GEN option is used to generate and input a group of conductors each having the same conductance value. This option requires eight data values for each group of similar conductors, as follows:*

```
GEN G#,#G,IG,NA,INA,NB,INB,G        $(basic format)
GEN -5,3,1,10,1,20,1,4.8E-12        $(example 7)
```

where:

G#.....	ID number of the first conductor to be generated (integer)
#G.....	Total number of conductors to be generated (integer). The value of #G can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: conductor generation is a one-time event performed by the preprocessor based on initial register values.
IG.....	Increment to be added to the current conductor number to form the conductor number of the next conductor (integer)
NA,NB.....	ID numbers of the pair of nodes connected by the first conductor (integer or smn.integer)
INA,INB...	Increments to be added to NA and NB, respectively, to arrive at the node numbers for the pair of nodes connected by the next conductor (integer)
G.....	Conductance of all conductors (floating point or expression)

A negative sign preceding the G# causes all conductors to be defined as radiation conductors. The negative sign, however, is never part of the conductor number in the arithmetic sense. Therefore, the conductor numbers generated in example 7 would be 5, 6 and 7 (not -5, -4 and -3), and all three would be radiation conductors. Similarly, a negative sign preceding NA or NB would define the group of conductors as being the one-way type. This negative sign, too, is never part of the node number in the arithmetic sense. Therefore, the NA, NB node pairs generated in example 8 would be (40, 50), (41, 49), and (42, 48). The increment values, IG, INA, and INB, may assume any nonzero integer value.* Example 7 is equivalent to the following cards prepared under the standard (3-blanks) option:

```
-5,10,20,4.8E-12
-6,11,21,4.8E-12
-7,12,22,4.8E-12
```

A “fan” of 5 like-valued conductors may be generated as follows:

```
GEN 51,5,1,22,0,31,10,con*area/length$ example 9
```

The zero entry for INA generates node pairs as follows: (22,31), (22,41), (22,51), (22,61), (22,71). Note the arithmetic in the G value field.†

Automated Variable Conductance Options *— Temperature-dependent conductances can be encoded into the expression defining the conductance itself. For example, if the conductivity of conductor 22 obeyed a linear relationship with temperature, Con(T) = Cdt*(T-Tref) + Conref, and an average of the endpoint node temperatures were used:*

```
22,21,23 \
   area*(Cdt*(0.5*(T#nodeA+T#nodeB)-Tref)+Conref))/length
```

* Use of zero IG is available, but not documented and not recommended.

† The eleven-data-value GEN option has been made obsolete. It is available but not documented.

However, more convenient means are available for describing the temperature-dependence of a single material and reapplying that relationship to multiple conductors. The rest of this section describes these features.

As will be explained in Section 4.1.2, the network solution routines call for the execution of the operations in the VARIABLES 1 block prior to performing each iteration on the heat transfer equations. Using this feature, the user could get a temperature-dependent conductance by specifying VARIABLES 1 operations which would insert a new conductance value in the conductance table. This new conductance could be calculated using any convenient algorithm. This is probably the most efficient way from a computational standpoint, but it tends to be rather cumbersome, especially for those users with limited programming experience.

The automated options cause the current value of variable conductors to be calculated within the network solution routines at the end of the VARIABLES 1 operations, and therefore relieve the user of the task of preparing and inputting conductance computation algorithms. As specified by the user, one of two standard methods is used to calculate the current value of a variable conductor: (1) interpolation, and (2) polynomial evaluation. For interpolation, the user must input, in the ARRAY DATA block, a bivariate array of (temperature, conductance) ordered pairs representing points on the curve of T vs. G. The conductance is then calculated by performing linear interpolation on this array using temperature as the independent variable. For polynomial evaluation, the user must input an array of coefficients (P_0, P_1, P_2,... P_n) in the ARRAY DATA block. The conductance would then be calculated by evaluating the n^{th} order polynomial in T (i.e., P_0+P_1*T+P_2*T**2... +P_n*T**n). To provide further versatility, the value of the temperature, T, used as the independent variable in both of the above methods, may be taken as the average of the temperatures of the two nodes to which the conductor is connected, or it may be set equal to the temperature of only one of the nodes. This feature is explained along with the options to which it applies.

To enable the same array of points or coefficients to be used for all conductors passing through the same type of material, all options cause the computed conductance to be multiplied by a factor before being inserted in memory. In this way, for example, the array can be used to compute conductivity, k, and the multiplying factor can be used for A/L in the case of linear conductors or, emissivity, e, and A, respectively, in the case of radiation conductors. In another application, the multiplying factor could be used to scale the conductance to some set of consistent units.

All of the automated options require a multiplying factor which may be input as a floating point data value or an expression. Note that the user may change registers or processor variables used in the definition of this factor, and thus indirectly the conductor value, at any point in the solution process.

The interpolation options (except BIV*) require a doublet array of temperature vs. conductance. (Conductance is used loosely in this context. The array could contain, for example, values of temperature vs. conductivity, as long as the associated multiplying factor had units of area/length.) The array must contain an even number of floating point data values, as follows:

$$T_1, G_1, T_2, G_2, \ \ldots, \ T_j, G_j, \ \ldots, T_n, G_n$$

* The BIV option requires a bivariate array of time and temperature vs. conductance

The temperature values, T_j, must be strictly increasing with j. If the value of the effective temperature (average or single node) at some time is less than T_1, then G_1 will be used in lieu of performing an extrapolation; if the temperature is greater than T_n, then G_n will be used.

The polynomial options require that a singlet array of coefficients be input in the ARRAY DATA block. The array must consist of n+1 floating point values, where n is the order of the polynomial. The coefficients should be entered in the array in the order of increasing powers of temperature (e.g., the first data value is the constant term, P_0, and the $(n+1)^{th}$ data value is the coefficient, P_n, of T**n).

For both classes of options, the arrays are referenced by using the form: An or smn.An, where n is the array number.

SIV and SPV Options - *These options allow the user to define single conductors having temperature varying conductance. The SIV option assumes the referenced array contains points of T vs. G and is to be used for interpolation, whereas the SPV option assumes that the array contains polynomial coefficients. Both options require five data values for each conductor as follows:*

```
SIV G#,NA,NB,AP,F              $(basic format)
SPV G#,NA,NB,AC,F              $(basic format)
SIV -12,24,-25,A3,-0.0001      $(example 10)
SPV 4,5,6,A2,factor            $(example 11)
SIV 4,5,6,A3,0.0001            $(example 12)
```

where:

`G#`..... Conductor ID number (integer)

`NA,NB`..... ID numbers of the pair of nodes to which the conductor is connected (integer or smn.integer)

`AP,AC`..... A reference to an array of points, or coefficients, respectively, of the form: An or smn.An, where: n is the array number

`F`..... Multiplying factor, input as a positive floating point data value or expression. If the factor is negative and a minus sign precedes the expression, the minus sign is taken as a flag to use the temperature of the first node (and not an average temperature), and the minus sign will then be ignored. *To avoid this treatment, use SIVM, SIVA, SPVM, or SPVA conductors described below.*

As for all other options, this type of conductor may be specified as a one-way conductor by prefixing NA or NB with a minus sign.

Normally, the average temperature of the two adjoining nodes (i.e., $(T_{NA} + T_{NB})/2$) will be used as the independent value for interpolation or polynomial evaluation. If the user wishes to use the temperature of one of the nodes only then this node is identified by placing its number as the NA data value (not the NB value), and the single temperature option is indicated by prefixing the expression with a minus sign. Since the negative serves only as a flag, the multiplying factor will always be interpreted as a positive quantity. *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. *To avoid this treatment, use SIVM, SIVA, SPVM, or SPVA conductors described below.*

Example 10 illustrates the correct input for a radiation conductor connected between nodes 24 and 25 but conducting no heat to or from node 25, wherein the temperature of node 24 will be used as the independent variable for interpolating on the T vs. G points supplied in array number 3, with the result of the interpolation being multiplied by 0.0001 (NOT -0.0001!) prior to insertion into memory. The above example, while unrealistic, encompasses all three methods for using minus signs as flags to establish conductor definition and evaluation options.

Examples 11 and 12 are discussed in detail in the following paragraphs.

Consider the thermal network shown in Figure 2-6. The conductance of the conductor is defined by the following equation:

$$G = (0.0001) \times (0.005 \times T^2 + 1.0)$$

where T is the average temperature of adjoining nodes.

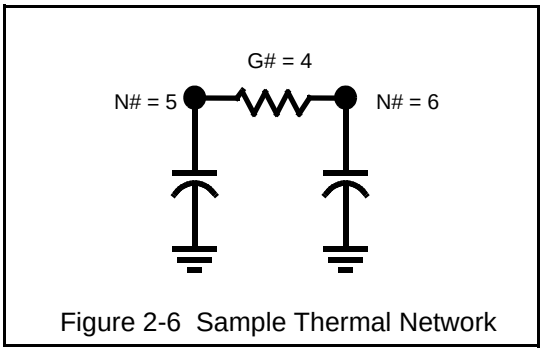


Figure 2-6 Sample Thermal Network

Example 11 illustrates the correct method of defining this conductor using the SPV option. The required coefficients of the polynomial would be entered in the ARRAY DATA block with the following record:

```
2 = 1.0,0.0,0.005
```

The required multiplying factor would be entered in the REGISTER DATA block with the following record:

```
factor = 0.0001
```

An alternate method of defining this conductor is illustrated in Example 12. This example uses the SIV option and thus requires an array of T vs. G points for interpolation. The array is developed by approximating the T vs. G curve with a series of straight lines as shown in Figure 2-7. The points which define this approximation are entered in the ARRAY DATA block as follows:

```
3 = 0.0,1.0,10.0,1.5,20.0,3.0,40.0,9.0
```

The multiplying factor is supplied directly on the conductor data card as the floating point data value, 0.0001.

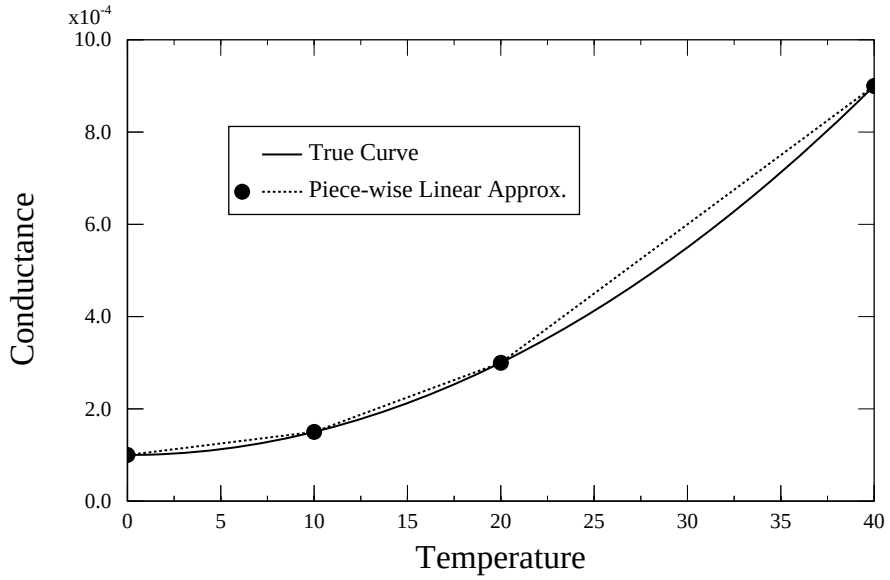


Figure 2-7 Curve of Temperature-Varying Conductance

A common usage of these options is to input an array of k vs. T (i.e., conductivity versus temperature) *once* for each material used in the model, and then to specify appropriate A/L (heat transfer area over length) of each linear conductance as the multiplying factor in each conductor declaration.

Defining the F value using formulas (register containing expressions, such as "AREA/ LENGTH") makes the model readily adjustable.

SIVM, SIVA, SPVM and SPVA Options - *These options are identical to the SIV and SPV options, except that the sign of the F factor is not used as a signal. Instead, the use of the mean or node A temperature is signaled by the suffix of an "M" (mean) or "A" (node A) onto "SIV" or "SPV." Both options require five data values for each conductor as follows:*

```
SIVM G#,NA,NB,AP,F          $ format: use mean T
SPVM G#,NA,NB,AC,F          $ format: use mean T
SIVA G#,NA,NB,AP,F          $ format: use node A's T
SPVA  G#,NA,NB,AP,F         $ format: use node A's T
```

where:

`G#`..... Conductor ID number (integer)

`NA,NB`..... ID numbers of the pair of nodes to which the conductor is connected (integer or smn.integer)

`AP,AC`..... A reference to an array of points, or coefficients, respectively, of the form: An or smn.An, where: n is the array number

`F` ..... Multiplying factor, input as a floating point data value or expression. *This factor or expression is taken verbatim regardless of its sign.*

SIM and SPM Options — *These two options combine the features of the SIV and SPV options with those of the GEN option. That is, they allow the user to generate and input a group of conductors all having the same temperature varying conductance. The SIM option generates conductors whose conductance will be evaluated by interpolation, and the SPM option generates conductors whose capacitance will be evaluated from a polynomial. Nine data values are required to define each group of conductors as follows:*

```
SIM G#,#G,IG,NA,INA,NB,INB,AP,F      $(basic format)
SPM G#,#G,IG,NA,INA,NB,INB,AC,F      $(basic format)
SIM 10,3,1,5,1,6,1,A3,factor         $(example 13)
```

where:

G# . . . . . ID number of first conductor to be generated (integer)

#G . . . . . Total number of conductors to be generated (integer). The value of #G can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: conductor generation is a one-time event performed by the preprocessor based on initial register values.

IG . . . . . Increment to be added to the current conductor number to form the conductor number of the next conductor to be generated (non-zero integer)

NA,NB . . . . . ID numbers of the pair of nodes to which the first conductor is connected (integer or smn.integer)

INA,INB. . . . Increments to be added to NA and NB, respectively, to form the node numbers for the pair of nodes connected by the next conductor (integer)

AP,AC . . . . . A reference to an array of T vs. G points, or polynomial coefficients, respectively, of the form: An or smn.An where n is the array number

F. . . . . Multiplying factor input as a floating point data value or expression. If negative and a minus sign precedes the expression, the minus sign is taken as a flag to use the temperature of the first node (and not an average temperature), and the minus sign will then be ignored. *To avoid this treatment, use SIMM, SIMA, SPMM, or SPMA conductors described below.*

The conductors defined by Example 13 are equivalent to those defined by the following:

```
SIV 10,5,6,A3,factor
SIV 11,6,7,A3,factor
SIV 12,7,8,A3,factor
```

As for all other options, a negative sign prefixing the G# establishes the entire group as radiation conductors; a negative sign prefixing NA or NB defines a group of one-way conductors: a negative sign preceding F establishes the temperature of node NA as the value of the independent variable to be used for interpolation or polynomial evaluation. Negative signs preceding G#, NA, or NB serve as flags to establish the type of conductors, in accordance with the usual conventions. Negative signs preceding IG, INA, or INB, however, indicate that the data value is, indeed, a negative arithmetic quantity.

SIMM, SIMA, SPMM and SPMA Options *- These options are identical to the SIM and SPM options, except that the sign of the F factor is not used as a signal. Instead, the use of the mean or node A temperature is signaled by the suffix of an "M" (mean) or "A" (node A) onto "SIM" or "SPM." Both options require nine data values for each conductor as follows:*

```
SIMM G#,#G,IG,NA,INA,NB,INB,AP,F      $ use mean T
SPMM G#,#G,IG,NA,INA,NB,INB,AC,F      $ use mean T
SIMA G#,#G,IG,NA,INA,NB,INB,AP,F      $ use node A's T
SPMA  G#,#G,IG,NA,INA,NB,INB,AC,F     $ use node A's T
```

where:

G#..... ID number of first conductor to be generated (integer)

#G..... Total number of conductors to be generated (integer). The value of #G can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: conductor generation is a one-time event performed by the preprocessor based on initial register values.

IG..... Increment to be added to the current conductor number to form the conductor number of the next conductor to be generated (non-zero integer)

NA,NB..... ID numbers of the pair of nodes to which the first conductor is connected (integer or smn.integer)

INA,INB ... Increments to be added to NA and NB, respectively, to form the node numbers for the pair of nodes connected by the next conductor (integer)

AP,AC..... A reference to an array of T vs. G points, or polynomial coefficients, respectively, of the form: An or smn.An where n is the array number

F ..... .Multiplying factor, input as a floating point data value or expression. *This factor or expression is taken verbatim regardless of its sign.*

Table 2-10 summarizes the SIV and SPV and related temperature-varying conductor options.

Table 2-10 SIV and SPV Family of Temperature-varying Conductors

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

The new user is encouraged to use the four-lettered options (SIVM, SPVM, etc.) rather than the three-lettered options (SIV, SPV, etc.), and the existing user is encouraged to convert to the four-lettered options to avoid reliance on the negative sign flagging of the F factor. Eventually, the three-lettered options will become undocumented.

DIV and DPV Options *— These options allow the user to define a conductor which passes through two materials, each having a different temperature varying conductance. The DIV option uses the interpolation method to evaluate the conductance, whereas the DPM option uses a polynomial. Both options permit one material to have a constant conductance. Both options require seven data values for each conductor as follows:*

```
DIV G#,NA,NB,APA,FA,APB,FB     $(basic format)
DPV G#,NA,NB,ACA,FA,ACB,FB     $(basic format)
DPV 10,20,30,A8,K9,A14,0.03    $(example 14)
```

where:

G# . . . . . Conductor ID number (integer)
NA,NB . . . . . ID number of the pair of nodes to which the conductor is connected (integer or smn.integer)
APA,APB. . . . References to arrays of T vs. G points of the form: An or smn.An, where n is the array number
ACA,ACB. . . . References to arrays of polynomial coefficients, of the form: An or smn.An
FA,FB . . . . . Multiplying factors input as floating point data values or expressions

Using interpolation (or polynomial evaluation) as the calculation scheme, the first conductance, GA, is computed by using the temperature of node NA as the independent variable for operating on array APA (or ACA) with the result multiplied by FA. In similar fashion, the temperature of node NB, array APB (or ACB) and factor FB are used to compute the second conductance, GB. These two conductances are then combined to form the total conductance, GT, which is entered in memory. The combination algorithm depends on the type* of the conductor, as follows:

Linear (series conduction): $GT = \left(GA^{-1} + GB^{-1} \right)^{-1}$

Radiation: $GT = GA \cdot GB$

If one of the two materials does not have a temperature varying conductance, this material may be identified by replacing its corresponding array reference with a floating point data value or expression. The constant conductance of this material will then be the product of this data value or

* This option should be used with caution when applied to radiation conductors to avoid the calculation of an effective radiation conductance of GT = $\sigma^2 F_1 F_2 A_1 A_2$

constant and its associated multiplying factor. This conductance is combined with the variable conductance to form GT, as described above. For completeness, all possible alternate formats are listed, as follows:

```
DIV G#,NA,NB,SUB,FA,APB,FB     $(GA = SUB*FA)
DPV G#,NA,NB,SUB,FA,ACB,FB     $(GA = SUB*FA)
DIV G#,NA,NB,APA,FA,SUB,FB     $(GB = SUB*FB)
DPV G#,NA,NB,ACA,FA,SUB,FB     $(GB = SUB*FB)
DPV 80,90,91,4.0,3.0,A6,factr $(example 15)
```

where SUB is a floating point data value or expression.

In example 14, the temperature of node 20 will be used with array 8 and constant 9 to compute GA, and the temperature of node 30 will be used with array 14 and the value 0.03 to compute GB. The total conductance, GT, will be entered for conductor number 10. In example 15, the value of GA will be computed as 12.0 (= 4.0*3.0), and the value of GB will be determined from T(91), A6, and “factr.”

DIM and DPM Options — *These two options combine the features of the DIV and DPV options with those of the GEN option. They are used to generate and input a group of conductors all made of the same two materials having different temperature varying conductances. The DIM option causes the conductances for the two materials to be evaluated by interpolating on the referenced arrays, and the DPM option computes the conductances from polynomial coefficients supplied in the arrays. As for the other variable-conductance options, multiplying factors are applied to the results of the operations on the arrays. Eleven data values are required to define each group of conductors, as follows:*

```
C (basic format)
       DIM G#,#G,IG,NA,INA,NB,INB,APA,FA,APB,FB
C (basic format)
       DPM G#,#G,IG,NA,INA,NB,INB,ACA,FA,ACB,FB
C (example 16)
       DIM 10,3,2,20,1,30,-1,A12,fact12,A13,fact13
```

where:

- G#..... ID number of the first conductor to be generated (integer)
- #G..... Total number of conductors to be generated (integer)
- IG..... Increment to be added to the current conductor number to form the conductor number of the next conductor (non-zero integer)
- NA,NB..... ID numbers of the pair of nodes to which the first conductor is connected (integer or smn.integer)
- INA,INB... Increments to be added to NA and NB, respectively, to form the NA, NB pair associated with the next conductor (integer)

APA, APB. . . . References to arrays of T vs. G points, of the form: An or smn.An, where n is the number of the array being referenced.
ACA, ACB. . . . References to arrays of polynomial coefficients, of the form: An or smn.An
FA, FB . . . . . Multiplying factors input as floating point data values or expressions

Negative signs preceding G#, NA, or NB serve as flags to establish the type of the conductors, in accordance with the usual conventions. Negative signs preceding IG, INA, or INB, however, indicate that the data value is, indeed, a negative arithmetic quantity.

Since the DIM and DPM options are equivalent, in effect, to a series of conductors defined using the DIV or DPV options, the rules for evaluating the conductance of each conductor generated under the former pair of options are the same as those described in the section covering the latter pair of options.

For either option, one of the two materials may be defined as having a constant conductance by replacing the array reference for the material with a floating point data value or expression. This feature makes the following alternate formats acceptable:

```
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,SUB,FA,ACB,FB
DPM G#,#G,IG,NA,INA,NB,INB,ACA,FA,SUB,FB
```

where SUB is a floating point data value or expression.

The conductors defined in example 16 are equivalent to those which are defined by the following:

```
DIV 10,20,30,A12,fact12,A13,fact13
DIV 12,21,29,A12,fact12,A13,fact13
DIV 14,22,28,A12,fact12,A13,fact13
```

BIV Option *— The BIV option enables the user to define a conductor having a conductance which is a function of both time and temperature. The card format for such a conductor is similar to that for the SIV option and requires five data values for each conductor, as follows:*

```
BIV G#,NA,NB,APB,F     $(basic format)
BIV 19,8,11,A43, factr $(example 17)
```

where:

G# . . . . . Conductor ID number (integer)
NA, NB . . . . . ID numbers of the pair of nodes to which this conductor is connected (integer)
APB . . . . . Reference to a bivariate array (of temperature and time vs. conductance points) of the form: An or smn.An, where n is the array number
F. . . . . Multiplying factor input as a floating point data value or expression

The temperature value used is the average value of the temperatures of the two nodes (NA, NB) to which the conductor is connected. *A negative value of F is not permitted in this option.* The time value used is the mean time for the computational interval (control constant TIMEM).

Section 2.11.2 describes the structure of a bivariate array. In this application, the X independent variable should be temperature, the Y independent variable should be time, and the Z dependent variable should be conductance.

Example 17 defines conductor 19, connected between nodes 8 and 11. Array 43 has a bivariate structure, and constant 28 contains a floating point value.

PIV and PIM Options **-** *The PIV option enables the user to define a conductor having a conductance which is a function of both an arbitrary time-dependent variable "P" and temperature. The PIM option combines the features of the PIV option with those of the GEN option.*

The arbitrary variable P is first interpolated from the AT array as a function of time. This interpolated variable and temperature are used as the independent variables entering into the bivariate array AAT to interpolate the conductance.

A common usage of these functions is interpolating the conductance of an insulation material where the conductance is a function of temperature and pressure. The pressure versus time history is entered into array AT and the conductance as a function of temperature and pressure is entered into array AAT.

To specify the PIV or PIM options the following data are entered into the conductor data block:

```
PIV G#,NA,NB,AT,FT,AAT,FAT
PIV G#,NA,NB,SUB,FT,AAT,FAT
PIV G#,NA,NB,AT,FT,SUB,FAT
PIM G#,#G,IG,NA,INA,NB,INB,AT,FT,AAT,FAT
PIM G#,#G,IG,NA,INA,NB,INB,SUB,FT,AAT,FAT
PIM G#,#G,IG,NA,INA,NB,INB,AT,FT,SUB,FAT
PIM 1,3,1,1,1,2,1,A6,factr,A20        $(example 18)
PIV 4,2,5,A6,0.870,A20,ortho          $(example 19)
```

where:

G#	Conductor ID number (non-zero integer)
NA,NB	ID numbers of the nodes to which the conductor is connected (integer or smn.integer - see Note 3)
#G	Total number of conductors to be generated (non-zero integer)
IG	Increment to the conductor number (integer)
INA,INB	Increments to the node numbers, NA and NB, respectively (integer)
AT	Reference to doublet array of an arbitrary variable "P" vs. time (t)
AAT	Reference to bivariate array 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 4].

FT, FAT . . . . . Multiplication factors (floating point data values or expressions)
SUB . . . . . Floating point data value (literal) or expression used as a substitute for an array reference. A user constant *cannot* be used.

Note 1: In both the PIM and PIV options when "SUB,FT,AAT,FAT" is used, the arbitrary variable P is evaluated as P = SUB*FT. When the format "AT,FT,SUB,FAT" is used, the arbitrary variable P is evaluated by interpolation of array AT, and the conductance (G) is evaluated as G = P*FT*SUB*FAT.

Note 2: Conductor type is specified by the sign of the G# as follows:
LINEAR = positive conductor number
RADIATION = negative conductor number

Note 3: The negative signs on node numbers 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. The minus sign appears in the smn format as smn.-integer.

Note 4: Consistent with the bivariate array input format described in Section 2.11.2 the arbitrary variable P 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.

The following examples demonstrate the use of the PIV and PIM options. With reference to the Figure 2-8, it is desired to input the conductors as a function of pressure and temperature.

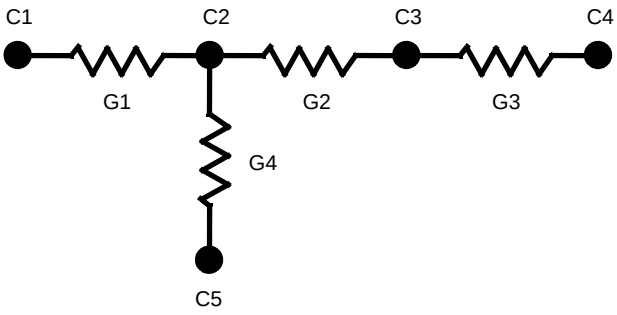


Figure 2-8 Example PIV and PIM Network

The following conditions are assumed:

1) A doublet array of pressure (Newtons/cm^2) versus time (sec) has been entered into the ARRAY DATA block as array no. 6.

2) A bivariate array of conductance (BTU/hr-°F) versus pressure (psi) and temperature (°F) has been entered into the ARRAY DATA block as array no. 20.

3) The constant 1.45 which changes Newtons/cm^2 to PSI has been entered into a REGISTER DATA block as “factr.”

Consistent with the above conditions, conductors 1, 2 and 3 can be specified using the PIM option, as shown in Example 18.

The following conditions are assumed for conductor no. 4:

1) Orthotropic effects reduce the conductance to 60 percent of the table value. Therefore, 0.6 has been entered into the REGISTER DATA block as “ortho.”

2) The pressure is determined to be 60 percent of the table value. Therefore, the pressure multiplication factors will be 0.6 * 1.45 = 0.870 (the factor 1.45 changes Newtons/cm^2 to psi).

Consistent with the above assumptions, conductor no. 4 can be specified using the PIV option, as shown in Example 19.

TVS and PER Options — *These options allow the user to define time-variable conductors. The time dependency is determined by interpolating on two singlet time and conductance arrays which should appear in an ARRAY DATA block. There is a restriction that both arrays must appear in the same block.*

If a problem involves time-dependent radiation conductors, (e.g., an articulating spacecraft in orbit) a huge volume of array data may be involved. This is the reason for restricting this option to singlet arrays, which require a minimum number of separate data values.

Probably the most efficient mode, up to a point, is to forgo the TVS and PER options, place the appropriate subroutine calls in VARIABLES 0 and use the ARRAY DATA block. This is practical if the subroutine calls can be provided by another computer program, but when one or more thousands of subroutine calls appear in VARIABLES 0, Fortran compilation time will definitely be nontrivial. The point where this approach compares poorly with TVS & PER must be determined by the user.

The ARRAY DATA format required for TVS & PER are:

$$nt = t_1, t_2, t_3 \text{ - - - } t_n$$
$$ng = g_1, g_2, g_3 \text{ - - - } g_n$$

where:

nt ..... time array number
t_i ..... time values, strictly increasing with i
ng ..... conductance array number
g_i ..... conductance values (dependent variable) (t_i, g_i are all floating point data values or expressions)

The CONDUCTOR DATA formats are as follows:

```
        TVS G#,NA,NB,At,Ag,F
C (example 20):
        TVS 16,120,DOOR.18,A1,A1412,fred*subm.T22
        PER G#,NA,NB,At,Ag,F,D,P
C (example 21):
        PER 23,131,62,POD.A1,POD.A113,1.0,0.0,1.522
```

where:

G# . . . . . Conductor ID number, integer

NA,NB . . . . . ID numbers of the nodes to which the conductor is connected (integer or smn.integer)

At . . . . . Reference to singlet time array of the form An or smn.An where n is the array ID number

Ag . . . . . Reference to singlet conductance array of the form An or smn.An where n is the array number

F. . . . . Conductance multiplying factor, floating point number or expression

D. . . . . Phase displacement as a decimal fraction of the total period, floating point value between 0.0 and 1.0 or an expression.

P. . . . . Period-specified in time units (floating point data value or reference to a user constant). P is usually the last time point in the At array, but not restricted to this value. If P is less than the last At 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 G value used for the time period between the last array time value and P is constant and equal to the G value for the last array time value

Example 20 defines a time-variable conductor no. 16 between node 120 and node 18 of submodel DOOR using time array 1 and conductance array 1412. The conductance value is multiplied by register “fred.”

Example 21 defines a time-variable conductor no. 23 between nodes 131 and 62. The time and conductance arrays are 1 and 113 of submodel POD, respectively. The period is 1.522 hours with no displacement offset.

2.14.2 Backward Compatibility: the Kn and XKn option

[This section is intended only for users of older versions of the code.]

CONDUCTOR DATA accepts numbered user constants (K and XK) as part of the definition of the conductance term in some options. **This usage is historic and has been replaced by the use of registers and expressions, as explained in Section 2.8 and Section 4.9. Usage of this feature is discouraged: older models should be updated in the event that this usage becomes unsupported.** To support this former usage, if no arithmetic expressions are used in combination with user

constants (e.g., only the Kn or XKn appear as the input), then the old fashioned methods are used to assure backward compatibility. However, if any multiplier is used (e.g., "3.0*XK3) or any other operation (excluding the minus sign used as a signal), then the modern dynamic expression handling methods are used instead.

2.14.3 Referencing Conductor Data in Logic Blocks

Each conductor is assigned an arbitrary submodel name and identification number by the user in the CONDUCTOR DATA blocks. As each conductor is accepted by the preprocessor, it is renumbered in sequence. The user assigned identification number is called the *actual*, or conductor identification number, and the preprocessor assigned sequence number is called the *relative* or internal number. There is no restriction as to what order conductors appear in the input file. After sorting by the preprocessor, they will be divided into linear and radiation conductors within each submodel. This is done for efficiency in solution routine execution and memory utilization.

As with all other variables, the Fortran subroutines resulting from the translation of the SINDA logic blocks (see Section 4.1) use the relative numbering system to access the conductor tables, where as the user refers to the conductors by their assigned ID numbers. Hence, the preprocessor maintains a table of actual vs. relative numbers, and performs the conversion from the former to the latter whenever a reference to a conductor is encountered in the logic blocks. The conductor tables are referenced by the following forms:

```
smn.Gn = conductance
```

where n is the actual (user specified identification) conductor number

As with other parameters, the smn.G(n) format is the "long" or complete reference form. This may be shortened to simply Gn when the submodel name is being supplied by an argument in the preceding header record. If the reference is to a G outside the submodel specified in the header record, then the long form is required. The OPERATIONS block is used for initialization and its logic relates to the entire model (as opposed to a single submodel) thus, the long form is the only one allowed in blocks such as OPERATIONS, unless DEFMOD is used.

The current absolute value of the heat rate through a conductor can similarly be referenced in logic or expressions as "smn.HRn." Note that HR is output parameter only: its value cannot be input directly. Note also that the use of absolute value for HR means that there is no sign convention for designating "from nodes" and "to nodes:" without looking at the temperature difference between the endpoint nodes, the user cannot immediately tell which node is gaining HR power and which is losing it.*

To indirectly reference the nodes on either end of a conductor or to calculate the heat rate through a conductor with a definite sign convention, use the CONDAT as described in Section 7.6.16. Otherwise, heat rate information is available via graphical user interfaces such as Sinaps® (see Preface, page xlii).

* If NODEPAIR has been specified in OPTIONS DATA in order to allow old models to run unchanged, then the HR parameter becomes unavailable and references to it will generate errors.

HR values are updated after temperatures have been updated in the steady state or transient solution routines. Therefore, the best location to employ these values in logic is in VARIABLES 2 or OUTPUT CALLS. A call to the output routine HRPRINT will also force an update of these values, so that fact can be exploited to force the HR values to be updated as frequently as needed.

2.14.4 Conductor Data Updating

In addition to updates dictated by the use of registers and processor variables used within expressions, the time and temperature dependent conductor data inputs defined in the CONDUCTOR DATA blocks are actually applied (that is, calculated and stored in the conductor locations) at the end of VARIABLES 0 and VARIABLES 1 for time and temperature-dependent operations, respectively (Section 4.1.2). The BIV option (time and temperature dependent) is done in VARIABLES 1 prior to each temperature iteration.

These updates are done at the end of the VARIABLES 0 and VARIABLES 1 subroutines using calls to program subroutines GVTIME and GVTEMP, respectively. The preprocessor normally inserts these calls subsequent to any user-supplied logic in the VARIABLES 0 or VARIABLES 1 blocks. This gives the user an opportunity to update any registers or arrays that may affect the conductor data calculations before they are performed. If user intervention is required after the GVTIME/GVTEMP calls, the user may supply a CALL GVTIME('smn') statement anywhere in the VARIABLES 0 input for submodel "smn" and it will not be overridden by a preprocessor-inserted call. This is also true for GVTEMP in VARIABLES 1.

All conductor value referencing in the logic blocks is done using the forms: Gn or smn.Gn where smn is a submodel name and n is the conductor ID number. As usual, the "smn" prefix is required only in the OPERATIONS block and similar global blocks, and for cross-references to a node outside the submodel stated in the header card currently in effect. With this referencing capability, the user may use or print G or HR values (and may also modify G values) in the logic blocks.

Refer also to Section 4.9 for information on how to automate updates using dynamic registers.

3 FLUID SYSTEM MODELING

This section is intended both as an introduction to FLUINT (*flu*id *int*egrator) and as a source of detailed explanations of fluid modeling methods. Note that FLUINT and SINDA are actually two halves of one program (which consists of one preprocessor and one processor library). Any reference in this section to *FLUINT* means "the fluid network capabilities in SINDA/FLUINT," while *SINDA* means "the thermal network capabilities."

3.1 Introduction and Overview

3.1.1 FLUINT Scope and Goals

FLUINT provides a general analysis framework for internal one-dimensional fluid systems. It can be used alone or in combination with the thermal analysis capabilities of SINDA. FLUINT is to thermohydraulic analyses what SINDA is to conduction/radiation analyses. The specific capabilities are described in the following paragraphs.

Arbitrary Fluid Systems. FLUINT is not restricted to any specific geometries or configurations. The program is based on network methods, allowing the user to describe almost any schematic representation.

Fluid Independence. FLUINT may be used to analyze any working fluid or mixture of fluids with adequately defined properties. Twenty refrigerants are immediately available, and the user may describe the properties of additional gases, liquids, and two-phase fluids with reusable FPROP DATA blocks (for cryogenic fluids, fire retardants, fuels and propellants, and other specialized heat transfer fluids).

Variable Resolution and Assumption Levels. FLUINT may be used for a wide variety of analyses, ranging from first-order sizing estimates to detailed transient response simulations. Spatial resolution may be varied by adding or subtracting network elements, and temporal resolution may be varied by choosing between time-dependent and time-independent types of elements. Furthermore, the performance of fluid components such as pumps can be as idealized or as realistic as appropriate to the problem and to the level of known detail.

Steady-State and Transient. FLUINT can be used for either steady-state analyses or for transient analyses. As noted above, the fidelity of the transient analysis can be varied from a time-independent thermal response (i.e., quasi steady-state hydrodynamics) to a fully time-dependent thermal/hydraulic response, depending on the *type* of network element selected.

Single and Two-Phase Flow. The code handles both single-phase and two-phase flow, along with transitions between states. This generality exists not only in the numeric solution, but also in the correlations used for each device or phenomena. In fact, the capability exists to handle capillary

devices, where surface tension forces dominate when two phases are present. Within the realm of two-phase flow, the phases are by default in equilibrium with each other and travel at the same velocity (i.e., homogeneous), but either assumption may be avoided if necessary.

Single- or Multiple-constituent Flow. Working fluids may consist of single substances or "constituents" in one or two phases, or may consist of mixtures of noncondensible gases and nonvolatile liquids with perhaps one constituent being condensible/volatile. In addition, noncondensible gases may be allowed to dissolve into and evolve from liquid phases, and chemical reactions can be modeled.

Liquid Compressibility and Water Hammer. Liquids are normally assumed incompressible, meaning that their densities are functions of temperature only. However, the compressibility of liquid *can* easily be included in the system response by making the control volume walls as compliant as the volumetric modulus of the liquid and perhaps its container. Liquid compressibilities and speeds of sound are available for such water hammer calculations. Alternatively, the fluid may be assumed to be thermodynamically compressible. If the time step is kept small enough to prevent integrating past such rapid events, FLUINT can be applied to pipe-burst or induced-load studies. Using such an approach, excellent comparisons have been made with both test data and with "method of characteristic" codes that are specifically intended for water hammer analyses. Refer to the COMPLQ and WAVLIM routines described in Section 7.11.9.6, and to the COMPLIQ option for user-defined fluids in Section 3.21.7.

Component Models. FLUINT includes general device models for a wide variety of common fluid system components. The user can tailor these models for specific devices. Users can even model new or unique devices by accessing program variables with their own, simultaneously executed Fortran logic. The basic building block style network elements were specifically designed for such usage.

Similarity to and Compatibility with SINDA. The basic methods in FLUINT contain many analogies to the finite-difference and lumped-parameter usage of SINDA, which should help the SINDA user learn FLUINT easily. FLUINT may be thought of as a fluids co-processor to SINDA. Analyses may be all-fluid, all-thermal, or both fluid and thermal simultaneously. FLUINT, along with SINDA, dynamically sizes the processor for minimum overhead, and allows user control and user logic.

FLUINT Limitations—The user should be also aware of current program limitations. These limitations can be divided into two classes. First, there are some limitations that may be circumvented by extensive user logic if the user is sufficiently experienced, but which are not yet available as prepackaged options. Second, there are certain program capabilities that cannot be appended within user logic, but may eventually be added to FLUINT as growth options. The subsequent paragraphs will briefly mention alternatives for advanced users if any alternatives exist.

Internal Fluid Systems. The program is intended for 1D piping networks and can be used on certain classes of 2D and 3D flow problems (such as flow in porous media, flow between sheets, etc.), but not for external flow or most 2D and 3D internal flows. Quasi-stagnant control volumes *can* be subdivided in 2D or 3D, perhaps as needed to model thermal stratification.

Working Fluid Choices. The default internal heat transfer and pressure drop correlations are not applicable to superfluid helium or liquid metals, although the correlations can be altered or replaced by users since the basic solution methods *are* applicable to such fluids. Any other fluid or mixture can be analyzed, providing the user develops a reusable description of the fluid. (A large selection of existing descriptions is available.)

Fluid Mixtures. When employing mixtures of working fluids, only one substance may exhibit phase change (although an arbitrary number of gases can dissolve into the liquid phase). In addition, axial diffusion is neglected. Mixtures assume for the most part noninteracting moderate to low-pressure fluids and apply an assumption of additive pressures (Dalton's, and Dalton-Gibb's) for gaseous phases, and independent solubilities (but not diffusion rates) for binary solvent/solute pairs.

Supersonic Flow and Traveling Shocks. The code is intended for viscous flows in ducts. Certain compressible flow phenomena such as traveling shocks and supersonic flows* are not included, although most other effects such as kinetic energies, choking (Section 3.18), acoustic waves (Section 7.11.9.6), high speed heat transfer, and frictional heating are either automatically included or can be optionally simulated. Fluid states are assumed to be static conditions (e.g., the thermodynamic state that an observer traveling with the fluid would measure) unless otherwise stipulated by the user.

3.1.2 The Fluid Submodel Concept

FLUINT uses a different type of submodel: the *fluid submodel*. Traditional SINDA networks composed of nodes and conductors are called *thermal submodels*. Master models may be composed of fluid submodel(s), thermal submodel(s), or both. No single submodel may have attributes of both thermal and fluid submodels; a coupled thermal-fluid analysis must therefore have at least one submodel of each type. As with thermal submodels, fluid submodels are semi-independent entities that have their own inputs, logic, and control. Provisions have been made, however, for allowing heat energy to pass between fluid and thermal submodels, as will be described later.

There is one important difference between the two types of submodels: while thermal models may be divided and combined arbitrarily into submodels and still produce the same results, fluid submodels are distinct and indivisible. While thermal submodels may be interconnected by common conductors, fluid submodels cannot exchange fluid with one another. The rule of thumb for determining what should be modeled as a fluid submodel is therefore "one fluid system, one submodel." Even if two fluid systems use the same working fluid description or the same mixture, if they do not share or exchange fluid, they should still be input as separate submodels for efficiency. This is not considered a restriction in usage because it enforces a logical breakdown for complex, multi-fluid systems.

One last distinction must be made between the two types of submodels. Fluid submodels are specialized to thermal/hydraulic analyses and have little capability for handling conduction and no capability for handling radiation. Thermal submodels, in turn, are specialized for conduction and radiation, and do not adequately handle fluid analysis tasks. Therefore, the scope of the fluid model

* Mach numbers up to approximately 1.2 are tolerated without numeric instabilities.

is the physics *within* the fluid container walls, while the scope of the thermal submodel is the physics *outside of and perhaps including* the fluid container walls. This specialization is a key ingredient in the efficiency and generality of SINDA/FLUINT.

FLUINT automatically calculates fluid properties. Therefore, the code must know which system of units is being used, and all submodels (both fluid and thermal) must use these units (as specified in OPTIONS DATA (Section 2.7). Conversion factors may be used to translate portions of the model into the master unit set.

3.1.3 Networks and Elements

The longevity and widespread use of SINDA is testimony to its flexibility. In SINDA, a thermal problem is treated by breaking the problem into two halves: energy transport (conductors) and energy storage (nodes). (Arranged in orderly meshes, the lumped parameter equations represent the *finite difference* approximation to the conduction equations. With suitable conversions such as those provided by C&R Thermal Desktop®, SINDA can also be used to solve the finite element equations just as easily.) Nodes and conductors are the elements with which mathematical models of real thermal systems are constructed. Nodes communicate with each other via conductors in carefully constructed *networks*. This allows the real system to be approximated with any degree of resolution desired. However, from the program's viewpoint, the organization of networks cannot be predicted beforehand, so the program must be organized to handle "arbitrarily" constructed networks of the basic elements.

FLUINT works the same way. However, in addition to energy there are at least two more variables in a lumped parameter approach for fluid networks: mass (of each constituent) and momentum. Energy and momentum of each *phase* may also become independent variables if either homogeneous or equilibrium flow is an inappropriate assumption. This increase in the number of variables and the complexity of the underlying phenomena can considerably complicate the analysis. Consequently, FLUINT has a greater number of input options than does SINDA, and the computational cost per network element is also much greater. However the basic approach is the same: FLUINT treats a fluid problem by breaking it into two halves: energy and mass transport (*paths*), and energy and mass storage (*lumps*). Fluid networks are, therefore, arbitrarily constructed networks of paths and lumps sharing a common working fluid. Energy exchange between thermal and fluid submodels is handled by *ties* between nodes and lumps. Energy exchange *within* a fluid system is handled by *fties* between lumps. Flexible boundaries between certain lumps may be modeled using interfaces (*ifaces* for short).

Lumps are the fluid analogs of nodes. While nodes are characterized by temperatures and perhaps thermal capacitances, lumps are characterized by temperatures, pressures, constituent concentrations, qualities and perhaps volumes. Lumps are assumed to always be perfectly mixed and to exist in thermodynamic equilibrium; *a lump has only one characteristic thermodynamic state*. (To deviate from perfect mixing within two-phase control volumes, see Section 3.25.)

Paths are the fluid analogs of conductors. However, paths can be very complicated, and this analogy is not as close as the one between lumps and nodes. Lumps communicate both mass and energy with each other via the flow rate in paths, while nodes communicate energy with each other

via the energy transport in conductors. Path flows are assumed one-dimensional, and phase velocities are assumed equal (homogeneous or zero slip); *a path has only one characteristic flow rate.* (To model slip flow within two-phase ducts, two parallel paths may be input. These pairs are called *twins*. See Section 3.17 for more details on slip flow modeling.)

To further aid in the distinction between lumps and paths, it should be noted that no specific flow rate or geometric shape can be attributed to a lump, and that no specific thermodynamic state (or mass or heat rate) can be attributed to a path. This is the same as saying that a node does not have a conductance, and that a conductor does not have a temperature. This distinction is important in fluid analysis because of its effect on heat transfer calculations, as described in a later section.

Ties are very much like linear conductors, in that they allow only heat energy to pass from higher temperature sources to lower temperature sinks according to a conductance based on the desired heat transfer phenomena. *One side of a tie is always a thermal node, and the other is always a fluid lump.* Without ties, fluid and thermal models would be independent of each other.

Fties (fluid to fluid ties) are even more analogous to conductors: they transport energy from lump to lump just as linear conductors transport energy from node to node. However, unlike conductors, fties can only connect lumps within the same fluid submodel: no intermodel ftie exists.

Ifaces are somewhat like paths in that they interconnect lumps. However, unlike a path no mass flows through an iface. Rather, the iface describes how the boundary between any two control volumes (“tanks”) behave. This boundary can be physical (such as a membrane, a liquid/vapor interface, or a piston) or it can be conceptual (such as an arbitrary subdivision of a larger control volume).

The user should note that FLUINT always calculates its own maximum allowable time step, and that steady-state solutions may be achieved either by taking long artificial time steps until the network is relaxed (STDSTL), or more commonly by iteratively relaxing the network (STEADY, aka “FASTIC”). The size of time-dependent elements (tanks and tubes) will determine the size of the real or artificial time step that can be taken—the larger the element, the larger the time step. See Section 4.5.2 for a discussion of time step control.

The following subsections describe the fluid elements in more detail. As a summary, Table 3-1 contains an abbreviated description along with the analogies with thermal elements. Figure 3-1 details the hierarchy of FLUINT elements.

Table 3-1 Summary of FLUINT Network Elements

ELEMENT	TYPE	ANALOG	DESCRIPTION
Lump	Tank	Diffusion Node	Control volume, time-dependent
	Junction	Arithmetic Node	Zero volume lump, time-independent
	Plenum	Boundary Node	Infinite volume lump, time-independent
Path	Tube	no analog	Line segment with significant inertia, time-dependent
	Connector	Linear Conductor	Time-independent, multi-purpose path (insignificant inertia). Subtypes (“devices”) are available for pumps, valves, ducts, etc.
Tie	HTU,HTUS	Linear Conductor	User input conductance
	HTN,HTNC	Linear Conductor	Conductance calculated according to
	HTNS, HTP		convection correlations
Ftie	USER	Linear Conductor	User input conductance
	AXIAL	Linear Conductor	Axial conduction along a duct element
	CONSTQ	no analog	Constant heat rate
Iface	various	no analog	Boundary between control volumes, either time-dependent or not

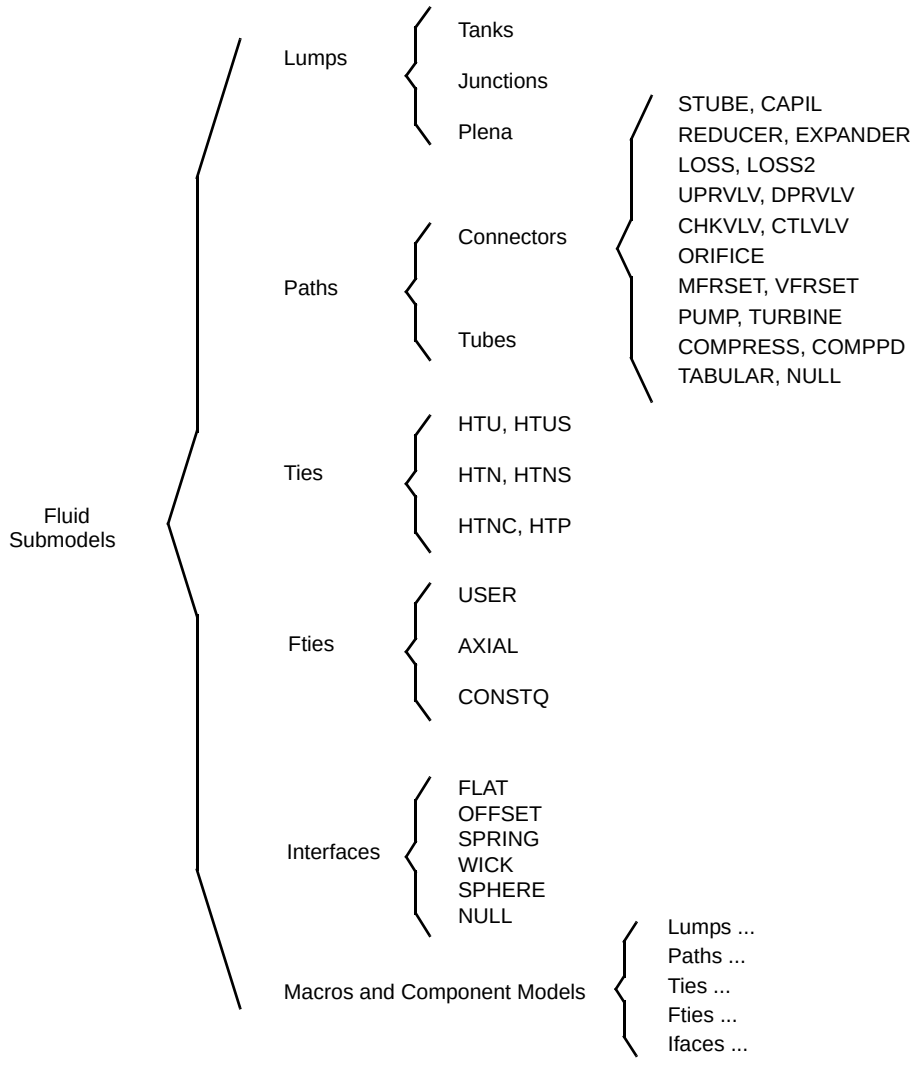


Figure 3-1 Hierarchy of FLUINT Network Elements

3.2 FLOW DATA Overview

This section and those following it describe the input methods and formats for FLOW DATA, which is the main input block for fluid submodel descriptions. The reader should note that the input rules for fluid data differ significantly from thermal data input rules. The FLOW DATA input rules were selected to reduce the amount of work ultimately required by the user, although an initial amount of effort is needed to learn these rules (if a GUI is not used instead).

SINDA network inputs (NODE and CONDUCTOR DATA, etc.) are line oriented, despite the ability to continue inputs over more than one line. In contrast, FLOW DATA is subblock oriented, where a *subblock* is like a miniature header block within the FLOW DATA block. Each fluid submodel is completely described within one HEADER FLOW DATA block, and network elements may be input within that block in any order.

3.2.1 General FLOW DATA Input Rules

A fluid submodel represents a fluid system that contains either a single fluid substance or a mixture consisting of a prescribed set of substances (“constituents”). Fluid submodels cannot exchange fluid with any other submodel. Each fluid submodel is described within an input section that begins with a HEADER FLOW DATA card. Following this header card, the user specifies the submodel name and desired working fluid(s), the fluid network, initial conditions, and calculation options. There are five basic types of *subblocks* within this section that may appear *in any order* to facilitate reading from a schematic. These subblocks begin with reserved letters in column 1 (and sometimes extends into column 2), and terminate when another subblock or header record is encountered. The five subblock types are:

LU . . . . . Lump input subblock
PA . . . . . Path input subblock
T. . . . . Tie subblock, used to specify energy exchange with thermal nodes
FT. . . . . Ftie subblock, used to specify energy exchange between lumps
IF . . . . . Iface subblock
M . . . . . Macro or Model subblock (for generation of elements or component models)

These subblock identifiers (LU, PA, T, FT, IF, and M) must appear in column 1. The remainder of the inputs may be given in any column and may extend to several lines. The general format of any fluid subblock is:

```
id option, required input #1, required input #2,...
       [,keyword=value] [,keyword=value]
       [,keyword=value]  ...
```

where:

`id` . . . . . subblock identifier (LU, PA, T, FT, IF, or M)
`option` . . . . . name of the subblock type (such as TANK, TUBE, etc.)

Following the *option* specification is a list of required inputs (i.e., inputs with no definable defaults) that are separated by commas. Following the required inputs, the user may specify any options that are specific to this *id* and *option*. These optional inputs are identified by *keywords*, and may occur in any order (with one exception noted in Section 3.4.2). The user may specify some, all, or none of these inputs. In the format descriptions given in the next section, different values will be expected depending on the keyword used:

Value	Expected Input
R	Real (floating point) input *or expression* expected
I	Integer input expected (expressions allowed in limited cases)
C	Character (alphanumeric) input expected, without quotes
A	User array ID ([smn.] 21, 3, etc.) expected.

Full expressions (with perhaps references to registers and processor variables, per Section 2.8) are allowed and should be used wherever possible.

A backslash is not necessary to continue a subblock. Subblocks may extend over several records, and are terminated when a new subblock or header is encountered. Whole lines may be specified as comments by beginning the line with a C in column 1. These lines are ignored by the program. A dollar sign ($) similarly causes all subsequent information on a line to be ignored, allowing comments at the end of the line. Comments do not terminate the current subblock.

In order to reduce the amount of input required, the user has the option of using *DEF* ("define" or "default") subblocks. With these subblocks, the user may preset values for information that would otherwise need to be repeated for each element. All input for DEF subblocks is optional* and therefore keyword-driven. There are two types of DEF subblocks: one for lumps (LU DEF) and one for paths (PA DEF). Any values defined in DEF subblocks become default values for all subsequent input in the current submodel. The user may use DEF subblocks as often as needed.

T subblocks (Section 3.6) are used to link together nodes and lumps and to select the methodology for heat transfer between fluid and thermal submodels. Six options are available: HTN, HTNC, HTNS, HTP, HTU, and HTUS. A seventh type, HTM, is an internal designation that is generated by certain macrocommands.

FT subblocks (Section 3.7) are used to specify heat exchange between lumps, usually conduction within the fluid itself. Three options are available: USER, AXIAL, and CONSTQ.

IF subblocks (Section 3.8) are used to generate interfaces ("ifaces") that describe how two adjacent tanks interact or how their control volume boundaries are interrelated.

M subblocks are used to generate many elements in subnetworks, similar to the GEN options in thermal input sections. Like GEN options, the generated elements may be independent (such as a string of lumps and paths in a subnetwork). Unlike GEN options, the generated elements may instead be logically linked to form *component models* (such as ducts and capillary evaporator pumps).

* Note that some keyword-driven "options" are only optional *if they have been defined previously in the current submodel using a DEF subblock.* An error will result if no default value has been defined and no input is given in the current subblock.

3.2.2 HEADER FLOW DATA Record

Unlike thermal submodel inputs, almost all inputs for fluid submodels (network element descriptions and source terms) are input under *one* HEADER block. Within this block, subblocks describing network elements may occur in any order. This style facilitates walking through fluid system schematics.

In the FLOW DATA block, the user defines the name of a new fluid submodel and identifies the working fluid(s) that it will contain. The format for this HEADER record is:

```
HEADER FLOW DATA, smn ,FID=I [,FIDx=I]
       [,NWORK=I]
```

where:

`smn` . . . . . fluid submodel name (must be unique, up to 32 characters)

`FID` . . . . . fluid identification number (ASHRAE number or user fluid ID number—see also Appendix C and Section 3.21).
If multiple-constituents are defined, "FID" is equivalent to "FIDA."*

`FIDx` . . . . . identification number of additional fluid constituents, where "x" is "A" through "Z" (i.e., "FIDA" through "FIDZ"). Up to 26 constituents can be defined, one (and only one) for each letter of the English alphabet. If more than one fluid substance is defined, a maximum of one can be a library fluid or a 6000 or 7000 series two-phase fluid: all others must be either 6000 NEVERLIQ fluids (real gases), 8000 series perfect gases, or 9000 series nonvolatile liquids (refer to Section 3.21).
Do not input "FID=" if "FIDA=" has been input, or vice versa.
Refer to Section C.4 for information on the temperature and pressure range limits that are applied to fluid mixtures.

`NWORK` . . . . . sparse matrix package workspace size for this submodel. *Ignore this input* unless directed to specify it by a processor error message. Otherwise, the processor will abort if this value is input and is too small.

defaults:

`FIDx` . . . . . none (single constituent)

`NWORK` . . . . . calculated by the program

Examples:

```
HEADER FLOW DATA, LOOP, FID=11

HEADER FLOW DATA, ATCS, FID=6070
```

* *Note:* In prior versions, the default FID was "718," an internal water description that has since been abandoned due to limitations and inaccuracies. In the current version, FID has no default. Furthermore, 718 is no longer a valid input. See VERSION 5.4 vs. VERSION 5.3 for more information and alternatives.

```
HEADER FLOW DATA, MIXD, FIDG=8278
      FIDL  = 9718,     FIDP  = 9404
      FIDA  = 717
```

3.3 Lumps

Lumps are subdivided into three types: *tanks*, *junctions*, and *plena*. Tanks are finite control volumes that may exchange energy with the environment and may grow or shrink. Because they are governed by differential equations for mass and energy in time, tanks cannot change instantaneously. Junctions can be conceptualized as zero volume tanks that may also exchange energy with the environment. Junctions react instantaneously because they are governed by algebraic (time-independent constraint) equations for mass and energy. They may be used to model tees, small tanks, or points where flow information is needed. Plena (plural of "plenum") can be conceptualized as tanks with infinite volume. They represent boundary conditions for modeling open systems.

Each lump has a single characteristic thermodynamic state, and therefore inherently presumes perfect mixing. To simulate lack of thermal equilibrium between phases when liquid and vapor or gas are both present, pairs of tanks ("twinned tanks") may be input, invoking nonequilibrium simulations.

Thermodynamic States: FLOW DATA—All lumps have a single specific thermodynamic state. This state must be initialized by the user in FLOW DATA, but can be customized in logic blocks via calls to the CHGLMP routine and related routines as described in Section 7.11.1.1.

While many properties describing the lump state are available for the user's convenience in logic blocks, there are only three properties needed to determine the lump state initially in FLOW DATA for single-constituent working fluids. These input properties are:

TL . . . . . Lump temperature (static, unless LSTAT=STAG option is used)
XL . . . . . Quality (vapor mass fraction: 0.0 for 100% liquid, 1.0 for 100% vapor or gas)
Alternate: AL . . . . . Void fraction (vapor *volume* fraction)
PL. . . . . Lump pressure (static, unless LSTAT=STAG option is used)

TL and PL must be input at least once (i.e., they have no default value until assigned one in an LU DEF subblock). When these values are missing from the examples that follow, they are assumed to have been set in a prior LU DEF subblock.

While only two properties (TL, PL) are required to completely specify a single-phase single-constituent state, the pressures and temperatures are not independent for two-phase states of pure substances per Gibb's Phase Rule. Therefore, a third property, either quality (XL) or void fraction (AL), is also needed. Since there is a potential conflict between these three properties for single-phase states (where only two are required), *by default FLUINT assumes that TL and XL (or AL) are always correct, but that PL may be a first guess and hence will be overridden if it conflicts with TL and XL (or AL).* In other words, a hierarchy exists:

1) XL (or AL) is always used. If AL is specified, XL is ignored.
2) TL is used by default
3) PL is overridden if it conflicts thermodynamically with TL and XL (or AL)

Table 3-2 summarizes the manner in which states are specified. The reason for this method is to allow users to specify all possible states without having to keep track of saturation pressures for two-phase states. Thus, the user can force FLUINT to initialize the lump in a saturated state by providing an unrealistically high PL (say 1.E30) if XL isn't 0.0,* or an unrealistically low PL (say 0.0) if XL isn't 1.0. Changes to the state can be made later in OPERATIONS using the CHGLMP routine before calls to any solution routines.

Table 3-2 Lump State Initialization (without "PL!")

Desired State	Input Quality (or Void Fraction)	Input Pressure
Subcooled Liquid	XL = 0.0	PL > P_{sat}(TL) *
Saturated Liquid	XL = 0.0	PL ≤ P_{sat}(TL) **
Two-phase	0.0 > XL > 1.0	anything **
Saturated Vapor	XL = 1.0	PL ≥ P_{sat}(TL) **
Superheated Vapor	XL = 1.0	PL < P_{sat}(TL)

* P_{sat}(TL) is the saturation pressure corresponding to TL.

** PL will be replaced with P_{sat}(TL) in these cases. NOTE: 6000 series fluids permit supercritical vapor, so if XL=1.0, then use PL > PGMAX or PL < P_{crit} for saturation, else P_{crit} < PL < PGMAX for supercritical.

Alternatively, the user may replace this hierarchy by another using "PL!" instead of "PL" within FLOW DATA blocks. *(PL! is illegal within logic blocks.)* By placing the exclamation mark after the "PL," the user may signal that the input pressure value is to be given priority over the input temperate value, which may be overridden if it conflicts with PL and XL (or AL):

1) PL! is always used

2) XL (or AL) is always used (except for single-phase working fluids)

3) TL is overridden if it conflicts thermodynamically with PL! and XL (or AL)

Thus, the user can force the use of a certain initial pressure. (The program is often more sensitive to initial pressures than to initial temperatures.) Arbitrarily high or low TL values can then be used to specify saturated states in a manner analogous to Table 3-2.

PL! may be specified in LU DEF blocks, in which case that value applies to all following lumps in which *neither* PL nor PL! is specified: a local subblock value of PL is assumed to have priority over a default value even though that local value may be overridden if inconsistent. The exclamation point can only occur in FLOW DATA for initial conditions, and has no meaning in logic blocks.

* In the special case of XL=1.0 (or AL=1.0) when using 6000 series fluids (Section 3.21.7), the pressure will not be overwritten even though it is greater than critical because those user-defined fluids permit supercritical vapor pressures: arbitrarily high PL will not conflict with TL and XL=1.0. In that case, specify PL higher than saturation *but less than the critical pressure* to force the initialization of the lump at a saturated vapor state, or specify a quality slightly less than unity, perhaps XL=0.9999, or simply specify PL=1.0E30 ... a pressure higher than any possible vapor state, and therefore intended to be overwritten with saturation.

XL (or AL) is always obeyed unless the working fluid is inherently single-phase (refer to Section 3.21), in which case XL and AL and ignored as redundant and PL is applied as input.

When PL, PL!, XL (or AL), or TL are specified by register or process-variable-containing expressions, the order in which these expressions are updated is pseudo-random and special cautions apply, as explained in the next subsection.

Note that the fluid submodel initial conditions need not represent real conditions—they may simply be guesses at a true set of initial conditions that are to be passed to the steady-state solution routine. (The solution routine STEADY is specifically intended to handle such rough guesses.) However, users will generally find it is worth their time to specify reasonable initial conditions—the program may not be able to recover from highly inconsistent or extreme initial conditions.

Thermodynamic States of Plena: Expression-based Updates—Plena are boundary conditions: their states are considered program *inputs*, where as the states of junctions and tanks are *outputs* of the solution routines. The state of a plenum (Section 3.3.3) can therefore not only be *initialized* by expression-based inputs (Section 2.8) based on the rules described in this section, its state will also be adjusted *during program execution* if changes are made to underlying registers or processor variables used to define PL, TL, XL (or AL), or species mass fractions. This means that the state of a plenum can change as a function of time, for example, or that a parametric sweep can be made on its temperature or pressure. However, the rules for these dynamic updates differ from those used for initializations, as follows:*

1. Whether or not temperature or pressure is assigned priority (i.e., PL vs. PL!) no longer matters after initialization. If there is a change to the current value for the expression for temperature (or pressure), then that temperature (or pressure) will change and XL and AL will be held constant ... as always. This rule might result in the pressure value being overwritten, for example, *even if it were originally defined using PL! and even if the new value differs from that of its defining expression.*
2. If the quality (XL), void fraction (AL), or a species mass fraction is changed, PL is held constant to minimize disruption to the network, *whether or not temperature was assigned priority during initialization.*
3. If both temperature and pressure values change, and if those changes conflict with each other (usually for saturated or two-phase states), then it is not possible to predict which update will occur first. An explicit call to CHGLMP may need to be inserted to guarantee a desired result (e.g., a parametric march along the saturated vapor line).
4. Recall that, like any other program input defined by expressions, the temperature or pressure will not be updated just because the current value becomes different from the underlying expression (perhaps due to the lack of update priority, as stated above). Only a change to the value of the underlying expression will trigger such an update.

* Note that these rules are only important when using working fluids that are condensible/volatile species, or are mixtures that contain such a species.

Thermodynamic States: Multiple-Constituents—If more than one fluid species or constituent is present within a submodel, then the initial concentrations of each constituent must also be defined in order to completely specify the thermodynamic state. These concentrations are specified using mass fractions, as follows:

XFx . . . . . Mass fraction of constituent “x” in liquid phase, where “x” is A through Z
XGx . . . . . Mass fraction of constituent “x” in gas phase, where “x” is A through Z

Both of these variables represent a fractional partioning of the liquid or vapor phases in each lump. Therefore, the sum of XFx over all constituents “x” will be unity for a liquid-containing lump (and zero otherwise). Similarly, the sum of XGx over all constituents “x” will be unity for a gas- or vapor-containing lump (and zero otherwise).

For these mixtures, if no condensible/volatile constituent is specified, then no saturation relationship exists and PL, TL, and XL (or AL) are always used. However, if a condensible/volatile constituent *is* specified (e.g., air/water mixtures), the situation can become rather complex, as will be described subsequently.

XL retains its original meaning: the mass of vapor constituents to the total mass of all fluids within the lump. Input values of XGx and XFx are ignored if they are inconsistent with the value of XL (or AL) or the type of fluid implied by HEADER FLOW DATA list, and errors will be produced if they refer to species that were not named in the HEADER FLOW DATA subblock.

If these values are over-specified or under-specified, special apportioning logic will be applied. For example, if XGA=0.75 and XGF=0.5 (i.e., a sum greater than unity) has been specified for a lump, then the initial values will be: XGA=0.6 and XGF=0.4 to maintain the requested *proportion*.

If, on the other hand, values of XGA=0.25 and XGF=0.5 (i.e., a sum less than unity) had been input and *no other gaseous species were present*, then the initial values will be: XGA=0.3333333 and XGF=0.6666667. If other species were present, then the unspecified portion (25%) would have been divided evenly amongst them. In other words, if four gaseous species are named and no value of XGx had ever been input, then the initial values would all be set to 0.25 for all lumps, dividing them evenly throughout the model.

Although seemingly complicated, these rules were developed to avoid errors and inconsistencies, and yet require a minimum of input.

Thermodynamic States: Multiple-Constituents with a Condensible/Volatile or Soluble Substance—The above rules that apply to the generation of initial lump states (given partial or complete specification of PL, TL, XL, AL, XG*, XF*, etc.) become more complicated if one of the species in the mixture is condensible. The reason for the added complication is that it is easy to specify a nonsensical state. Therefore, the user should expect that the initial state of a lump, when the processor starts, will need to be verified and that calls to CHGLMP or CHGLMP_MIX (perhaps several calls per lump) may be required to achieve a specific initial condition.

The initial values of TL and XL (or AL) will be preserved if at all possible during lump initialization. (If PL! is used, then PL and XL or AL will be preserved.) Values of other input parameters will be preserved if they don’t conflict, or if they don’t cause property range limits to be reached.

If a gaseous species is soluble (Section 3.23), then *by default the initial amount of dissolved gas is zero.* If instead the user has directly specified the value of XF for the dissolved species, then that initial concentration will be used. However the user is cautioned against specifying a supersaturated initial condition since that state can lead to immediate nucleation. In the event of excessive concentrations, property calculation errors can occur.

There is no simple way to specify 100% saturated solute, nor any other fraction of dissolution saturation. Instead, estimate the desired XF for the solute, then make a preliminary run with a call to CNSTAB (Section 7.11.8) to find the saturation fraction.

Special rules apply to the XG of condensible species ($XG_{condensible}$). Attempting to rigorously preserve the input values of XG or XF often results in unrealistically large or small temperatures or pressures. For example, if any volatile liquid is present and noncondensible gas is present, then $XG_{condensible}$ cannot be zero: it will instead be changed to the value corresponding to saturation at either the lowest temperatures or the largest pressure. (If $XF_{condensible}$ had been zero, on the other hand, $XG_{condensible}$ could have been arbitrarily small.)

Therefore, the initial value of $XG_{condensible}$ is often assumed to have been input casually if it has been defaulted, and it will often be adjusted as necessary to preferentially preserve the input values of PL and TL and other parameters. This treatment often results in changes to the XG of other gaseous species as well.

If no volatile liquid has been specified, and if the $XG_{condensible}$ has been input as a small number (namely, subsaturated), then the requested input state is preserved. Otherwise, if the input value of $XG_{condensible}$ has not been defaulted to be 1/(M+1) (where M is the number of 8000 series fluids), then the value of PL (or TL if PL! is used) will be varied until the input value of $XG_{condensible}$ is achieved. If a range limit is reached, the value of $XG_{condensible}$ is then reset to saturation at that limit.

If the user truly needs a specific value of $XG_{condensible}$, and to preserve that value (permitting PL and TL to change if necessary), the CHGLMP or CHGLMP_MIX routines (Section 7.11.1.1) should be used at the start of OPERATIONS to correct for inappropriate initializations.

Finally, it should be noted that the resulting state may differ slightly from the specified state. For example, the quality of the lump might be changed slightly over what was input. Again, such minor adjustments are considered expedient since most initial conditions are rough guesses. The user can employ the CHGLMP or CHGLMP_MIX routines if that presumption is incorrect, and if precise initial conditions are required.

Air/Water (and other Psychrometric) Mixtures—Utilities such as HUMR2X (described in Section 7.11.2.4) are available to help calculate inputs such as XG (absolute humidity) of a condensible phase given relative humidity or dew point.

Thermodynamic States: References in Logic Blocks—As mentioned before, there are many variables describing the thermodynamic state that are available within user logic blocks. These variables, which are referenced in a manner analogous to the way thermal node parameters are referenced, are as follows:

TL . . . . . Lump temperature (user-defined units: °F, °C, K, or R)
PL . . . . . Lump static pressure (user-defined units: psia, psig, Pa, etc.)
XL. . . . . Lump quality (ratio of vapor mass to total mass; dryness fraction)
AL. . . . . Void fraction (ratio of vapor volume to total volume)
DL. . . . . Lump density (kg/m^3 or lb_m/ft^3)
HL. . . . . Lump enthalpy (intensive, J/kg or BTU/hr-lb_m)
DG . . . . . Density of vapor or gas phase (zero if none present)
DF. . . . . Density of liquid phase (zero if none present)
XFx. . . . . Mass fraction of constituent “x” in the liquid phase, where “x” is A through Z
XGx . . . . . Mass fraction of constituent “x” in the gaseous phase, where “x” is A through Z
MFx . . . . . Liquid molar fraction of constituent “x”, where “x” is A through Z (note MF is a real, not integral, value)
PPGx. . . . . Gas pressure fraction of constituent “x”, where “x” is A through Z

These properties should not be altered within user logic blocks without the use of the CHGLMP or CHGLMP_MIX routines. They are available to the user for control or as input to calculations. There are many such variables in FLUINT. Altering their value may produce unexpected results.

QDOT and QTM, QCHEM, Heat Exchangers—The specified heat input rate, QL, is one of many sources of energy that can be added or extracted from a lump. Heat can flow to or from thermal nodes via *ties* (Section 3.6), or to or from other lumps conductively or convectively* via *fties* (Section 3.7). It can also be generated by chemical reactions (QCHEM, Section 3.24) and flow into it from heat exchangers (HXMASTER, Section 7.11.10.3) . It can also be generated or extracted within flow *paths* (Section 3.4) as the result of the action of machinery (pumps and other turbomachinery as well as rotating passages in which shaft power can be added or subtracted). The total of all such “turbomachinery” power† is available as the lump output parameter QTM. The total of all heat loads (QL, QTM, QCHEM, and the sum of tie and ftie heat rates and heat exchangers) is available as the output parameter QDOT. All such heat loads use the convention of positive values representing energy added to the lump, and negative values representing energy extracted from the lump.

Other Lump Parameters—Optionally, lumps may be assigned an elevation or coordinate location (CX, CY, CZ) in Cartesian three-space. These coordinates, combined with global acceleration vector components ACCELX, ACCELY, and ACCELZ, are used to calculate body forces resulting from gravity, launches, or orbital maneuvering. These features are described in Section 3.13.

* Heat flowing between lumps via advection (that is, as a result of mass transport) is the realm of *paths* (Section 3.4) and is not tallied into the QDOT term. See Section 7.11.10.3 for treatment of heat exchanger outlets.

† Specifically, QTM = ΣQTMK where QTMK is the turbomachinery power of each path currently flowing into this lump (this simplified equation assumes unit duplication factors).

Lumps have additional descriptive variables, depending on type, as discussed below. Recall that there are three basic types of nodes: diffusion (finite capacitance), arithmetic (zero capacitance), and boundary (infinite capacitance). There are three analogous types of fluid lumps: *tanks* (finite volume), *junctions* (zero volume), and *plena* (infinite volume). The differences between these types of lumps are described below.

Additional lump descriptors pertaining to modeling of dissolved substances are described in Section 3.23.

3.3.1 Tanks

Tanks are fluid lumps with a definite, nonzero volume.* Like all lumps, they have specific thermodynamic properties. Unlike other lumps, mass and energy within a tank change gradually with time; *tanks resist most sudden changes*. Therefore, the relative size of a tank might affect the size of the allowable time step—bigger tanks usually mean larger time steps during a transient solution. Tank volumes can expand or shrink at prescribed rates (independent of tank pressure), or may expand or shrink as a function of pressure, or both effects can be superimposed. Like the Q-source term in thermal networks, tanks have an analogous source/sink term. The additional descriptors for tanks are:

VOL . . . . . Tank volume
VDOT. . . . Rate of change of VOL with time (positive for growth)
COMP. . . . Tank wall compliance (volumetric spring rate with pressure)
QL. . . . . Heat input (source term)

VDOT, COMP, and QL may be altered within user logic blocks, but VOL should not be altered except with the CHGVOL routine (Section 7.11.1.1).† Tanks may not have negative volumes, so care should be taken in the selection of VDOT. The tank wall compliance (COMP) may be used to model gas inclusions, container flexibility, and liquid compressibility (if the liquid is otherwise incompressible), as described in later sections. Tanks can also be interconnected using ifaces, described in Section 3.8, which define how the walls between tanks are interrelated for modeling pistons, wicks, springs etc., or for subdividing control volumes.

Tanks can be used in one of two basic ways. First, as their name implies, they can be used to model a vessel, reservoir, or accumulator directly. Second and most important, tanks are control volumes—they can be used to simulate the energy or mass storage of pipe segments, heat exchanger segments, or any portion of the fluid system with significant volume. Note that if all tanks in a submodel are or can become hard-filled (subcooled or otherwise completely filled with incompressible liquid, such that XL=AL=0.0) and are stiff (COMP=0.0), *at least one plenum must be present.* The pressure in such hard-filled tanks will change instantly to respond to flow rate, volume, or even heat rate (density) changes.

Such hard-filled, noncompliant tanks (*hard tanks*) react much differently than *soft tanks* (i.e., tanks with any vapor, gas, or compressible liquid). Since the flow rates into hard tanks must always nearly balance, these tanks react faster than soft tanks to changes and their hydrodynamic response can be instantaneous if the liquid is assumed to be incompressible. Because of special treatment, hard-filled tanks can take much larger time steps and consume much less computer time per step than soft tanks. For this reason, soft tanks should be used more sparingly, and junctions should be considered as alternatives. However, pressures may change too quickly or extremely in a hard tank since there is no place to store or release any excess fluid. Tank wall compliance factors (COMP) and interfaces can be used to reduce this problem by removing the unrealistic but useful assumption

* Note that a large tank volume does not imply to the program that the velocity of fluid is small, since velocity has no meaning for lumps and more importantly, lump pressures are static by default. To apply an additional assumption of negligible velocity in the tank, use the LSTAT=STAG option.

† The VOL of a tank used within a FloCAD Compartment is initialized automatically.The CX, CY, and CZ values of such a tank are also updated automatically.

of inflexible walls. (Refer to the COMPLQ routine in Section 7.11.9.6 and to the compressible liquid option, COMPLIQ, for user-defined fluids in Section 3.21.7. Interfaces, as described in Section 3.8, may also be appropriate.)

Tanks may also be used in pairs ("*twinned*") to avoid the perfect mixing assumption within two-phase control volumes. Inputs related to that option are deferred to Section 3.25.

Tanks may also be interconnected using *ifaces* to model shared boundaries (whether real or imaginary). Refer to Section 3.8.

TANK Subblock Format:

```
LU TANK,id[,VOL=R][,TL=R][,XL=R or AL=R][,PL(!)=R]
        [,VDOT=R][,QL=R][,COMP=R][,XFx=R][,XGx=R]
        [,LSTAT=C][,CX=R][,CY=R][,CZ=R]
```

where:

- `id` . . . . . tank identification number
- `VOL` . . . . . initial tank volume
- `TL` . . . . . initial temperature
- `XL` . . . . . initial quality
- `AL` . . . . . initial void fraction (overrides XL inputs)
- `PL` . . . . . initial pressure, PL! means PL value given priority over TL
- `VDOT` . . . . . initial time rate of change of tank volume
- `QL` . . . . . initial rate of heat input. "QDOT" is the total resulting heat input (sum of QL and other effects) available in logic and expressions.*
- `COMP` . . . . . initial tank wall compliance factor
- `XFx` . . . . . initial mass fraction of constituent "x" in the liquid phase, where "x" is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock
- `XGx` . . . . . initial mass fraction of constituent "x" in the gas phase, where "x" is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock
- `LSTAT` . . . . . tank status option: NORM (static) or STAG (stagnation). See guidance in Section 3.3.3. Use LSTAT=STAG only if velocities are negligible, not just to force PL to represent stagnation pressures.
- `CX` . . . . . coordinate location on the X axis
- `CY` . . . . . coordinate location on the Y axis
- `CZ` . . . . . coordinate location on the Z axis

* Prior to Version 4.2, "QDOT" was the name of both the input term *and* the resulting total heat input.

defaults (if not previously defined in an LU DEF subblock):

XL..... use AL if it has been input, else zero if neither has been input
VDOT ..... zero (volume constant over time)
QL..... zero (note: QLs are not zeroed and summed as are nodal Qs). Zero QL does not mean that heat transfer cannot be added using ties (Section 3.6) for convection, or fties (Section 3.7) for conduction within the fluid.
COMP ..... zero (inflexible wall)
LSTAT..... NORM (static pressure, no entrance accelerations applied)
CX..... zero
CY..... zero
CZ..... zero

Guidance: Pressure spikes and notches may occur if tanks are hard-filled with incompressible liquid and if their walls are inflexible (COMP=0.0 and no active ifaces). A small compliance factor (see Section 3.14.5) can avoid such problems. Refer also to the simulation routine COMPLQ in Section 7.11.9.6. Interfaces, as described in Section 3.8, may also be appropriate (especially in capillary modeling).

Guidance: The LSTAT=STAG option is often appropriate for tanks if they represent a large reservoir or vessel, etc. and do not simply represent a constant pressure at a cross section within a pipeline. By default, "PL" is taken as a static pressure, and no accelerational entrance losses are applied. Specifying LSTAT=STAG, on the other hand, causes "PL" to be treated as a stagnation (total) pressure, and automatically adds acceleration losses to any suitable path* currently flowing out of the lump (visible in the last column of the TUBTAB output routine, Section 7.11.8). It also causes any choking calculations (Section 7.11.9.5) applied to such out-flowing paths to include modeling of the expansion process (i.e., from zero velocity). Finally, kinetic energy is also neglected at the entrance (but not exit) of out-flowing paths if LSTAT=STAG. To toggle the LSTAT option during execution, use the CHGLSTAT routine (Section 7.11.1.1). An "x" after the FR/FRC number in the output routines PTHTAB and TWNTAB (Section 7.11.8) means the an inlet acceleration has been applied to that path because of the use of LSTAT=STAG.

Do not use the LSTAT=STAG option just to find the current total or stagnation temperature and pressure. Instead, use the TOTALTP routine (Section 7.11.1.10) for such inquiries.

Guidance: References to processor variables within expressions may refer to "#this" as the lump identifier.

* "Suitable paths" means tubes, STUBE connectors, REDUCER and EXPANDER connectors, and LOSS-family connectors (LOSS, LOSS2, CHKVLV, CTLVLV, PRVLV, BPRVLV, ORIFICE) but excludes NULL, CAPIL, PUMP, TABULAR, VFRSET, and MFRSET connectors. If that path rotates (Section 3.26), note that a stagnant lump is defined in the absolute sense of a nonrotating, stationary reference frame.

Restrictions: A tank must be linked via paths to at least one but not more than 25 other lumps. *If all tanks in a network are or can become hard-filled with liquid and are inflexible (COMP=0.0 and no active ifaces to "soft" tanks), at least one plenum must be present or a fatal error will result.* In this sense, a "network" is any isolated or valved-off part of the submodel.

Examples:

```
LU TANK, 10, VOL=length*0.25*pi*DH42^2, TL=300.0
LU TANK, 20, QL=50.0, VOL=7.0, COMP=1.0E-5
LU TANK, 39, PL! = 3.0E6$ OVERRIDE TL, USE THIS PL
            XFW = 0.25  $ AMOUNT OF CONSTITUENT "W"
LU TANK, 2001, VOL = length*pervol, AL = 0.5
            COMP = 0.01/PL#this$ note #this = 2001
```

3.3.2 Junctions

Junctions are to tanks what arithmetic nodes are to diffusion nodes—they are "zero volume tanks." In all other respects, a junction is like a tank. It has a specific thermodynamic state and an energy source/sink term, QL, which may be altered within user logic blocks. However, because it has zero volume, a junction cannot grow or shrink nor can it store mass and energy. Junctions are time-independent: they are always at steady-state even during a transient analysis. All mass and energy entering a junction equals the mass and energy leaving at all times: *junctions react instantly to changes*. They therefore do not directly affect the allowable time step during transients.

Junctions may be used in three basic ways. First, they may be placed at any point in the fluid system where temperatures or pressures are needed, or where heat must enter or leave the system. Second, they may be used as tees, end caps (dead ends), or common points to branching lines. Third, they may be used as control volumes that react instantaneously. In short, they may be used to join paths together wherever the volume is negligible or has been attributed to a nearby tank.

There are three cautions to be used regarding junctions. First, a fluid submodel cannot be built entirely of junctions for the same reasons that a thermal submodel cannot be built entirely of arithmetic nodes.

Second, fatal errors will result if a closed loop exists in the network that is composed solely of junctions *and* if the heat inputs into that loop are not zero or do not exactly balance (except in STEADY). The user should therefore avoid such *junction loops*, two examples of which are shown in Figure 3-2. The reason for this restriction is actually the same reason why thermal models cannot be built entirely of arithmetic nodes: within each recirculating loop, there is nowhere for energy imbalances to be absorbed, causing the temperatures to spiral to positive or negative infinity. Note that junction loops depend on the actual flow rate direction, not the defined flow rate direction—what may not have been a junction loop at the start of the run could become one if some flow rates reverse. Remember, the user need not worry about this restriction if all of the junctions are adiabatic.

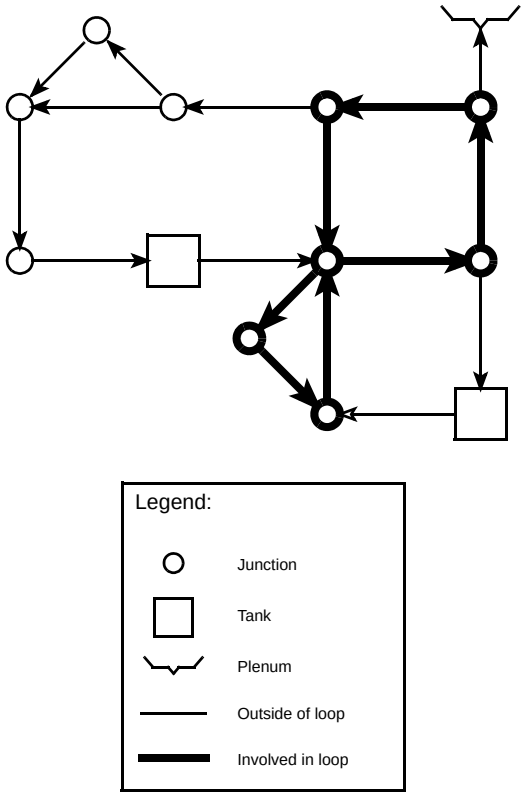


Figure 3-2 Examples of Junction Loops

Third, diabatic junctions are sensitive to flow rate changes. To conserve energy between thermal and fluid sub-

models in transient routines, the heat rate into all lumps is held constant over the time interval, as is the corresponding Q term on the thermal nodes. If the flow rate changes by more than a few percent during that interval, property routine error messages may result. While these messages are inconsequential if infrequent or not persistent, they may otherwise signal modeling problems.

At zero flow, junctions become mathematically undefined. Their thermodynamic state is irrelevant, and may hit extremes if flow rates oscillate about zero or to and from zero, causing error messages that are usually of no concern (because there is no significant energy route to or from the junction).

Junctions can also be used in a fashion similar to SINDA heater nodes via the HTRLMP routine (Section 7.11.1.4). Using HTRLMP, the enthalpy of a junction can be fixed (which usually means either the temperature or quality is nearly constant if the pressure does not change much), and the heat rate is updated to contain the heat rate required to hold that enthalpy fixed. Such "heater junctions" are very useful for modeling oversized heat exchangers, and for several modeling tricks where the actual heat transfer processes are not considered part of the problem. RELLMP may be used to "release" the enthalpy, returning the junction to a normal mode of operation. (HTRLMP may also be used on tanks within STEADY.)

JUNC Subblock Format:

```
LU JUNC,id[,TL=R][,XL=R or AL=R][,PL(!)=R][,QL=R][,VOL=R]
       [,XFx=R][,XGx=R][,LSTAT=C][,CX=R][,CY=R][,CZ=R]
```

where:

TL . . . . . initial temperature
XL . . . . . initial quality
AL . . . . . initial void fraction (overrides XL inputs)
PL . . . . . initial pressure, PL! means PL value given priority over TL
QL . . . . . initial rate of heat input. "QDOT" is the total resulting heat input (sum of QL and other effects) available in logic and expressions.*
VOL . . . . . "fake" (largely ignored) volume (see note below)
XFx . . . . . initial mass fraction of constituent "x" in the liquid phase, where "x" is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock
XGx . . . . . initial mass fraction of constituent "x" in the gas phase, where "x" is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock
LSTAT . . . . . junction status option: NORM (static) or STAG (stagnation). See guidance in Section 3.3.3. Use LSTAT=STAG only if velocities are negligible, not just to force PL to represent stagnation pressures.

* Prior to Version 4.2, "QDOT" was the name of both the input term *and* the resulting total heat input.

CX..... coordinate location on the X axis
CY..... coordinate location on the Y axis
CZ..... coordinate location on the Z axis

defaults (if not previously defined in an LU DEF subblock):

XL..... use AL if it has been input, else zero if neither has been input
QL..... zero (note: QLs are not zeroed and summed as are nodal Qs). Zero QL does not mean that heat transfer cannot be added using ties (Section 3.6) for convection, or fties (Section 3.7) for conduction within the fluid.
VOL ..... zero
LSTAT..... NORM (static pressure, no entrance accelerations applied)
CX..... zero
CY..... zero
CZ..... zero

Guidance: Junctions are less expensive to analyze than are tanks, and should be used instead of tanks whenever possible. Such substitutions are not *always* preferred.

Guidance: The volume "VOL" of a junction is ignored by almost all program options, since junctions have zero volume. However, it may be defined by the user as a convenience when switching back and forth between using junctions and tanks. It is also useful for certain options such as the SUMFLO or SUMDFLO routines (Section 7.11.1.10) when calculating the *virtual* volume or mass of a system (independent of the fact that the volume is neglected for modeling purposes). One option which *does* use junction volumes is the modeling of homogeneous nucleation of dissolved gases (see Section 3.23 and Section B.7.2).

Guidance: *Do not* use the LSTAT=STAG option just to find the current total or stagnation temperature and pressure. Instead, use the TOTALTP routine (Section 7.11.1.10) for such inquiries.

Guidance: References to processor variables within expressions may refer to "#this" as the lump identifier.

Restrictions: A junction must be linked via paths to at least one but not more than 25 other lumps. No submodel can be built entirely from junctions. Also, fatal errors will result if a closed loop exists in the network that is composed solely of junctions and heat rate values for the loop are not zero or do not exactly balance (except in STEADY).

Examples:

```
LU JUNC,3010,QL=410.0,XL=0.1
LU JUNC,14,  PL=1.0E7,  CX=1.0, CZ=2.0, CY=-3.0
LU JUNC, 202, AL = 1.0  $ ALL GAS
            XGD = 0.5   $ HALF OF WHICH IS CONST. "D"
LU JUNC,21, QL = 30.0*(RAD.T33 - TL#this)
```

3.3.3 Plena

Plena are the "boundary nodes" of the fluid network. A plenum has an infinite volume.* As such, it represents an inexhaustible source or sink of fluid; the thermodynamic state does not change. Plena do not affect time steps or solution speeds. They have no energy source/sink term. Their descriptors simply consist of thermodynamic properties. Plena may be used as boundary conditions for open systems, or as a simplified model of a very large tank whose response is not part of the problem at hand. Note that tanks and junctions may be temporarily made to act like plena using the HLDLMP and RELLMP routines (Section 7.11.1.4). Tanks may be specifically "held" as plena using the HLDTANKS routine, which preserves their initial conditions during a steady-state (in preparation for a transient) and letting any junctions and paths adjust to their states.

Like any lump, the state of a plenum may be changed using the CHGLMP routine (Section 7.11.1.1). Unlike any other lump, the state of a plenum is considered an *input*. (Tank and junction states, on the other hand, are normally *output* by the steady and transient solutions.) Therefore, the state of a plenum may be changed during a solution not only via CHGLMP, but also via expressions used for TL, PL (or PL!), and XL. Such expressions are used only for initializing tanks and junctions. For example, defining TL=*tinlet* (where *tinlet* is a register) for a tank or junction specifies only the initial condition: changes to *tinlet* during processor execution will *not* update a tank or junction state since to do so would conflict with the solution routines. For a plenum, on the other hand, changes to *tinlet* (perhaps during a parametric sweep, or as part of control logic) *will* change its state, and the rules applied during such dynamic updates are not necessarily the same as those for initialization, as explained above (see "Thermodynamic States of Plena: Expression-based Updates" on page 14).

PLEN Subblock Format:

```
LU PLEN,id[,TL=R][,XL=R or AL=R][,PL(!)=R]
        [,XFx=R][,XGx=R][,LSTAT=C][,CX=R][,CY=R][,CZ=R]
```

where:

`id` . . . . . plenum identification number
`TL` . . . . . plenum temperature
`XL` . . . . . plenum quality
`AL` . . . . . plenum void fraction (overrides XL inputs)
`PL` . . . . . plenum pressure, PL! means PL value given priority over TL

* This does not imply that the velocity of fluid in a plenum is zero, since lumps have no cross sectional area and therefore no velocity and more importantly, ***lump pressures are static by default***. Rather, a plenum represents a cross section of constant state. Fluid extracted from a plenum is considered already moving. If instead the plenum represents a zero-velocity stagnation state (and accelerational entrance losses must be applied to paths flowing out of the plenum), then use the LSTAT=STAG option (see guidance in this subsection).

XFx ..... mass fraction of constituent “x” in the liquid phase, where “x” is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock

XGx ..... mass fraction of constituent “x” in the gas phase, where “x” is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock

LSTAT..... plenum status option: NORM (static) or STAG (stagnation)

CX..... coordinate location on the X axis

CY..... coordinate location on the Y axis

CZ..... coordinate location on the Z axis

defaults (if not previously defined in an LU DEF subblock):

XL..... zero

LSTAT..... NORM (static pressure, no entrance accelerations applied)

CX..... zero

CY..... zero

CZ..... zero

Guidance: References to processor variables within expressions may refer to “#this” as the lump identifier.

Guidance: Although it can be used with any lump, the LSTAT=STAG option is often appropriate for plena if they represent a large reservoir, ambient, etc. and do not simply represent a constant pressure at a cross section within a pipeline. By default, “PL” is taken as a static pressure, and no accelerational entrance losses are applied. Specifying LSTAT=STAG, on the other hand, causes “PL” to be treated as a stagnation (total) pressure, and automatically adds acceleration losses to any suitable path* currently flowing out of the lump (visible in the last column of the TUBTAB output routine, Section 7.11.8). It also causes any choking calculations (Section 7.11.9.5) applied to such out-flowing paths to include modeling of the expansion process (i.e., from zero velocity). Finally, kinetic energy is also neglected at the entrance (but not exit) of out-flowing paths if LSTAT=STAG. To toggle the LSTAT option during execution, use the CHGLSTAT routine (Section 7.11.1.1). An “x” after the Mach number in the output routines PTHTAB and TWNTAB (Section 7.11.8) means the an inlet acceleration has been applied to that path because of the use of LSTAT=STAG.

Do not use the LSTAT=STAG option just to find the current total or stagnation temperature and pressure. Instead, use the TOTALTP routine (Section 7.11.1.10) for such inquiries.

* “Suitable paths” means tubes, STUBE connectors, REDUCER and EXPANDER connectors, and LOSS-family connectors (LOSS, LOSS2, CHKVLV, CTLVLV, PRVLV, BPRVLV, ORIFICE) but excludes NULL, CAPIL, PUMP,TURBINE, COMPPD, COMPRESS, TABULAR, VFRSET, and MFRSET connectors. If that path rotates (Section 3.26), note that a stagnant lump is defined in the absolute sense of a nonrotating, stationary reference frame.

Restrictions: A plenum may be linked via paths to any number of lumps. A warning will be produced if the plenum is isolated (i.e., unconnected).

Caution: Since many analyses begin with steady state solutions, the initial thermodynamic state of tanks and junctions are not as important as those of plena. For plena, the initial thermodynamic state *is* the final state (without user intervention in logic or expressions). For fluids with a condensible or volatile component (i.e., a fluid or mixture in which Gibb's Phase Rule applies), particular attention should be paid to the rules regarding the specification of thermodynamic state (see Section 3.3).

Examples:

```
LU PLEN,77, PL!=14.7, TL=200.0, XL=0.5, CZ=-10.0
LU PLEN,1929,XL = 0.0, XFH = 0.25, XFA = 0.5
LU PLEN,999, TL = (timen <= 30.0)? Tinit : Tfinal
      LSTAT = STAG$ This plenum is stagnant
```

3.3.4 Lump Defaults in FLOW DATA

As can be seen from the above format definitions, most lumps share the same parameters. To reduce inputs, the user may define the default definitions of these parameters within the FLOW DATA input stream in a separate subblock called LU DEF.

Once the value for a parameter has been specified in an LU DEF subblock, the program will use that value for all subsequent lumps if none has been supplied by the user. Any number of such subblocks may be used, overriding previous default values if needed, or merely adding new default values to the list.

If the program requires an input, it will search the previous LU DEF subblocks (in reverse input order) until that requirement is satisfied, or it will issue an error if no such input has been made. The parameters TL, PL and VOL must be defined at least once.

LU DEF Subblock Format:

```
LU DEF[,VOL=R][,TL=R][,TLADD=R][,TLFACT=R][,XL=R or AL=R]
       [,PL(!)=R][,PLADD=R][,PLFACT=R][,VDOT=R]
       [,QL=R][,COMP=R][,XFx=R][,XGx=R][,LSTAT=C]
       [,CX=R][,CY=R][,CZ=R]
```

where:

VOL ..... initial volume (for tanks and junctions)
TL..... initial temperature
XL..... initial quality
AL..... initial void fraction (overrides XL inputs)
TLADD..... temperature offset
TLFACT .... temperature multiplier
PL..... initial pressure, PL! means PL value given priority over TL
PLADD..... pressure offset
PLFACT .... pressure multiplier
VDOT ..... initial time rate of change of volume (for tanks)
QL..... initial rate of heat input (for tanks and junctions)
COMP ..... initial tank wall compliance factor
XFx ..... mass fraction of constituent “x” in the liquid phase, where “x” is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock
XGx ..... mass fraction of constituent “x” in the gas phase, where “x” is A through Z and corresponds to the fluid constituent defined in the FLOW DATA subblock

LSTAT . . . . . lump status option: NORM (static) or STAG (stagnation)
CX . . . . . coordinate location on the X axis
CY . . . . . coordinate location on the Y axis
CZ . . . . . coordinate location on the Z axis

defaults (if not previously defined in an LU DEF subblock):

TLADD . . . . . zero
TLFACT . . . . . one
XL . . . . . use AL if it has been input, else zero if neither has been input
PLADD . . . . . zero
PLFACT . . . . . one
VDOT. . . . . zero
QL . . . . . zero (Note: QLs are not zeroed as are nodal Qs)
COMP. . . . . zero
LSTAT . . . . . NORM (static pressure, no entrance losses applied)
CX . . . . . zero
CY . . . . . zero
CZ . . . . . zero

Guidance: An LU DEF subblock may be used to define default values for the lumps following the subblock. Any number of LU DEF subblocks may be used in any order to set or change default values as needed. These defaults operate only within the current submodel, and stay in effect until specifically overridden by another LU DEF subblock. Inputs given in any other LU subblock do not affect these defaults. Offsets (TLADD, PLADD) and multipliers (TLFACT, PLFACT) can be used to change units or reference temperatures and pressures. These factors apply *only during input* to subsequent TLs and PLs in the following manner:

Actual Value = (input value + offset)*multiplier

Guidance: The use of PL without an exclamation mark will erase any previous LU DEF subblock that did use PL!, meaning that PL is once again second in priority to TL. Because explicit LU subblock inputs *always* override any LU DEF subblock inputs, the explicit use of PL (without the "!") in a subsequent lump subblock will override the presence of PL! used in an earlier LU DEF subblock.

Restriction: Use of indirect referencing (such as "#this") is illegal in LU DEF subblocks.

Examples:

```
LU DEF, TL=300.0, PL=0.0, TLFACT=1.8, PLADD=14.7
LU DEF, XL=0.0, QL=100.0, CX=10.0
LU DEF,     COMP  = 1.0E-6
            XFB   = 0.0
```

3.4 Paths

Paths are subdivided into two types: *tubes* and *connectors.* Tubes are paths with significant line inertia; they are governed by a differential equation for momentum and cannot change flow rate instantaneously. Connectors are paths with insignificant inertia (flow rates may change instantaneously), and are governed by an algebraic (steady state) relationship between flow rate, pressure differential, etc. Connectors may be used to simulate pumps, valves, pressure losses, capillary passages, leaks, orifices, turbines, compressors, and short ducts. This last connector type, the STUBE or short tube, is an important element since it is identical to the tube except that it has no inertia. *Within this section, all references to tube parameters and options apply to STUBE connectors as well, unless otherwise noted.*

All paths have at least one descriptive parameter in common: a mass flow rate (FR, in units of lb_m/hr or kg/s). This descriptor must be given an initial value, and is available to users in logic blocks. As with thermodynamic properties in lumps, FR should not be directly altered within user logic blocks once the solution process is under way. It is available to the user for initialization, or as input for calculations such as heat transfer or pressure drop. The positive direction for FR is defined by specifying an upstream and downstream lump. This only determines the sign convention—flow rates may be zero or negative (e.g., reversed). For this reason, the endpoint lumps are sometimes referred to as the *defined upstream* and the *defined downstream* lumps, referring to definition of positive flow rate. As a further naming convention, the defined upstream lump is termed the i^{th} lump, and the defined downstream lump is the j^{th} lump for each path.

Optionally, paths may be assigned duplication factors, which is a general device for exploiting symmetry to reduce the size of the model. For example, an upstream duplication factor (DUPI) of 10.0 means that 10.0 such paths appear from the perspective of the defined upstream lump even though only one path exists. Discussion of these factors is deferred to Section 3.12.

Paths may also utilize phase-specific suction options, whereby the path extracts only liquid or only vapor or gas from a two-phase lump. Paths may also be directed to extract or neglect one or more species in a mixture. These options are exercised by the STAT flag in FLOW DATA and by the CHGSUC routine in logic blocks, are described below (phase suction) and in Section 3.19.4 (species suction).

Normally, paths are stationary. Alternatively, they can be assumed to be spinning or rotating, with the additional fluidic forces associated with such motion applied to the fluid within. For paths with co-rotating walls (e.g., oil within a hollow cam shaft), these additional forces are the main effect of rotating. It is also possible to specify independently rotating walls (e.g., flow in a seal between a rotor and stator), in which case (for tubes and STUBE connectors), pressure drop and heat transfer will be strongly affected. Section 3.26 describes the path rotation options, which can include hydraulic torque and dissipation and/or shaft work calculations.

By default, paths are single, homogeneous entities: vapor and liquid flow at the same velocity. Alternatively, they may be input in permanently matched pairs or *twins*. Twins are nearly identical to each other; they share the same endpoint lumps, same type, etc. However, one of the pair is intended to carry liquid when it is available, and the other is intended to carry vapor. These are called the *primary twin* and *secondary twin*, respectively.

For tubes and STUBE connectors, slip flow modeling is automatically invoked when paths are twinned. This feature is intended to be used in conjunction with the default IPDC=6 (or negative values of IPDC, as explained later), where IPDC is the two-phase frictional pressure drop correlation identifier defined later. When only single phase fluid is present in tubes or STUBE connectors, the secondary twin will disappear and only the primary path will exist. It will behave as if no twin existed. When two-phase flow reappears, the primary path will revert to carrying liquid and the secondary path will reappear, carrying vapor or gas. Refer to Section 3.17.

Any other connector device may also be twinned. Two identical paths are generated, the first with STAT initialized to DLS and the second with STAT initialized to DVS. However, unlike tubes or STUBE connectors, no other modeling logic is invoked for such twinned devices.

For two-phase flow in tubes and STUBE connectors, the phases either move together (using the default homogeneous assumption or they can slip past each other (using twinned paths). A third option is available: axially stratified flows (Section 3.27) or “flat front,” which can be used to model the purging of a liquid-filled line with gas, or the priming of a gas-filled line with liquid.

By default, the path is assumed to start at the coordinate location of the defined upstream (i^{th}) lump (as defined by its CX, CY, and CZ positions), and terminate at the coordinate location of the defined downstream (j^{th}) lump. These locations are used for body force terms and flow regime determination. However, when modeling large two-phase vessels (e.g., a fuel tank, boiler, dewar, or a liquid reservoir) in a gravity field or with vehicle accelerations, the lump at the end of the path is no longer tightly associated with the path: its void fraction or density may no longer be representative of what is flowing through the path. In such cases, the changes associated with one of the path being either submerged (wet) or exposed to ullage (dry) need to be taken into account. Refer to Section 3.13.4 for more options for this type of application.

3.4.1 Tubes

Tubes are models of fluid lines with nonnegligible inertia. This means that *the flow rate inside a tube cannot change instantaneously*—it takes time for the forces acting on the tube (such as pressures at the ends) to accelerate or decelerate the fluid inside the tube. Thus, tubes are time-dependent and the size of the tube (measured by the fluid density times the length to diameter ratio, $\rho L/D$) might affect the allowable time step during a transient. Liquid lines have more inertia than vapor or gas lines because of the density difference. Other approximations are available for small $\rho L/D$ ducts, as discussed below.

Recall that tubes, in spite of being defined by lengths and diameters, do not contain any *mass* nor can heat be transferred through the walls directly. Any mass associated with a tube should be attributed to the lumps on either end of it, and any energy must similarly be added or subtracted

through those lumps. This does not imply that tubes do not have *inertia*, or that the heat transfer calculations neglect tube shape (see Section 3.6 and Appendix B).

There are many descriptive inputs available for tubes, and most may be altered within user logic blocks. The first-time user should not be concerned with the number or apparent complexity of these parameters. Most have convenient defaults, or are calculated automatically. Thus, the rule of thumb for tubes is: *If you don't understand it, ignore it. It will probably go away.* However, experienced users will find that access to so many parameters will stretch the intended usage of FLUINT to many analysis needs.

Table 3-3 summarizes the tube descriptors. They are described in more detail in Subsection 3.4.1.2. Additional tube descriptors pertaining to the modeling of dissolved substances are described in Section 3.23.

The governing differential equation for tubes is:*

$$\frac{dFR_k}{dt} = \frac{AF_k}{TLEN_k}\left(PL_{up} - PL_{down} + HC_k + FC_k \bullet FR_k \bullet |FR|_k^{FPOW_k} + AC_k \bullet FR_k^2 - \frac{FK_k \bullet FR_k \bullet |FR|_k}{2 \bullet \rho_{up} \bullet AF_k^2} \right)$$

This equation is simply Newton's second law relating force and acceleration (F=m•a). The subscript k refers to the current (k^{th}) tube. This equation is presented as an aid in the understanding of certain tube options. It is not necessary to understand this equation to use tubes effectively.

3.4.1.1 Twinned Tubes: Slip Flow Simulation

Tubes may optionally be twinned, or input in matched pairs, to simulate slip flow. The primary twin carries liquid when liquid is present upstream, and the secondary twin (i.e., the one named after the TWIN= keyword) carries vapor or gas. In the limits of single phase flow, the secondary tube is ignored and the primary tube carries all the flow, whether liquid or vapor. This option is intended to be used in conjunction with the flow regime mapping option described below, which is invoked by specifying IPDC=6, -1, -2, -3, -4, or -5. Refer to Section 3.16 for more in-depth discussion of flow regime mapping and slip flow simulation.

* These equations omit body force and rotation terms for simplicity.

Table 3-3 Tube (and STUBE Connector) Descriptive Parameters

Param.	Req'd* Input?	Avail.**? (FLOGIC)	Can Alter?	Automatic Calculation?	Description
FR	Yes	Yes	No	Yes (solved for)	Mass flow rate
STAT	No	No	Yes***	If twinned	Suction status
MCH, HCH	No	Yes	Yes	No	Choked flow method, hysteresis
TLEN	Yes	Yes	Yes	No	Tube length
DH	Yes	Yes	Yes	No	Hydraulic diameter
DEFF	No	Yes	Yes	No	Effective diameter
AF	No	Yes	Yes	If AFI/AFJinput	Flow area
AFTH	No	Yes	Yes	No	Throat flow area
AFI, AFJ	No	Yes	Yes	No	Inlet and outlet flow areas
IPDC	No	Yes	Yes	No	Frictional pressure drop correlation
UPF	No	Yes	Yes	No	Upstream fraction of properties
WRF	No	Yes	Yes	No	Wall roughness fraction
FK	No	Yes	Yes	No	Additional K-factor losses
CURV	No	Yes	Yes	No	Radius of curvature (coils, etc.)
HC	No	Yes	Yes	No	Head coefficient
AC	No	Yes	Yes	If in macro or AFI/AFJ	Recoverable loss coefficient
FC	No	Yes	Yes	If nonzero IPDC	Irrecoverable loss coefficient
FPOW	No	Yes	Yes	If nonzero IPDC	Flow rate exponent in FC term
FD	No	Yes	Yes	If twinned, IPDC=6	Interface drag coefficient
FG	No	Yes	Yes	If in macro, twinned	Phase generation coefficient
AM	No	Yes	Yes	If twinned	Added mass coefficient
FCLM	No	Yes	Yes	No	Laminar friction multiplier
FCTM	No	Yes	Yes	No	Turbulent friction multiplier
ROTR	No	Yes	Yes	No	RPM for spinning paths
RADI, RADJ	No	Yes	Yes	No	Starting/ending radius for spinning paths
VAI, VAJ	No	Yes	Yes	No	Starting/ending angle for spinning paths
VXI, VXJ	No	Yes	Yes	No	Starting/ending angle to rotation axis
RVR	No	Yes	Yes	No	Rotational velocity ratio
REY	No	Yes	No	Yes	Reynolds number
REW	No	Yes	No	Yes	Rotational Reynolds number
REX	No	Yes	No	Yes	Effective/total Reynolds number
EFFP	No	Yes	Yes	Yes	Pumping (rotation) efficiency
TORQ	No	Yes	Yes	Yes	Hydraulic torque
QTMK	No	Yes	Yes	Yes	Power or dissipation (pump, rotate)
CXI,CYI,CZI	No	Yes	Yes[4]	No	Inlet coordinate location ("port")
CXJ,CYJ,CZJ	No	Yes	Yes[4]	No	Outlet coordinate location ("port")
BFI, BFJ	No	Yes	No	Yes	Output depth of each "port"
XMA	No	Yes	No	Yes	Mach number
MREG	No	Yes	No	Yes	Flow regime identifier
PLTH	No	Yes	No	Yes	Throat pressure (if choked)
TLTH	No	Yes	No	Yes	Throat temperature (if choked)
XLTH	No	Yes	No	Yes	Throat quality (if choked)
FRC	No	Yes	No	Yes	Critical flow rate

* Default can be defined earlier (see PA DEF subblocks)

** In general, refer only to primary twin in logic or expressions; secondary twin will assume that value if changed

*** Use CHGSUC routine

4 Only if these have first been defined in FLOW DATA, else changes are ignored.

The governing differential equation for each member of a pair of twinned tubes is:

$$\frac{dFR_k}{dt} = \frac{AF_k}{TLEN_k}\left(\alpha_k (PL_{up} - PL_{down} + HC_k) + \frac{1}{\alpha_k}\left(FC_k \cdot FR_k \cdot |FR|_k^{FPOW_k} + AC_k \cdot FR_k^2 - FK_k \cdot FR_k \cdot |FR|_k / (2\rho_k \cdot AF_k^2)\right)\right) +$$

$$\frac{1}{\alpha_k} FD_k \cdot FR_k - \frac{1}{\alpha_t} FD_t \cdot FR_t + \varepsilon_k \left(\frac{1}{\alpha_k} FG_k \cdot FR_k + \frac{1}{\alpha_t} FG_t \cdot FR_t\right) +$$

$$\alpha_t \cdot AM_t \cdot \frac{dFR_t}{dt} - \alpha_k \cdot AM_k \cdot \frac{dFR_k}{dt}$$

Again, this equation is derived from Newton's F=m·a. The suffix t refers to the twin of the current (k^{th}) tube; the t^{th} tube follows the same equation except that the k and t suffixes are reversed. Note that AF_k=AF_t, $TLEN_k$=$TLEN_t$, and FK_k=FK_t, but that other parameters in the above equation are not shared.

The areal fraction of the k^{th} tube is α_k, where this fraction is defined as the ratio of the phasic cross sectional area to total cross sectional area. If k is the secondary (vapor) tube, α_k is equal to the void fraction although the above definition is intended to cover the parallel liquid volume fraction as well. Also, just as $\alpha_v = 1 - \alpha_l$, then by definition $\alpha_k = 1 - \alpha_t$. The above equation was derived from the simple homogeneous tube equation by multiplying all pressure (driving) terms by α, and by dividing all velocity (response) terms by α.

The term ε_k, whose value is either +1 or -1, is defined below in the paragraph describing FG.

3.4.1.2 Tube (and STUBE Connector) Input Parameters

The following paragraphs describe the parameters that comprise the tube governing equations. STUBE connectors, which will be described later, use a slightly modified version of this equation with the same parameters. Therefore, the following discussion is applicable to both tubes and STUBE connectors, except as noted.

Figure 3-3, presented in a later section, describes the sequence used to update these parameters depending on the solution routine and the options selected.

Note that while some of the following parameters are input by the user, *most are calculated by the program by default* and can be safely ignored by novice or casual users. Such users should become familiar with FR, TLEN, DH, AF, WRF, and FK only.

Additional path descriptors pertaining to modeling of rotating are described in Section 3.26, and those pertaining to dissolved substances are described in Section 3.23.

FR: Mass flow rate—The initial value of FR should be chosen carefully (unless STEADY is called) since it will affect solution speed. Zero flow rates should only be input where they are expected. Small flow rates adversely affect time steps because the flow rate changes per time step are often restricted to a percentage of the total current flow rate. Once initialized (in FLOW DATA or in OPERATIONS DATA), the FR should not be changed directly. (Indirect changes may be made via other parameters, such as the SMFR factor for MFRSET connectors, which are described in later sections.)

STAT: Liquid/vapor and constituent suction status—The assumption of homogeneous phase distribution in two-phase lumps may be avoided using this option, as can the assumption of homogeneous distribution of species in mixtures. STAT can only be altered during execution using the CHGSUC routine. The use of the STAT option is not restricted to tubes. With a few exceptions, it can be applied to any path.

The default value is STAT=NORM. Under this condition, a tube will extract both liquid and vapor from a two-phase upstream lump, and will extract the homogeneous mixture of constituents from the upstream lump.

STAT can be set such that the tube will extract only liquid or only vapor or gas from such a lump. This is useful in the simulation of stratified tanks, liquid/vapor separators, and capillary devices. The following list details the results caused by different STAT values:

NORM . . . Suction of both phases (default)
LS. . . . . Liquid-only suction from defined upstream end
VS . . . . . Vapor-only (or gas-only) suction from defined upstream end
RLS . . . . . Liquid-only suction from defined downstream end
RVS . . . . . Vapor-only suction from defined downstream end
DLS . . . . . Liquid-only suction from either end
DVS . . . . . Vapor-only suction from either end
LRV . . . . . Liquid-only from the defined upstream, vapor-only from the downstream
VRL . . . . . Vapor-only from the defined upstream, liquid-only from the downstream

An important restriction applies to STAT operation: liquid-only or vapor-only suction only applies when the upstream lump is in a two-phase state. Otherwise, the tube extracts whatever phase is available even if that phase is not the one specified in the STAT option. Phase-specific suction from junctions (or tanks in STEADY) can produce nonintuitive results, as described in later sections (Section 3.11.1).

The STAT option is manipulated internally by twinned paths, and should not be used in combination with them unless they have been forced to stay homogeneous using the NOSLIP routine (Section 7.11.1.3).

The STAT option may also be manipulated internally if path end locations (“ports”) are used, as described in Section 3.13.4.

Additional STAT options can be used to extract or avoid specific species from upstream tanks, as needed to model nonhomogeneous distributions, mass transfer effects, and even chemical reactions. These species-specific options work with the above phase-suction options, and are described

separately in Section 3.19.4.

MCH and HCH: Choked flow detection and modeling—See Section 3.18.

TLEN: Tube length—The tube length is a required input that is available in the user logic blocks and may be altered if necessary. It should be noted that the "size" of the tube is directly proportional to TLEN. This is important information to keep in mind when developing a fluid model, since tube size can directly affect the allowable transient time step. TLEN is also used to determine frictional pressure drops if IPDC is nonzero, and may also be used in other calculations such as heat transfer.

DH: Hydraulic diameter—The hydraulic diameter is a required input that is available in the user logic blocks and may be altered if necessary. The hydraulic diameter is defined as four times the flow area divided by the wetted perimeter. This reduces to the diameter for a circular cross section, and to the length of one side for a square cross section. DH is used in calculations such as heat transfer. DEFF (below) defaults to DH, and DEFF is used to determine frictional pressure drops if IPDC is nonzero. If AFI and AFJ are input and no value of DH is input, DH and DEFF are initialized as the diameter at the average flow area assuming a circular cross section: $(2*(AFI+AFJ)/\pi)^{1/2}$.

DEFF: Effective diameter—In most cases, the effective diameter DEFF will be the same as the hydraulic diameter DH. The friction factor calculated using the hydraulic diameter assumption is accurate to within about 15% for turbulent flow, but can be off by as much as 40% for laminar flow in extreme cross sections. A corrected or effective hydraulic diameter, DEFF, may therefore also be input for tubes and STUBE connectors. DEFF defaults to DH.

For example, for a square duct, DEFF=(64/57)*DH (where DH is the length of a side). For parallel plates, where DH=2G where G is the gap between the plates, DEFF = 2*DH/3 = 4*G/3. Such factors are based on exact laminar flow solutions, as provided in references such as Berker.*

DEFF is supplied *in addition* to DH; DEFF does not replace DH in all calculations, and therefore DH is still a required and independent variable. DEFF becomes the basis of the Reynolds number for both laminar and turbulent flow frictional calculations. It should also be the basis of the WRF roughness factor supplied independently by the user: WRF = ε/DEFF. However, DH is still used as part of the calculation of the pressure drop. DEFF is used to calculate the friction factor for the shear stress at the wall, but that stress is then applied to the wetted perimeter, which is a function of DH. In other words, if single-phase pressure drop were treated as an equivalent K-factor,† $K_{equiv} = f_{Deff}*TLEN/D_h$: the friction factor (f) is a function of DEFF, but the pressure drop itself is also a function of the length (TLEN) and the hydraulic diameter (D_h).

* R. Berker, "Integration des equations du mouvement d'un fluide visqueux incompresible," Handbuch der Physik, Vol VIII/2 (1963). A useful subset of solutions is presented in F. M. White's Viscous Fluid Flow (Section 3-3.2, 1974). However, many references list laminar friction factors such as the Poiseuille number ($0.25*f_d*Re$ where f_d is the Darcy friction factor, or f_f*Re where f_f is the Fanning fraction factor). Since Po=16 for circular flow, and Po=24 for flow between parallel plates, therefore DEFF = Po_{cir}/Po_{pp} * DH = 2*DH/3. While defined for laminar flow, such DEFF values are very good estimates for turbulent flow as well (F.M. White, Fluid Mechanics, 1979, eqn. 6.98, p 345).

† This equation is an oversimplification of the actual methods used in the software.

DH is also still used for most two-phase flow calculations and almost all heat transfer calculations, including the calculation of wetted perimeter and the basis of the Nusselt number, since that is how most heat transfer correlations are referenced. To modify heat transfer calculations to account for noncircular cross-sections, use the XNUL, XNLM, and XNTM parameters for ties (Section 3.6.3.2).

AF: Flow area—If no cross-sectional flow area is given or if AF is negative, AF is calculated by assuming that the duct is circular. Note that AF is therefore only needed for noncircular sections, and that the user may return to inputting circular sections in a FLOW DATA block by setting a negative value as default. *Otherwise, AF and DH and DEFF are independent of each other.* AF is available in the user logic blocks and may be altered if necessary, but a negative AF can only be used as a signal in FLOW DATA blocks. It should be noted that AF and DH may be used to calculate the wetted perimeter for convective heat transfer. It should also be noted that the "size" of the tube is *inversely* proportional to AF. AF is also used to determine frictional pressure drops if IPDC is nonzero, and may also be used in other calculations such as kinetic energy, heat transfer, and accelerational pressure forces. If AFI and AFJ are input, AF is maintained as the average of the those two values.

AFI and AFJ: Inlet and outlet flow areas—If the cross sectional area varies axially, AFI and AFJ can be used to handle that case. Otherwise, AFI and AFJ are left as negative, signaling that AF should be used instead. AFI is the flow area at the defined upstream end (i^{th} end) of the path, and AFJ is the flow area at the defined downstream end (j^{th} end). These definitions do not change if the flow rate happens to reverse during a solution.

If both AFI and AFJ are set, then AF becomes the average value and AC is automatically calculated to model the accelerational effects (Section 3.15) of both area and density changes. To model density change effects in ducts with constant cross section, AFI and AFJ may be equal. Users are strongly cautioned to add loss factors (e.g., FK) when areas change, especially when the flow area expands in the flow-wise direction. The AC factor does not include such irrecoverable losses, and excluding them can result in incorrect answers and instabilities. Consider alternatively a REDUCER or EXPANDER connector in series with the tube.

AFTH: Throat flow area—Flow area for any internal restrictions or contractions that may limit flow rates due to choking (flow cannot exceed sonic velocities in such throats). If no throat flow area is given or if AFTH is nonpositive, AFTH is assumed equal to AF unless AFI and AFJ are input, and then it is assumed equal to the smaller of those two values. *Otherwise, AFTH independent of AF, AFI, and AFJ.* AFTH is available in the user logic blocks and may be altered if necessary. Currently, AFTH is *only* used in the choked flow detection and simulation options described separately in Section 3.18.

IPDC: Frictional pressure drop correlation number—While there are standard frictional pressure drop models for single-phase flow, there is no such agreement when it comes to two-phase flow. This flag specifies whether or not FLUINT should automatically perform frictional pressure drop calculations (for FC and FPOW described below), and which correlation should be used for two-phase flow. Setting IPDC=0 means FC and FPOW will not be updated for this tube. Any other value

means FC and FPOW will be updated by the standard methods for single-phase flow, and the value of IPDC determines which of several popular two-phase correlations should be used. See Appendix A for further description of these friction models.

The following list details the action taken according to the value of IPDC:

0 . . . . . No FC or FPOW calculations for this tube (user input)
1 . . . . . Standard single-phase model, plus McAdam's Homogeneous for two-phase
2 . . . . . Standard single-phase model, plus Lockhart-Martinelli's for two-phase
3 . . . . . Standard single-phase model, plus Baroczy's for two-phase
4 . . . . . Standard single-phase model, plus Friedel's for two-phase
5 . . . . . Standard single-phase model, plus Whalley's recommended choice of 2, 3, or 4 (above) for two-phase
6 . . . . . (**Default.**) Standard single-phase model, plus flow regime determination and frictional models based on those simplified regimes. Also invokes calculation of FD parameter when used in conjunction with twinned tubes. Flow regime information is also used in two-phase multiple-constituent and nonequilibrium flows to help predict heat and transfer coefficients.

As described in Section 3.16, the user may force the selection of a particular regime by using negative values of IPDC.

Note that IPDC=6 is the default* since flow regime information is required for most mixture calculation (especially heat transfer and dissolution/evolution) as well as nonequilibrium modeling. This default should be used unless the user has knowledge of two-phase regimes (which can be specified using negative values of IPDC as described in Section 3.16). Otherwise, without regime information the user must have advanced knowledge of wall friction apportionment, interface drag correlations, wall heat transfer apportionment, interfacial surface areas, interfacial heat and mass transfer coefficients, etc.

UPF: Upstream fraction of properties—This parameter is used if IPDC is not zero. This parameter allows the user to calculate FC and FPOW according to modeling preferences. If UPF=1.0, only the properties of the lump *currently* upstream of this tube will be used for the friction calculations. If UPF=0.0, only the downstream lump properties will be used. If UPF=0.5 (the default), both will be used for an average of the two. Any other UPF (between 0.0 and 1.0 only) will result in a weighted average.

Note that UPF does not affect the fluid density used in the FK loss term. These K-factor losses are always evaluated using the current upstream density.

The UPF value may also be manipulated internally if path end locations ("ports") are used, as described in Section 3.13.4.

Caution: Use of ***the default value of UPF=0.5 may cause instabilities and overestimated pressure drops*** in compressible flows that use excessively long tubes or STUBE connectors (i.e., inadequate resolution). Use a larger value of UPF (perhaps 1.0) or more resolution to avoid such

* Prior to Version 4.2, IPDC=1 was the default for single paths, and IPDC=6 for twinned paths.

problems. FLUINT will attempt to autodetect such instabilities ("UPF RUNAWAY ..."), overriding what is presumed to be a default of UPF=0.5 with the more stable UPF=1.0. In such cases, the user should review results to see if other tubes or STUBE connectors require UPF=1.0, and also whether resolution (number of elements) should be increased so as not to underestimate frictional pressure drop using UPF=1.0.

WRF: Wall roughness fraction—The roughness fraction is defined as the ratio of the characteristic length of the wall roughness to DEFF (usually the same as DH), and is usually on the order of 1.0^{-6} to 1.0^{-3}. If no WRF is given, it is assumed to be zero (smooth wall—this does not mean frictionless). WRF is available in the user logic blocks and may be altered if necessary. WRF is only relevant if IPDC is nonzero.

FK: Additional K-Factor head losses—Effects of pressure head losses due to entrances* and exits† as well as fittings such as bends, tees,‡ expansions,** contractions, orifices, valves, etc. may be included as K-factor head losses. Under steady flows, the irrecoverable pressure loss due to these effects is $\Delta P = FK \bullet FR^2/(2 \bullet \rho \bullet AF^2)$. Most undergraduate fluid mechanics texts contain tables and equations of appropriate values. Otherwise, *Flow of Fluids through Valves, Fittings, and Pipe* (Crane Technical Paper 410) is a definitive source.†† Note fully turbulent flow is often implicitly assumed in such tables and equations. FK is normally positive, although negative values can be used with caution. Caution: FK is based on the upstream density and on the average AF (if AFI and AFJ are used).

HC: Head coefficient—The head coefficient may be thought of as an additional pressure force superimposed on the tube. This may be caused by body forces or even by a pump contained within the tube (for a combined model). This parameter is left strictly to the user to specify and alter as needed. Otherwise, the value is assumed zero (no additional forces). The units of HC are pressure.

AC: Recoverable loss coefficient—A recoverable loss is essentially a pressure drop that becomes a pressure gain if the flow is reversed. The default is zero. This is most often caused by area and/or density changes between the inlet and the outlet of the tube, but may also be caused by radial addition or extraction (blowing or suction) of fluid. This parameter, which is automatically updated if AFI and AFJ are input, or if the tube is within a duct macro, is described in detail in Section 3.15.

---

* Because FLUINT uses static pressures, and entrance should have an FK=1.0 (if not already included in the AC factor) to account for acceleration from stagnation. ***This accelerational loss is automatically applied if the upstream lump uses LSTAT=STAG.***

† Because FLUINT uses static pressures by default, no exit loss is required - see also above footnote regarding entrances.

‡ When multiple mass flow rates and flow areas are involved such as in wyes and tees, extra conversions may be necessary (see YTKONV in Section 7). See also YTKALI and YTKALG for correlations.

** See also the AC factor and Section 3.15.

†† Other recommendations:
D.S. Miller, *Internal Flow Systems*, ISBN 0-87201-020-1 (2nd edition, 1990)
I.E. Idelchik, *Handbook of Hydraulic Resistance*, ISBN 1-56700-074-6 (3rd edition, 1993)

FC and FPOW: Irrecoverable loss coefficient and corresponding flow rate exponent—Unlike a recoverable loss, an irrecoverable loss remains a loss even when the flow is reversed. This is most often caused by wall friction, although bends, tees, and pipe fittings can contribute. When IPDC is nonzero, 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).* With FPOW equal to 1.0, FC is proportional to a K-factor loss: $FC = -K_{eq}/(2\rho AF^2)$, where K_{eq} is the equivalent K-factor, ρ is the fluid density, and AF is the flow area as previously defined. If FPOW is 0.0, the resistance is linearly proportional to the flow rate. Note FC is typically a large negative number. *The units of FC depend on the value of FPOW.*

The user has several options when dealing with FC and FPOW. The user may (1) ignore them and let FLUINT calculate them according to default options, which is by far the most common approach, (2) specify the types of calculations to be performed (see below), (3) let the program calculate FC and FPOW and then modify or replace them, or (4) suppress FC and FPOW calculations altogether. When frictional pressure drops are desired, the user has many options available for calculating FC and FPOW. These options are controlled by parameters IPDC, UPF, and WRF, and are described below. It should be noted that setting FC to zero is valid, and corresponds to a frictionless pipe. However, *all of FK, FC, and AC cannot be zero for a steady-state solution, or with STUBE connectors*.

FCLM and FCTM: Friction Multipliers—Two frictional pressure drop multipliers are available: FCLM for laminar flow, and FCTM for turbulent flow. FCLM and FCTM affect the calculated FC (irrecoverable loss coefficient) values. These factors are defaulted to unity.

These single-phase factors might be used to adjust pressure drops as needed for noncircular cross-sections (in which case DEFF should be defaulted to DH), but more commonly are used for including strong property variations (in heated or cooled ducts), modeling helical wires or ribs used for heat transfer enhancements (see HTCENH, Section 7.11.4.7), modeling journal bearing losses, and other special circumstances.

FD: Interface drag coefficient (Twinned tubes and STUBE connectors)—The interface drag term describing the retarding force exhibited by each phase on the other, in proportion to the difference in velocities. When IPDC=6, FD is calculated for each phase as $FD_k = -f_{vl}|V_v-V_l|/\rho_k$, where f_{vl} is the interface drag coefficient, V_v-V_l is the current difference between the velocities of each phase, and ρ_k is the density of the phase of the current tube. The interface drag coefficient is determined as a function of flow regime and void fraction, among other factors. Like FC, FD is typically a large negative number. If FD equals $-\infty$, the flow will be homogeneous since any difference in velocity will be quickly extinguished. Along these lines, |FD| is typically larger for dispersed flow regimes (bubbly and slug) than for separated regimes (annular and stratified), resulting in smaller slip ratios (ratio of vapor to liquid velocity) for those regimes.

* The path output parameter REY can be used to check a path's upstream Reynolds number (before any area or phase changes occur within that path). For curved or coiled tubes (small CURV values), bear in mind that the critical Reynolds number is higher than the traditional value of 2000.

FG: Phase generation coefficient (Twinned tubes and STUBE connectors)—FG is the term covering the momentum transfer associated with phase change: when one phase is converted to another it must be accelerated or decelerated to compensate, and any velocity change must be caused by a force on the fluid. FG is zero in multiple-constituent two-phase flows, since phase change is not currently modeled in mixtures.

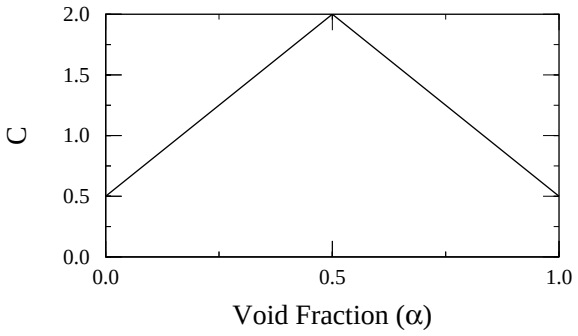
FG is calculated for each phase as $FG_k = \Gamma_t/\rho_k$, where Γ_t is the mass generation rate per unit volume of the twin phase, and ρ_k is the density of the phase of the current tube. (Since $-\Gamma_k=\Gamma_t$, it can also be defined as $FG_k = -\Gamma_k/\rho_k$.) The larger the rate of phase conversion, the larger |FG| becomes, although this is usually a small term.

Its relative insignificance is fortunate because of an inherent uncertainty in its formulation: to truly capture this term an interface velocity must be estimated, where the interface velocity is somewhere between the liquid and vapor velocities. In FLUINT, like several other similar codes, a donor formulation is used: the interface velocity is assumed to be the same as that of the donor phase. (While an average of the two velocities results in reversibility and therefore represents a physical limit, the donor formulation errs on the side of irreversibility.) The term e_k governs this donor formulation. For evaporation, e_k is zero for the liquid equation and unity for the vapor equation, in which case the liquid FG is positive and the vapor FG is negative. In other words, the FG term acts like a retarding force on vapor during evaporation, and an accelerating force on liquid during condensation.

This parameter is normally defaulted to zero. However, it is automatically updated for tubes and STUBE connectors within duct macros—see also Section 3.9.2.

AM: Added (virtual) mass coefficient (twinned tubes, not applicable to STUBE connectors)—Because of the typically large difference in density between the liquid and vapor phase, by itself vapor has negligible inertia compared to that of liquid. However, each phase is not alone: because the acceleration of one phase involves the displacement of another, the distinction is not as clear cut. Consider a vapor bubble rising within a pool of liquid: after detachment from the bottom of the pool, the bubble does not accelerate instantly up to ‘terminal velocity’ because in order to do so it must displace an equal volume of liquid, which has arguable inertia. The added mass term (also called virtual mass) can be thought of as a correction term that allows each phase to ‘share’ the inertia of the liquid phase, tending to reduce |dFR/dt| for both phases.

AM is calculated as $AM_k = C\rho/\rho_k$, where ρ is the mixture density and ρ_k is the density of the phase of the current tube. C is a unitless constant whose exact value is subject to debate. For separated regimes, C approaches zero since the phase can accelerate somewhat independently. For dispersed regimes, values of approximately 0.5 have been recommend-



ed. Because larger values of C generally increase stability, FLUINT errs on the side of larger C and does not use flow regimes in its estimation of C. Rather, C is estimated as a ramp function of void fraction, as shown above at the right.

Although AM is calculated automatically, it can be overwritten in FLOGIC 1.

CURV: Radius of Curvature—Curved passages (helically coiled tubes, spirals, etc.) result in secondary flows that increase pressure drop. The radius of curvature, CURV, may therefore be specified for any tube or STUBE connector. The default for CURV is 1.0E30 (essentially infinite), meaning a straight duct. When a finite value of CURV is supplied, internal correlations* for laminar and turbulent friction factors are applied, as well as correlations for critical Reynolds number (the tighter the turn, the longer transition to turbulent is delayed).

While primarily a single-phase adjustment, finite values of CURV can also indirectly affect the regime selection and void fraction prediction in two-phase flows due to centripetal accelerations.

Note that specification of finite CURV values does *not* affect tie heat transfer coefficients directly. Refer to the HTCURV routine (Section 7.11.4.5) for such independent adjustments.

Note also that CURV is *not* appropriate for pipe bends (e.g., 45 degree or 90 degree), since it assumes fully developed secondary flows.

"FTIE = AXIAL" directive—To include heat conduction between the endpoint lumps on the basis of the tube or STUBE connector, a parallel AXIAL ftie (Section 3.7) can be easily generated by specifying "FTIE=AXIAL" within the path subblock. The generated ftie will have the same identifier as the path, and will use the same endpoint lumps. It will also use the same duplication factors. If the endpoint lumps are twinned as so is the path, a pair of twinned fties will automatically be generated using this directive.

3.4.1.3 Tube Subblock Format

```
PA TUBE,id,l1,l2[,FR=R][,MCH=I],HCH=R][,TWIN=I][,STAT=C]
        [,TLEN=R][,DH=R][,AF=R][,AFTH=R][,AFI=R,AFJ=R]
        [,FK=R][,WRF=R][,IPDC=I][,UPF=R][,FTIE=AXIAL]
        [,DEFF=R][,CURV=R][,FC=R][,FCLM=R][,FCTM=R]
        [,HC=R][,FPOW=R][,AC=R][,AM=R][,FG=R][,FD=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]
        [,DUPI=R][,DUPJ=R][,DUP=R][,FF=C]
```

* E.C. Guyer et al, Handbook of Applied Thermal Design, Chapter 4.7, McGraw-Hill, 1989.

where:

`id` . . . . . tube identifier
`l1,l2` . . . . . upstream and downstream lump identifiers
`FR` . . . . . initial mass flow rate
`MCH` . . . . . choked flow detection and modeling method flag (see Section 3.18)
`HCH` . . . . . hysteresis fraction to apply to critical flow rate (see Section 3.18)
`TWIN` . . . . . unique identifier of secondary twin tube to be generated (for slip flow and nonequilibrium modeling), where id is the primary twin
`STAT` . . . . . phase suction status flag:
NORM . . . .normal (extracts both phases from upstream lump)
LS . . . . . .Liquid suction from l1 only
VS . . . . . .Vapor suction from l1 only
DLS . . . . . .Liquid suction from l1 *or* l2
DVS . . . . . .Vapor suction from l1 *or* l2
RLS . . . . . .Liquid suction from l2 only
RVS . . . . . .Vapor suction from l2 only
LRV . . . . . .Liquid suction from l1, vapor suction from l2
VRL . . . . . .Vapor suction from l1, liquid suction from l2
See also Section 3.19.4 for species-specific suction options.
`TLEN` . . . . . tube length
`DH` . . . . . hydraulic diameter (if twinned, applies to whole passage)
`DEFF` . . . . . Effective diameter, for cases where DH is inappropriate
`AF` . . . . . flow area (if twinned, applies to whole passage)
`AFTH` . . . . . throat flow area for choking calculations
`AFI,AFJ` . . . . flow area at the defined inlet and outlet, respectively (if one is input, they both must be). Consider also using a REDUCER or EXPANDER in series, leaving the tube with a constant cross section.
`FK` . . . . . K-factor due to associated bend losses, contained valves or orifices, etc.
`WRF` . . . . . wall roughness fraction

IPDC . . . . . Two-phase frictional pressure drop correlation number:
- 0 . . . . . User-input FC and FPOW even for single-phase flow
- 1 . . . . . McAdam's Homogeneous
- 2 . . . . . Lockhart-Martinelli
- 3 . . . . . Baroczy
- 4 . . . . . Friedel
- 5 . . . . . Whalley's recommendations
- 6 . . . . . (**Default.**[*]) Flow regime based two-phase friction, or:
  - -1. . . . . use bubbly regime
  - -2. . . . . use slug regime
  - -3. . . . . use annular regime
  - -4. . . . . use stratified regime
  - -5. . . . . use axially stratified regime (Section 3.27)

UPF . . . . . (if IPDC isn't 0) upstream fraction of properties to use for friction

FTIE=AXIAL
Generates an AXIAL ftie (Section 3.7) with the same ID and endpoint lumps as the tube for convenient inclusion of axial conduction along the tube. If either of the endpoint lumps are twinned and the tube itself is also twinned, then a pair of twinned AXIAL fties are automatically generated.

CURV . . . . . Radius of curvature for coiled tubes

HC. . . . . initial head coefficient (impressed pressure gradient)

FC. . . . . initial irrecoverable loss coefficient

FCLM . . . . . FC multiplier for laminar flow

FCTM . . . . . FC multiplier for turbulent flow

FPOW . . . . . flow rate exponent in frictional pressure drop term: $FC \bullet FR \bullet |FR|^{FPOW}$

AC. . . . . initial recoverable loss coefficient

AM. . . . . initial added (virtual) mass coefficient, for twinned tubes

FG. . . . . initial phase generation coefficient, for twinned tubes

FD. . . . . initial interface friction coefficient, for twinned tubes

ROTR . . . . . Spin or rotation rate in revolutions per minute (rpm). See Section 3.26.

RADI . . . . . Radius from center of spin at defined inlet. See Section 3.26.

RADJ . . . . . Radius from center of spin at defined outlet. See Section 3.26.

VAI . . . . . Angle between flow and tangent at defined inlet. See Section 3.26.

VAJ . . . . . Angle between flow and tangent at defined outlet. See Section 3.26.

VXI . . . . . Angle between flow-tangent plane and axis at defined inlet. See Section 3.26.

VXJ . . . . . Angle between flow-tangent plane and axis at outlet. See Section 3.26.

RVR . . . . . Rotational velocity ratio (unity for co-rotating walls). See Section 3.26.

EFFP . . . . . Rotational "pumping" efficiency. See Section 3.26.

TCF . . . . . Torque conversion factor. See Section 3.26.

INTORQ . . . . Flag: torque as an input or output. See Section 3.26.

TORQ . . . . . Hydraulic torque (if input). See Section 3.26.

---

* Prior to Version 4.2, IPDC=1 was the default for single paths, and IPDC=6 for twinned paths.

CXI.....X coordinate of inlet (if not the same as that of the upstream lump, see Section 3.13.4)
CYI.....Y coordinate of inlet (if not the same as that of the upstream lump)
CZI.....Z coordinate of inlet (if not the same as that of the upstream lump)
CXJ.....X coordinate of outlet (if not the same as that of the downstream lump)
CYJ.....Y coordinate of outlet (if not the same as that of the downstream lump)
CZJ.....Z coordinate of outlet (if not the same as that of the downstream lump)
DUPI.....Upstream duplication factor (number of this path that appear to l1)
DUPJ.....Downstream duplication factor (number of this path that appear to l2)
DUP.....Duplication factor (number of this path that appear to *both* l1 and l2; sets both DUPI and DUPJ in preprocessor only)
FF .....FF=FILL or FF=PURGE sets appropriate values of STAT, UPF, and IPDC as described in Section 3.27 for flat-front two-phase modeling

defaults (if not previously defined in a PA DEF subblock):

TWIN.....no twin (single homogeneous path created)
MCH.....-2: nonequilibrium phasic expansion with Bursik's metastable two-phase speed of sound (see Section 3.18). MCH=-3 is recommended instead if the fluid is condensible/volatile and if a throat contraction exists (AFTH<AF or AFI/AFJ are used, or if the upstream lump is stagnant).
HCH.....0.02 (2% hysteresis on critical flow rate, see Section 3.18)
STAT.....NORM
DH .....none, unless AFI and AFJ are input, in which case DH is initialized as the diameter of the average flow area assuming a circular cross section. If AFI and AFJ are input, PA DEF values are ignored.
DEFF.....DH
AF .....-1.0 (nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH. Otherwise, DH and AF are independent parameters.). If AFI and AFJ are input, AF is both initialized and maintained at the average: AF = 0.5*(AFI + AFJ)
AFI,AFJ....-1.0 (not used)
AFTH.....AF, or the minimum of AFI and AFJ if they are input.
FK .....zero
WRF.....zero
IPDC.....6 (flow regime mapping)
UPF.....0.5 (Average of up and downstream properties)
CURV.....1.0e30 (straight pipe)
FC .....zero (calculated automatically if IPDC isn't 0)
FCLM.....1.0
FCTM.....1.0

HC..... zero
FPOW..... 1.0 (calculated automatically if IPDC isn't 0)
AC..... zero (calculated automatically if AFI and AFJ are input)
AM..... zero (calculated automatically if twin tubes are input)
FG..... zero
FD..... zero (calculated automatically if twin tubes are input and IPDC=6)
ROTR..... zero
RADI..... zero
RADJ..... zero
VAI 90.0
VAJ 90.0
VXI ..... 90.0 (radial flow)
VXJ ..... 90.0 (radial flow)
RVR ..... 1.0 (co-rotating walls)
EFFP..... 1.0
TCF 1.0
INTORQ .... NO (TORQ will be calculated)
TORQ..... (calculated by default)
CXI,CYI,CZI . . .CX, CY, CZ of defined upstream lump (port is inactive)
CXI,CYJ,CZJ . . .CX, CY, CZ of defined downstream lump (port is inactive)
DUPI..... 1.0 (or value of DUP)
DUPJ..... 1.0 (or value of DUP)
DUP 1.0

Guidance: The order of l1,l2 indicates positive flow direction. The *defined* upstream end is often referred to as the i^{th} end, with the opposite end the j^{th} end so as to avoid confusion if flow reverses: the i^{th} and j^{th} designations do not change.

Guidance: While the parameters FC, FPOW, AC, AM, FG, and FD are available for direct input, they are normally calculated automatically by devices options and their presence can be ignored by casual users.

Guidance: If a frictional pressure drop model other than those available is desired, or if FC is to be zero (no frictional losses) or otherwise provided by user logic, set IPDC=0. If IPDC=0, then the input values of TLEN, WRF, UPF, and DH are ignored although TLEN and DH might still be used by HTN, HTNC, and HTNS ties. If IPDC=0, the user assumed control of both FC and FPOW.

Guidance: IPDC=6 (the default) is strongly recommended with twinned tubes, twinned tanks, and two-phase mixtures (especially those involving dissolution/evolution).

Guidance: If twinned, the initial flow rate is assumed to be the total flow rate. It will be apportioned to each twin on the basis of initial upstream quality.

Guidance: If twinned, all other initial values apply to both twins. If FC, FPOW, AC, AM, FG, or FD are being specified in FLOW DATA for twinned tubes, the user should consider modifying these values for each twin in OPERATIONS prior to a solution routine call since these parameter will rarely be equal for both twins. In fact, the magnitude of AM, FG, and FD should normally differ by the ratio of densities, and FG should be of opposite sign for each phase.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump. If the path is twinned, the identifier of the twin may be referred to as “#twin.” However, the “TWIN=I” declaration must be placed on an input line that is encountered prior to any reference to “#twin.”

Restriction: To refer to a twin’s ID as “#twin” inside of an expression, the “TWIN=I” must have been input on a previous line within the same subblock.

Caution: If AFI and AFJ are input to model flow area changes, the corresponding irrecoverable losses should also be represented via the FK (K-factor) term. See Section 3.15. Consider also using an EXPANDER or REDUCER (Section 3.5.3 and Section 3.5.4) in series and leaving the tube with a constant cross section.

Restriction: Phase suction options (STAT) are ignored in conjunction with twinned tubes (see also NOSLIP and GOSLIP support subroutines in Section 7.11.1.3).

Restriction: If IPDC=0, all of FK, FC, and AC cannot be zero in STEADY, and they should not all be zero indefinitely in other solution routines.

Examples:

```
PA TUBE,10,1,2, FR=0.03, FK=2.0, IPDC=4, UPF=1.0, DUP=500.0
PA TUBE,13,100,200, DH=1.0/12.0, AF=1.0, WRF=0.00003/DH#THIS
      FTIE  = AXIAL     $ adds axial conduction
PA TUBE,203,2,3,TWIN = 1203   $ GENERATES TWO TUBES
PA TUBE,9,991,992, HC = 100.0-PL#down$ #down = 992
```

3.4.2 Connectors

Connectors are time-independent paths through which lumps exchange mass and energy. However, unlike tubes, there is no single interpretation for a connector—there is no real fluid component that resembles a "connector." Connectors, however, may be used to simulate a wide variety of fluid components such as pumps, turbines, compressors, reducers and expanders, valves, bends, orifices, short ducts, pressure regulators, and even certain capillary devices. For these reasons, it is convenient to think of connectors in terms of the devices they model. FLUINT offers a variety of prepackaged models to simulate a number of devices automatically. Section 3.5 describes these device options in detail.

Like junctions, connectors are governed by an algebraic, time-independent equation. They therefore do not *directly* affect the allowable time step during transients. The governing equation for the k^{th} connector, for which the positive flow rate direction is from lump i to j, is:

$$\Delta FR_k^{n+1} = GK_k^n(\Delta PL_i^{n+1} - \Delta PL_j^{n+1}) + HK_k^n + EI_k^n \cdot \Delta HL_i^{n+1} + EJ_k^n \cdot \Delta HL_J^{n+1} + DK_k^n \cdot \Delta FR_t^{n+1}$$

The connector equation applies during every solution time step or steady state relaxation of the network. The superscripts refer to the iteration or time interval; n is the value at the beginning of the solution, and all values superscripted n+1 are solved implicitly during the integration or relaxation.

The parameters GK, HK, EI, EJ, and DK, along with a mass flow rate (FR), are all that any two connectors have in common. These parameters are defined as follows:

GK . . . . . partial derivative of the flow rate with respect to (w.r.t.) pressure drop changes
HK. . . . . flow rate offset: difference between next FR and current, where next FR is calculated assuming that the endpoint lumps do not change
EI . . . . . partial derivative of the flow rate w.r.t. upstream enthalpy changes
EJ . . . . . partial derivative of the flow rate w.r.t. downstream enthalpy changes
DK. . . . . partial derivative of the flow rate w.r.t. its twinned STUBE's flow rate (if twinned)

Unless the connector is a twinned STUBE carrying two-phase flow, EI, EJ, and DK are by default zero. Furthermore, DK is ignored for any connector unless it is twinned. Thus, the above equation often reduces to:

$$\Delta FR_k^{n+1} = GK_k^n(\Delta PL_i^{n+1} - \Delta PL_j^{n+1}) + HK_k^n$$

The user has three choices when using connectors. First and most common, the user may name a device model and thereafter treat the connector as that device and ignore GK, HK, EI, EJ, and DK, which will be handled internally. The discussion of available device models is given in the next section (Section 3.5). It should be noted here that all of these models are generalized, and that the

user has access to their individual operating parameters within his or her logic blocks and expressions. Therefore, it is unlikely that one of these models can't be made to simulate a unique device whose operation can be described with a single path.

Second, users may specify a *NULL* device, which forces them to take the responsibility of specifying and controlling GK, HK, etc. continuously. *This choice* (which is the default only because no other choice is appropriate) *should only be taken by the advanced user who cannot make a prepackaged model fit the particular device in question. Incorrect usage of GK, HK, EI, EJ, and can easily cause a fatal error or unexpected results.* However, this option is purposely available so that FLUINT can be used by advanced users with unanticipated device modeling requirements. NULL connectors are also used by several simulation subroutines such as COMPRS.

The third option, also recommended only for the advanced user, is to name a device, and then overwrite or modify GK, HK, EI, EJ, and DK in FLOGIC 1.

As general guidance, negative or zero GK should be used with caution. GK is often much less than unity in SI units, but much greater than unity in English units. HK is usually small compared to the value of FR, and is zero under steady conditions.

Actually, during the solution process the previously described tube momentum equations are converted into the same formulation as the above connector equation. Therefore, tubes also have the same parameters GK, HK, etc. However, these values are time-dependent, and are updated after FLOGIC 1 and before the integration. Therefore, while they are available for inspection in FLOGIC 2, they are normally of little use to the user.

Figure 3-3 summarizes the internal solution procedure for tubes and connectors in terms of when parameters are used, and in which logic blocks it makes sense to inspect or modify them. To use this chart, remember that tubes are treated like STUBE connectors in STEADY (aka, "FASTIC"). Connectors other than STUBEs follow similar rules—this figure can be extrapolated to them as well.

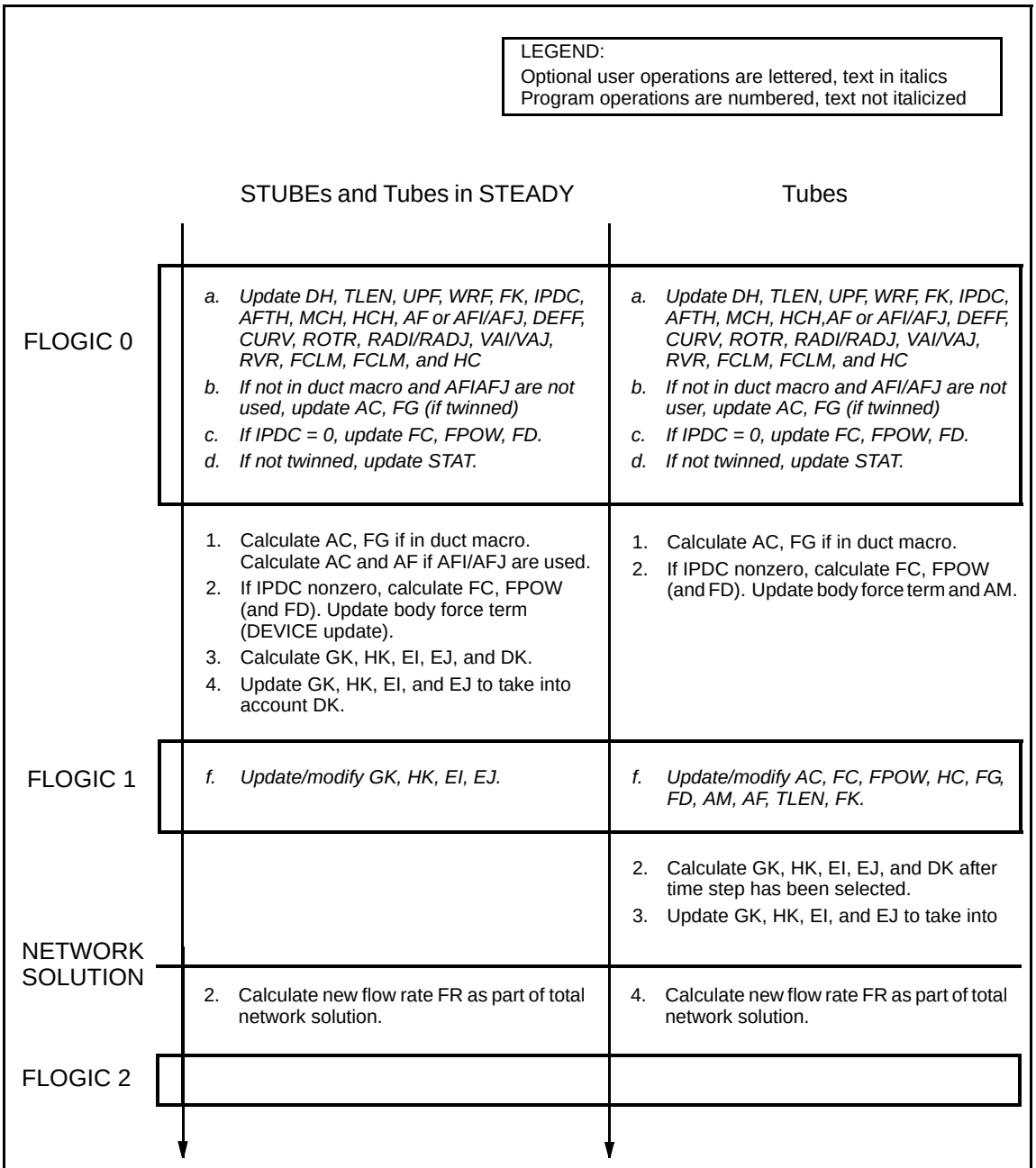


Figure 3-3 Update Sequence for Tube and STUBE Solutions

CONN Subblock Format:

```
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]
        [,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]
        [,DUPI=R][,DUPJ=R][,DUP=R][,FF=C]
        [,DEV=C,...,...]
```

where:

`id` . . . . . connector identifier

`l1,l2` . . . . . upstream and downstream lump identifiers

`FR` . . . . . initial mass flow rate

`MCH` . . . . . choked flow detection and modeling method flag (see Section 3.18)

`HCH` . . . . . hysteresis fraction to apply to critical flow rate (see Section 3.18)

`TWIN` . . . . . unique identifier of secondary twin STUBE to be generated (for slip flow and nonequilibrium modeling), where id is the primary twin.
For connectors, this option is only valid for DEV=STUBE.

`STAT` . . . . . phase suction status flag:
NORM . . . . normal (extracts both phases from upstream lump)
LS . . . . . Liquid suction from l1 only
VS . . . . . Vapor (or gas) suction from l1 only
DLS . . . . . Liquid suction from l1 *or* l2
DVS . . . . . Vapor suction from l1 *or* l2
RLS . . . . . Liquid suction from l2 only
RVS . . . . . Vapor suction from l2 only
LRV . . . . . Liquid suction from l1, vapor suction from l2
VRL . . . . . Vapor suction from l1, liquid suction from l2
See also Section 3.19.4 for species-specific suction options.

`GK` . . . . . initial derivative of flow rate with respect to pressure drop

`HK` . . . . . initial flow rate offset

`EI` . . . . . initial derivative of flow rate with respect to defined upstream enthalpy

`EJ` . . . . . initial derivative of flow rate with respect to defined upstream enthalpy

`DK` . . . . . initial derivative of flow rate with respect to twin STUBE's flow rate
For connectors, this option is only valid for DEV=STUBE.

`ROTR` . . . . . Spin or rotation rate in revolutions per minute (rpm). See Section 3.26.*

`RADI` . . . . . Radius from center of spin at defined inlet. See Section 3.26.

`RADJ` . . . . . Radius from center of spin at defined outlet. See Section 3.26.

* Rotation terms are ignored for MFRSET, VFRSET, and NULL connectors. See Section 3.26.3.3 for interactions with PUMP connectors.

VAI ..... Angle between flow and tangent at defined inlet. See Section 3.26.
VAJ ..... Angle between flow and tangent at defined outlet. See Section 3.26.
VXI ..... Angle between flow-tangent plane and axis at defined inlet. See Section 3.26.
VXJ ..... Angle between flow-tangent plane and axis at outlet. See Section 3.26.
RVR ..... Rotational velocity ratio (unity for co-rotating walls). See Section 3.26.
EFFP ..... Rotational "pumping" efficiency. See Section 3.26.
TCF ..... Torque conversion factor. See Section 3.26.
INTORQ .... Flag: torque as an input or output. See Section 3.26.
TORQ ..... Hydraulic torque (if input). See Section 3.26.
CXI ..... X coordinate of inlet (if not the same as that of the upstream lump, see Section 3.13.4)
CYI ..... Y coordinate of inlet (if not the same as that of the upstream lump)
CZI ..... Z coordinate of inlet (if not the same as that of the upstream lump)
CXJ ..... X coordinate of outlet (if not the same as that of the downstream lump)
CYJ ..... Y coordinate of outlet (if not the same as that of the downstream lump)
CZJ ..... Z coordinate of outlet (if not the same as that of the downstream lump)
DUPI ..... Upstream duplication factor (number of this path that appear to l1)
DUPJ ..... Downstream duplication factor (number of this path that appear to l2)
DUP ..... Duplication factor (number of this path that appear to *both* l1 and l2; sets both DUPI and DUPJ)
FF..... FF=FILL or FF=PURGE sets appropriate values of STAT, UPF, and IPDC as described in Section 3.27 for flat-front two-phase modeling
DEV ..... name for device simulation. May require extra inputs depending on device type. *Any such device specific inputs must appear at the* **end** *of the subblock following the DEV=C in the CONN subblock.* Refer to Section 3.5 for formats.

defaults (if not previously defined in a PA DEF subblock):

TWIN ..... no twin (single homogeneous path created)
For connectors, this option is only valid for DEV=STUBE.
MCH ..... -2 for most connectors: nonequilibrium phasic expansion with Bursik's metastable two-phase speed of sound (see Section 3.18). Note that the default MCH is zero (no choking detected) for MFRSET, VFRSET, PUMP, TURBINE, COMPPD, COMPRESS, and NULL connectors. MCH=-3 is the default for REDUCER, EXPANDER, and ORIFICE connectors. MCH=-3 is recommended for LOSS and STUBE connectors as well if the fluid is condensible/volatile and if a throat contraction exists (AFTH<AF or AFI/AFJ are used, or if the upstream lump is stagnant).
HCH ..... 0.02 (2% hysteresis on critical flow rate, see Section 3.18)
STAT ..... NORM

GK . . . . . .(required if DEV=NULL, otherwise calculated automatically)
HK . . . . . .zero (calculated automatically if device model is chosen)
EI . . . . . .zero (calculated automatically if twinned STUBEs are chosen)
EJ . . . . . .zero (calculated automatically if twinned STUBEs are chosen)
DK . . . . . .zero (calculated automatically if twinned STUBEs are chosen)
ROTR. . . . . .zero
RADI. . . . . .zero
RADJ. . . . . .zero
VAI 90.0
VAJ 90.0
VXI . . . . . .90.0 (radial flow)
VXJ . . . . . .90.0 (radial flow)
RVR . . . . . .1.0 (co-rotating walls)
EFFP. 1.0
TCF 1.0
INTORQ . . . . .NO (TORQ will be calculated)
TORQ. . . . . .(calculated by default)
CXI,CYI,CZI . . .CX, CY, CZ of defined upstream lump (port is inactive)
CXI,CYJ,CZJ . . .CX, CY, CZ of defined downstream lump (port is inactive)
DUPI. . . . . .1.0 (or value of DUP)
DUPJ. . . . . .1.0 (or value of DUP)
DUP 1.0
DEV . . . . . .NULL (Warning: This option is for experienced users only!)

Guidance: The order of l1,l2 indicates positive flow direction. The *defined* upstream end is often referred to as the i^{th} end, with the opposite end the j^{th} end so as to avoid confusion if flow reverses: the i^{th} and j^{th} designations do not change.

Guidance: NULL connectors are rarely used, and in fact should only be used by experienced users. Thus, while the parameters GK, HK, EI, EJ, and DK are available for direct input, they are normally calculated automatically by devices options and their presence can be ignored by casual users.

Guidance: If twinned, the initial flow rate is assumed to be the total flow rate. It will be apportioned to each twin on the basis of initial upstream quality.

Guidance: If STUBEs are twinned, all other initial values apply to both twins. If GK, HK, EI, EJ, or DK are being specified in FLOW DATA for twinned STUBEs, the user should consider modifying these values for each twin in OPERATIONS prior to a solution routine call since these parameter will rarely be equal for both twins.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump. If an STUBE is twinned, the identifier of the twin may be referred to as “#twin.” However, the “TWIN=I” declaration must be placed on an input line that is encountered prior to any reference to “#twin.”

Restriction: To refer to a twinned STUBE’s ID as “#twin” inside of an expression, the “TWIN=I” must have been input on a previous line within the same subblock.

Restriction: Phase suction options (STAT) are ignored in conjunction with twinned paths (see also NOSLIP and GOSLIP support subroutines in Section 7.11.1.3.)

Restrictions: Either a device model must be specified, or GK and HK must be provided and manipulated by the user logic. Nonpositive GK should be used cautiously.

Restrictions: Note that the DEV option provides for a variable subset of inputs (and therefore keywords). With the exception of the STUBE, REDUCER, and EXPANDER connectors, no defaults may be specified for these inputs in PA DEF subblocks.

Examples:

```
PA CONN,1,15,16,STAT=DLS, FR=17.0, DEV=MFRSET
PA CONN,777,101,102, GK=1.0E-3
PA CONN,12,443,444, TWIN = 1012, GK = 1.0
                    DEV   = NULL
```

3.4.3 Path Defaults in FLOW DATA

As can be seen from the previous format definitions, most paths (at least tubes, STUBE, REDUCER, and EXPANDER connectors) share the same parameters. To reduce inputs, the user may define the default definitions of these parameters within the FLOW DATA input stream in a separate subblock called PA DEF.

Once the value for a parameter has been specified in an PA DEF subblock, the program will use that value for all subsequent lumps if no value for that parameter is supplied by the user.* Any number of such subblocks may be used, overriding previous default values if needed, or merely adding new default values to the list.

If the program requires an input, it will search the previous PA DEF subblocks (in reverse input order) until that requirement is satisfied, or it will issue an error if no such input has been made. The parameters FR, DH, and TLEN must be defined at least once.

PA DEF Subblock Format:

```
PA DEF[,FR=R][,FRFACT=R][,MCH=I],HCH=R][,STAT=C]
        [,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]
        [,ROTR=R][,RADI=R][,RADJ=R][,VAI=R][,VAJ=R]
        [,DEFF=R][,CURV=R][,FCLM=R][,FCTM=R][,FF=C]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=R]
```

where:

FR . . . . . initial mass flow rate

FRFACT . . . . . mass flow rate multiplier

MCH . . . . . choked flow detection and modeling method flag (see Section 3.18). Note that the default value of MCH *cannot* be set for PUMP, TURBINE, COMPPD, COMPRESS, and NULL connectors: the default MCH for those connector devices remains zero unless specifically overridden in the input subblock for that path. Similarly, the default value of MCH *cannot* be set for REDUCER, EXPANDER, and ORIFICE connectors: the default MCH for those connector devices remains -3 unless specifically overridden in the input subblock for that path.

HCH . . . . . hysteresis fraction to apply to critical flow rate (see Section 3.18)

* An exception is the DH parameter, whose default (PA DEF) value will be ignored if AFI and AFJ are input.

STAT . . . . . phase suction status flag:
NORM . . . normal (extracts both phases from upstream lump)
LS . . . . . Liquid suction from l1 only
VS. . . . . Vapor suction from l1 only
DLS . . . . . Liquid suction from l1 *or* l2
DVS . . . . . Vapor suction from l1 *or* l2
RLS. . . . . Liquid suction from l2 only
RVS . . . . . Vapor suction from l2 only
LRV . . . . . Liquid suction from l1, vapor suction from l2
VRL . . . . . Vapor suction from l1, liquid suction from l2
See also Section 3.19.4 for species-specific suction options. However, note that only one default specification can apply to subsequent paths: cumulative STAT options are not permitted in PA DEF blocks.

TLEN . . . . . tube length (tubes and STUBE, REDUCER, and EXPANDER connectors)

DH. . . . . hydraulic diameter (tubes and STUBE connectors; used by all other paths if AF is neither input nor defaulted)

DEFF . . . . . Effective diameter, for cases where DH is inappropriate (tubes and STUBE connectors)

AF. . . . . flow area

AFTH . . . . . throat flow area for choking calculations

WRF . . . . . wall roughness fraction (tubes and STUBE, REDUCER, and EXPANDER connectors)

IPDC . . . . . Two-phase frictional pressure drop correlation number (tubes and STUBE connectors):
0 . . . . . User-input FC and FPOW even for single-phase flow
1 . . . . . McAdam's Homogeneous
2 . . . . . Lockhart-Martinelli
3 . . . . . Baroczy
4 . . . . . Friedel
5 . . . . . Whalley's recommendations
6 . . . . . Flow regime based two-phase friction, or:
-1. . . . . use bubbly regime
-2. . . . . use slug regime
-3. . . . . use annular regime
-4. . . . . use stratified regime
-5. . . . . use axially stratified regime (Section 3.27)

UPF . . . . . (if IPDC isn't 0) upstream fraction of properties to use in friction pressure drop calculations (tubes and STUBE connectors)

FTIE=AXIAL
Generates AXIAL fties (Section 3.7) with the same ID and endpoint lumps as the tube for convenient inclusion of axial conduction along the tube. If either of the endpoint lumps are twinned tanks and the tube or STUBE itself is also twinned, then a pair of twinned AXIAL fties are automatically generated (tubes and STUBE connectors)

HC . . . . . initial head coefficient (tubes and STUBE, REDUCER, and EXPANDER connectors)

FC . . . . . initial irrecoverable loss coefficient (tubes and STUBE connectors)

FPOW. . . . . flow rate exponent in frictional pressure drop term: $FC \cdot FR \cdot |FR|^{FPOW}$ (tubes and STUBE connectors)

AC . . . . . initial recoverable loss coefficient (tubes and STUBE connectors)

AM . . . . . initial added (virtual) mass coefficient, for twinned tubes and STUBEs

FG . . . . . initial phase generation coefficient, for twinned tubes and STUBEs

FD . . . . . initial interface friction coefficient, for twinned tubes and STUBEs

GK . . . . . initial derivative of flow rate with respect to pressure drop

HK . . . . . initial flow rate offset

EI . . . . . initial derivative of flow rate with respect to defined upstream enthalpy

EJ . . . . . initial derivative of flow rate with respect to defined upstream enthalpy

DK . . . . . initial derivative of flow rate with respect to twin's flow rate (twinned tubes and STUBE connectors)

ROTR. . . . . Spin or rotation rate in revolutions per minute (rpm). See Section 3.26.*

RADI. . . . . Radius from center of spin at defined inlet. See Section 3.26.

RADJ. . . . . Radius from center of spin at defined outlet. See Section 3.26.

VAI . . . . . Angle between flow and tangent at defined inlet. See Section 3.26.

VAJ . . . . . Angle between flow and tangent at defined outlet. See Section 3.26.

VXI . . . . . Angle between flow-tangent plane and axis at defined inlet. See Section 3.26.

VXJ . . . . . Angle between flow-tangent plane and axis at outlet. See Section 3.26.

RVR . . . . . Rotational velocity ratio (unity for co-rotating walls). See Section 3.26.

EFFP. . . . . Rotational "pumping" efficiency. See Section 3.26.

TCF . . . . . Torque conversion factor. See Section 3.26.

INTORQ . . . . . Flag: torque as an input or output. See Section 3.26.

TORQ. . . . . Hydraulic torque (if input). See Section 3.26.

DEFF. . . . . Effective diameter, for cases where DH is inappropriate

CURV. . . . . Radius of curvature for coiled tubes

FCLM. . . . . FC multiplier for laminar flow

FCTM. . . . . FC multiplier for turbulent flow

FF . . . . . FF=FILL or FF=PURGE sets appropriate values of STAT, UPF, and IPDC as described in Section 3.27 for flat-front two-phase modeling

CXI . . . . . X coordinate of inlet (if not the same as that of the upstream lump, see Section 3.13.4)

CYI . . . . . Y coordinate of inlet (if not the same as that of the upstream lump)

CZI . . . . . Z coordinate of inlet (if not the same as that of the upstream lump)

CXJ . . . . . X coordinate of outlet (if not the same as that of the downstream lump)

CYJ . . . . . Y coordinate of outlet (if not the same as that of the downstream lump)

CZJ . . . . . Z coordinate of outlet (if not the same as that of the downstream lump)

---

* Rotation terms are ignored for MFRSET, VFRSET, and NULL connectors. See Section 3.26.3.3 for interactions with PUMP and other turbomachinery connectors.

defaults (if not previously defined in a PA DEF subblock):

FRFACT .... one

MCH ..... -2 for most connectors: nonequilibrium phasic expansion with Bursik's metastable two-phase speed of sound (see Section 3.18). Note that the default MCH is zero (no choking detected) for MFRSET, VFRSET, PUMP, TURBINE, COMPPD, COMPRESS, and NULL connectors. The default MCH is -3 (Dyer's method) for REDUCER, EXPANDER, and ORIFICE connectors.

HCH ..... 0.02 (2% hysteresis on critical flow rate, see Section 3.18)

STAT ..... NORM

DEFF ..... DH

AF..... calculated using DH, assuming circular cross section, or using average of AFI and AFJ if those are input. Note: negative AF signals a circular cross section, and AF will be overwritten. This allows users to override previous defaults for noncircular cross sections.

AFTH ..... AF, or minimum of AFI and AFJ if those are input

WRF ..... zero

HC..... zero

FC..... zero (calculated automatically if IPDC isn't 0)

FPOW ..... 1.0 (calculated automatically if IPDC isn't 0)

AC..... zero (calculated automatically if paths are part of a duct macro)

IPDC ..... 6 (flow regime based pressure drop)

AM..... zero (calculated automatically if twin tubes are input)

FG..... zero (calculated automatically if paths are part of a duct macro)

FD..... zero (calculated automatically if twin paths are input and IPDC=6)

GK..... (required if DEV=NULL, otherwise calculated automatically)

HK..... zero (calculated automatically if device model is chosen)

EI..... zero (calculated automatically if twinned STUBEs are chosen)

EJ..... zero (calculated automatically if twinned STUBEs are chosen)

DK..... zero (calculated automatically if twinned STUBEs are chosen)

ROTR ..... zero

RADI ..... zero

RADJ ..... zero

VAI 90.0

VAJ 90.0

VXI ..... 90.0 (radial flow)

VXJ ..... 90.0 (radial flow)

RVR ..... 1.0 (co-rotating walls)

EFFP 1.0

TCF 1.0

INTORQ .... NO (TORQ will be calculated)

DEFF ..... DH

CURV ..... 1.0E30 (straight pipe)

FCLM. 1.0
FCTM. 1.0
CXI,CYI,CZI . . .CX, CY, CZ of defined upstream lump (port is inactive)
CXI,CYJ,CZJ . . .CX, CY, CZ of defined downstream lump (port is inactive)

Guidance: A PA DEF subblock may be used to define default values for the paths following the subblock. Any number of PA DEF subblocks may be used in any order to set or change default values as needed. These defaults operate only within the current submodel, and stay in effect until specifically overridden by another PA DEF subblock. Inputs given in any other PA subblock do not affect these defaults.

Guidance: FRFACT is used to change units or reference status. This factor applies to all subsequent flow rates *during input only* as follows: Actual FR = (input FR)*FRFACT

Guidance: While the parameters FC, FPOW, AC, AM, FG, FD, GK, HK, EI, EJ, and DK are available for direct input, they are normally calculated automatically by various options and their presence can be ignored by casual users.

Restriction Note the absence of duplication factors (DUPI and DUPJ). The defaults for these are always 1.0, which may not be reset because of the possibility for major errors that are difficult to detect.

Restriction: Use of indirect referencing (“#this”, “#up”, “#down”, and “#twin”) is illegal in PA DEF subblocks.

Examples:

```
PA DEF, DH=0.01, TLEN=1.0, AF=1.0E-4
PA DEF, IPDC=1, UPF=0.0, FRFACT=0.4536/3600.0
```

3.5 Connector Device Models

Connectors can be used to simulate a wide variety of flow devices, as summarized in Table 3-4. The only restriction on these elements is that they are time-independent—all such devices are assumed to operate under quasi steady-state conditions; they react instantaneously to changes. Also, like any path, connectors do not have any associated mass and cannot accept or reject energy.* These attributes are handled by lumps on either end of the connector.

Table 3-4 FLUINT Connector Device Models

NAME	DESCRIPTION
NULL	No device (user-input GK, HK, EI, EJ, DK.)
STUBE	Small or short (time-independent) tube. Same as a tube, but without inertia (reacts instantaneously).
REDUCER	Reducer, nozzle, or sudden contraction
EXPANDER	Expander, diffuser, or sudden expansion
CAPIL	Capillary (surface tension dominated) passage. For wicks, slots, tubules, grooves, etc.
LOSS	Generic head loss component. For losses in bends, tees, orifices, valves, etc.
LOSS2	Two way generic head loss component. Same as LOSS but direction sensitive.
CHKVLV	Check valve: LOSS element that closes for negative flow rates.
CTLVLV	Control valve: LOSS element that user may alter or close according to control logic.
UPRVLV	Upstream pressure regulator valve: LOSS-like element that opens or closes as needed to maintain a desired pressure.
DPRVLV	Downstream regulator valve: regulates downstream pressure.
ORIFICE	Sharp or long orifice, or valve (regulator, or opening/closing)
MFRSET	Maintains a constant mass flow rate. For simple pumps, boundary conditions, certain valves (including closed,) etc.
VFRSET	Maintains a constant volumetric flow rate. Similar to MFRSET.
PUMP	Centrifugal pump or fan, partial or full performance curves.
TABULAR	Generalized head/pressure-drop versus flow rate relationship input as arrays: performance mapping via interpolation tables
TURBINE	Gas, steam, or hydraulic turbine, partial or full performance curves
COMPRESS	Axial or centrifugal (variable displacement) compressor
COMPPD	Piston, vane, scroll (positive displacement) compressor

Beginning users should pay particular attention to STUBE and MFRSET connectors. Many models consist entirely of those devices.

* This does not exclude shaft work or mechanical shearing, nor even changes in kinetic energy across the path. However, these effects apply to the endpoint lumps, and not to the path itself which is isenthalpic.

Restrictions: *Any device-specific inputs must be located at the end of the CONN subblock following DEV=C*, although they may appear in any order after DEV=C. For example, if DH for an STUBE is given, it must appear after the DEV=STUBE. Only one device specification is allowed per connector.

3.5.1 NULL Option

If no prepackaged connector device model (as defined below) is appropriate for a particular modeling need and if the user is sufficiently advanced, then the NULL device option can be used. This suppresses automatic GK, HK, EI, EJ, and DK calculations of any type: the user must manipulate these variables via the logic blocks. This option provides the user with great modeling power, but has several potential pitfalls if not used carefully. Therefore, it should only be used as a last resort. It is the default option, but only because no other default is possible. ***WARNING: This option should only be used by very advanced FLUINT users, or as directed by other program options!***

The underlying coefficients in the linearized flow rate versus pressure drop relationship are defined on the basis of the equation: $\Delta FR = \Delta(\Delta PL)\bullet GK + HK + \Delta HL_{up}\bullet EI + \Delta HL_{down}\bullet EJ$, which is obeyed by *all* connectors during each solution step. GK represents the partial derivative of flow rate with respect to pressure gradient (holding enthalpies constant), and is evaluated at the conditions that *would* exist if the flow were steady under the current boundary conditions—that is, if the endpoint lumps did not change. EI and EJ represent the partials with respect to up and downstream enthalpy (holding pressures constant) at that hypothetical flow rate. HK represents the difference in flow rate between the current flow rate and the hypothetical flow rate.

Zero GK means that the flow rate is independent of pressure gradient. Zero EI and EJ mean that the flow rate is independent of enthalpy (density, quality) changes. All parameters are updated by other connector options (between FLOGIC 0 and FLOGIC 1), and are available for inspection within the logic blocks. Thus, the NULL connector represents direct access to these normally “invisible” coefficients.

Some simulation routines such as COMPRS rely on NULL connectors, and the CAPPMP macro generates NULL connectors. In all these cases, the user does not have responsibility for updating GK, HK, etc.

AF (flow area) and AFTH (throat area) factors can be applied to NULL connectors if desired. Otherwise, kinetic energy and choking calculations will not apply to NULL connectors. As with other paths, a prior default (PA DEF subblock) value of DH will be used to calculate both AF and AFTH if these values are not supplied.

Choking calculations are suspended by default for NULL connectors.

3.5.2 STUBE Option

The STUBE (short tube) model is a time-independent version of a tube. The use of a tube to model either a vapor passage or a liquid passage with a large AF/(ρ•TLEN) ratio may cause small solution steps to be taken. If such resolution is beyond the scope of the analysis desired, then the user should use a connector with an STUBE model instead. *Most thermal analyses do not require such resolution, so STUBE connectors should normally be used in preference to tubes.* However, tubes are relatively inexpensive if flow rates are not changing, and behave much more predictably than do STUBEs, especially in two-phase flows.

The operation of the STUBE model can be summarized concisely: an STUBE connector has all of the modeling options of a tube, but it reacts instantaneously to flow forces (e.g., always operates under steady-state assumptions). The governing equations for STUBE connectors correspond to those for tubes in the limits as dFR/dt goes to zero; all parameters have the same meaning and units. Note that there is no AM (added mass) term for twinned STUBE connectors since they have no acceleration associated with them. Therefore, with the exception of the inertial term AM, the STUBE connector has all the same descriptors and options as does a tube, so a detailed description will not be repeated here—refer to the previous section on tube parameters (DH, AF, HC, etc.). Both tubes and STUBE connectors may be used with HTN, HTNS, and HTNC ties, and both may be used in macrocommands generating duct models (“duct macros,” Section 3.9.2).

While STUBEs (and tubes, for that matter) might initially appear to be useful only for representing ducts, they are actually the most adaptable paths because of the access to the underlying fundamental modeling parameters HC, FC, FPOW, and AC. In fact, advanced users will find that *nearly all other devices can be equivalently simulated by manipulation of these parameters.* The HC factor can be used to simulate pumps, magnetic body forces, or constant pressure drop devices like control valves. Using the AC factor, STUBEs may be used to simulate losses and pressure recovery due to area change, side flows, and density change (refer to Section 3.15).

STUBE Format in CONN Subblocks:

```
... DEV=STUBE[,TLEN=R][,DH=R][,DEFF=R]
              [,AF=R][,AFTH=R][,AFI=R,AFJ=R]
              [,FK=R][,WRF=R][,IPDC=I][,UPF=R][,FTIE=AXIAL]
              [,HC=R][,FC=R][,FCLM=R][,FCTM=R]
              [,FPOW=R][,AC=R][,FG=R][,FD=R][,FF=C]
```

where:

`TLEN` . . . . . length
`DH` . . . . . hydraulic diameter
`DEFF` . . . . . Effective diameter, for cases where DH is inappropriate

AF . . . . . flow area (if twinned, applies to whole passage)

AFTH. . . . . throat flow area for choking calculations. Consider setting MCH=-3 if a throat is defined and if the fluid is also condensible/volatile (see Section 3.18).

AFI,AFJ. . . . flow area at the defined inlet and outlet, respectively (if one is input, they both must be). Consider also using a REDUCER or EXPANDER in series, leaving the STUBE with a constant cross section.

FK . . . . . K-factor due to associated bend losses, contained valves or orifices, etc.

WRF . . . . . wall roughness fraction

IPDC. . . . . Two-phase frictional pressure drop correlation number:

- 0 . . . . . User-input FC and FPOW even for single-phase flow
- 1 . . . . . McAdam's Homogeneous
- 2 . . . . . Lockhart-Martinelli
- 3 . . . . . Baroczy
- 4 . . . . . Friedel
- 5 . . . . . Whalley's recommendations
- 6 . . . . . (**Default.**[*]) Flow regime based two-phase friction, or:
  - -1 . . . . . use bubbly regime
  - -2 . . . . . use slug regime
  - -3 . . . . . use annular regime
  - -4 . . . . . use stratified regime
  - -5 . . . . . use axially stratified regime (Section 3.27)

UPF . . . . . (if IPDC isn't 0) upstream fraction of properties to use in friction pressure drop calculations.

FTIE=AXIAL

Generates an AXIAL ftie (Section 3.7) with the same ID and endpoint lumps as the STUBE connector for convenient inclusion of axial conduction along the path. If either of the endpoint lumps are twinned tanks and the path itself is also twinned, then a pair of twinned AXIAL fties are automatically generated.

HC . . . . . initial head coefficient (impressed pressure gradient)

FC . . . . . initial irrecoverable loss coefficient

FCLM. . . . . FC multiplier for laminar flow

FCTM. . . . . FC multiplier for turbulent flow

FPOW. . . . . flow rate exponent in frictional pressure drop term: $FC \cdot FR \cdot |FR|^{FPOW}$

AC . . . . . initial recoverable loss coefficient

FG . . . . . initial phase generation coefficient, for twinned STUBEs

FD . . . . . initial interface friction coefficient, for twinned STUBEs

FF . . . . . FF=FILL or FF=PURGE sets appropriate values of STAT, UPF, and IPDC as described in Section 3.27 for flat-front two-phase modeling

* Prior to Version 4.2, IPDC=1 was the default for single paths, and IPDC=6 for twinned paths.

defaults (if not previously defined in a PA DEF subblock):

DH..... none, unless AFI and AFJ are input, in which case DH is initialized as the diameter of the average flow area assuming a circular cross section. If AFI and AFJ are input, PA DEF values are ignored.

DEFF ..... DH

AF..... -1.0 (nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH. Otherwise, DH and AF are independent parameters.). If AFI and AFJ are input, AF is both initialized and maintained at the average: AF = 0.5*(AFI + AFJ)

AFI,AFJ ... -1.0 (not used)

AFTH ..... AF, or the minimum of AFI and AFJ if they are input.

FK..... zero

WRF ..... zero

IPDC ..... 6 (flow regime mapping)

UPF ..... 0.5 (Average of up and downstream properties)

HC..... zero

FC..... zero (calculated automatically if IPDC isn't 0)

FCLM 1.0

FCTM 1.0

FPOW ..... 1.0 (calculated automatically if IPDC isn't 0)

AC..... zero

FG..... zero

FD..... zero (calculated automatically if twin paths are input and IPDC=6)

Guidance: While the parameters FC, FPOW, AC, FG, and FD are available for direct input, they are normally calculated automatically by various options and their presence can be ignored by casual users.

Guidance: If a frictional pressure drop model other than those available is desired, or if FC is to be zero (no frictional losses) or otherwise provided by user logic, set IPDC=0. If IPDC=0, then the input values of TLEN, WRF, UPF, and DH are ignored although TLEN and DH might still be used by HTN and HTNC ties. If IPDC=0, the user assumed control of both FC and FPOW.

Guidance: IPDC=6 (the default) is strongly recommended with twinned paths, twinned tanks, and two-phase mixtures (especially those involving dissolution/evolution).

Guidance: If twinned, the initial flow rate is assumed to be the total flow rate. It will be apportioned to each twin on the basis of initial upstream quality.

Guidance: If twinned, all other initial values apply to both twins. If FC, FPOW, AC, FG, or FD are being specified in FLOW DATA for twinned STUBEs, the user should consider modifying these values for each twin in OPERATIONS prior to a solution routine call since these parameters will rarely be equal for both twins. In fact, the magnitude of FG and FD should normally differ by the ratio of densities, and FG should be of opposite sign for each phase.

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump. If the path is twinned, the identifier of the twin may be referred to as "#twin." However, the "TWIN=I" declaration must be placed on an input line that is encountered prior to any reference to "#twin."

Restriction: To refer to a twin's ID as "#twin" inside of an expression, the "TWIN=I" must have been input on a previous line within the same subblock.

Caution: If AFI and AFJ are input to model flow area changes, the corresponding irrecoverable losses should also be represented via the FK (K-factor) term. See Section 3.15. Consider also using an EXPANDER or REDUCER (Section 3.5.3 and Section 3.5.4) in series and leaving the STUBE with a constant cross section.

Restriction: Phase suction options (STAT) are ignored in conjunction with twinned STUBEs (see also NOSLIP and GOSLIP support subroutines in Section 7.11.1.3.)

Restriction: If IPDC=0, all of FK, FC, and AC cannot be zero.

Examples:

```
PA CONN,1,2,3, DEV=STUBE, FK=2.5, TLEN=14.0, WRF=0.0
      AF=1.0, DH=0.1
PA CONN,44,55,66, FR=2.0, STAT=DLS, DEV=STUBE
      IPDC=2, UPF=0.5
PA CONN,212,2,12, TWIN = 10212 $ GENERATES TWO STUBES
      DEV = STUBE, FTIE = AXIAL$ adds axial conduction
PA CONN,1213,12,13, DEV = STUBE
      AF = DH#this^2          $ #this = 1213
      FK = (PL#up >= PL#down)? 1.0 : 0.0
      AC = 1.0/(DL(#up)*AF(#this)*2)
```

3.5.3 REDUCER Option

The REDUCER option provides various models for handling the irrecoverable losses associated with a reduction in flow area. (Recoverable losses associated with static pressure and velocity changes are handled automatically, as described in Section 3.15.)

When flow reverses in a REDUCER, it essentially becomes an EXPANDER (Section 3.5.4). However, the losses and vena contracta are all very different if the flow reverses.

A reducer is completely described by an inlet diameter DHI and an outlet diameter DHJ, providing the default option of “sudden contraction” (MODEC=1) is chosen (Section 3.5.3.1).

If the device is not circular in cross section, the flow areas AFI and AFJ may be substituted at either or both ends.

For all REDUCER connectors, the AF is output as the average flow area: AF=0.5*(AFI+AFJ)). The DH is output as the effective circular diameter at that average area.

The MODEC=2 option (Section 3.5.3.2) is suitable for rounded contractions, such as a nozzle. The parameters RAD_A and RAD_R may be used to describe the elliptical curvature of such a contraction (Figure 3-4). For this option, like most others, the length TLEN is needed and the wall roughness parameter WRF is optional.

A conical reducer corresponds to MODEC=3 (Section 3.5.3.3), though this option is mostly present to support the symmetry of reversed flow in a diffuser (Section 3.5.4.3).

A smooth reducer corresponds to MODEC=4 (Section 3.5.3.4). This option should be chosen for butt-weld pipe fittings, or for the entrance to a Herschel venturi tube.

Depending on the mode chosen via the MODEC parameter, the outlet K-factor FKJ is predicted by the program as described in subsequent subsections, as is the throat area AFTH. FKI is left as a user input if augmented losses are required, and FK is output for informational purposes as the effective K-factor at the average flow area AF. FK includes the effects of both FKI and FKJ.

The user may chose to specify both AFTH and FKJ directly (as well as the optional FKI) using the MODEC=0 option.

The default choking method for REDUCER connectors is Dyer’s method (MCH=-3, see Section 3.18).

Both REDUCER and EXPANDER connectors may be used with HTN, HTNS, and HTNC ties, though they contribute no heat transfer area to such ties. They also both may be part of extended duct macros generated in Sinaps or FloCAD (Section 3.9.2.2).

3.5.3.1 Sudden Contraction (MODEC=1)

The simplest option is also the default: a sudden contraction. Rennels* Section 10.2 provides the model used:

$$FKJ = 0.0696(1-\beta^5)\lambda^2 + (\lambda-1)^2$$

$$\lambda = 1 + 0.622(1 - 0.215\beta^2 - 0.785\beta^5) = AFJ/AFTH$$

$$\beta^2 = AFJ/AFI = DHJ^2/DHI^2$$

The jet velocity ratio λ is the ratio of velocity within the vena contracta divided by velocity of the outlet for incompressible flow. This provides a convenient way of estimating the throat area for choking: AFTH = AFJ/λ. For sudden contractions, AFTH is often significantly less than AFJ.

If the flow reverses, the methods described in Section 3.5.4.1 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, a compressible correction appears (Γ>1) but there is no vena contracta (λ=1, so AFTH=AFJ).

3.5.3.2 Rounded-edged Contraction (Nozzle, MODEC=2)

A rounded edge contraction or nozzle is also available. This shape of contraction might also be applicable for the entrance of a cavitating venturi.

A single radius can be defined by specifying *either* RAD_A *or* RAD_R, leaving the other entry zero. To specify an elliptical shape, use *both* RAD_A and RAD_R as depicted in Figure 3-4.

Rennels Section 10.3 provides the model used:

$$FKJ = 0.0696\left(1 - 0.569\left(\frac{R_{eff}}{DHJ}\right)\right)\left(1 - \sqrt{\frac{R_{eff}}{DHJ}}\beta\right)(1-\beta^5)\lambda^2 + (\lambda-1)^2$$

$$\lambda = 1 + 0.622\left[1 - 0.3\sqrt{\frac{R_{eff}}{DHJ}} - 0.7\left(\frac{R_{eff}}{DHJ}\right)\right]^4 (1 - 0.215\beta^2 - 0.785\beta^5)$$

$$R_{eff} = \min\left(DHJ, \sqrt[3]{RAD\_R \cdot RAD\_A^2}\right)$$

As with the sudden contraction, the jet velocity ratio λ is the ratio of velocity within the vena contracta compared to that of the outlet for incompressible flow. This provides a convenient way of estimating the throat area for choking, AFTH = AFJ/λ. For rounded inlets, AFTH is usually slightly less than AFJ.

* Donald C. Rennels and Hobart M Hudson, Pipe Flow, John Wiley & Sons, First Edition (with Errata), 2012.

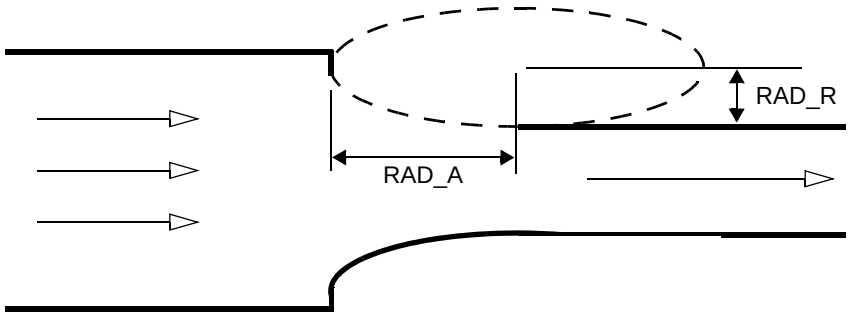


Figure 3-4: Rounded-edge Reducer (Nozzle)

If the flow reverses, the methods described in Section 3.5.4.2 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, a compressible correction appears ($\Gamma>1$) but there is no vena contracta ($\lambda=1$, so AFTH=AFJ). The radii have no effect in this reversed mode., however, TLEN becomes important, which is why it is a required input even though it does not appear in the above formulations.

3.5.3.3 Conical Contraction (MODEC=3)

For a cone of length TLEN, entrance diameter DHI and exit diameter DHJ, the full angle of the cone can be calculated as α. For α in radians, and defining f_j as the friction factor at the j^{th} (smallest) end:

$$FKJ = 0.0696 \cdot \sin(\alpha/2)(1-\beta^5)\lambda^2 + (\lambda-1)^2 + \frac{f_j(1-\beta^4)}{8\sin(\alpha/2)}$$

$$\lambda = 1 + 0.622\left(\frac{\alpha}{\pi}\right)^{0.8}(1 - 0.215\beta^2 - 0.785\beta^5)$$

$$\alpha = 2 \cdot \text{atan}[(DHI - DHJ)/(2 \cdot TLEN)]$$

These relationships are also from Rennels, Section 10.4.

Few contractions are conical by design. Instead, the purpose of this option is to support the reverse flow case of a single-stage conical diffuser (Section 3.5.4.3). In that reversed flow mode, a compressible correction appears ($\Gamma>1$) but there is no* vena contracta ($\lambda=1$, so AFTH=AFJ). The inertial losses become a strong function of α.

* Actually, note that an internal minimum of $\lambda=1.001$ is applied, so AFTH ~ 0.999*AFJ.

3.5.3.4 Smooth Contraction (Piping Reducer, MODEC=4)

For a smooth contraction, such as that used in a butt-weld piping reducer or the entrance to a venturi tube, Rennels Sections 10.6 and 10.7 suggest treating losses as if only frictional losses in an equivalent conic section existed (based on the exit friction factor f_j):

$$FKJ = \frac{f_j(1-\beta^4)}{8\sin(\alpha/2)}$$

$$\lambda = 1 = AFJ/AFTH$$

$$\alpha = 2 \cdot \text{atan}[(DHI - DHJ)/(2 \cdot TLEN)]$$

Note that there is almost no* vena contracta in this case: AFTH=AFJ.

If the flow reverses, the methods described in Section 3.5.4.4 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, a compressible correction appears (Γ>1). The frictional resistance is much less dependent on friction in this reversed flow mode, and a much stronger function of α.

3.5.3.5 REDUCER Formats in FLOW DATA

REDUCER Format in CONN Subblocks:

```
... 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]
```

where:

MODEC . . . . . Expansion/contraction device mode
- 0 . . . . . User-input FKI, FKJ, and AFTH
- 1 . . . . . Sudden contraction (sudden expansion in reverse)
- 2 . . . . . Rounded edge (defined by RAD_A, RAD_R)
- 3 . . . . . Cone (single-stage diffuser in reverse)
- 4 . . . . . Smooth (pipe fitting reducer)

DHI,DHJ . . . . diameter at the defined inlet and outlet, respectively.

AFI,AFJ . . . . . flow area at the defined inlet and outlet, respectively. Use either diameter or area at any one end, not both. One end can be defined by diameter and the other by flow area, however.

* Actually, note that an internal minimum of λ=1.001 is applied, so AFTH ~ 0.999*AFJ.

RAD_R..... radial wall radius. See Figure 3-4 above. Either or both of RAD_R and/or RAD_A is required if MODEC=2.
RAD_A..... axial wall radius if the curvature is elliptical. See Figure 3-4 above. Either or both of RAD_R and/or RAD_A is required if MODEC=2.
TLEN ..... length. Required if MODEC=2, 3, or 4.
WRF ..... wall roughness fraction: ε/DHJ. Optional if MODEC=2, 3, or 4
AFTH ..... throat flow area for choking calculations (user-specified if MODEC=0, otherwise this is automatically calculated for other MODEC values)
FKI ..... K-factor based on the dynamic pressure at the defined inlet, optional.
FKJ ..... K-factor based on the dynamic pressure at the defined outlet (optional if MODEC=0, otherwise this is automatically calculated for other MODEC values).
HC..... initial head coefficient (impressed pressure gradient)
FF..... FF=FILL or FF=PURGE sets appropriate values of STAT and IPDC as described in Section 3.27 for flat-front two-phase modeling

defaults (if not previously defined in a PA DEF subblock):

MODEC..... 1 (sudden contraction)
DHI ..... based on AFI assuming circular (for ports and other rare uses)
AFI ..... based on DHI assuming circular
DHJ ..... based on AFJ assuming circular (for ports and other rare uses)
AFJ ..... based on DHJ assuming circular
RAD_R..... zero
RAD_A..... zero
WRF ..... zero
AFTH ..... autocalculated unless MODEC=0
FKI ..... zero
FKJ ..... autocalculated unless MODEC=0
HC..... zero

Guidance: The parameters AF, DH, and FK are normally inputs for most other path types. For REDUCERs and EXPANDERs, they are outputs.

Guidance: The AC factor is also updated automatically for a REDUCER or EXPANDER connector, nearly identically to how a tube or STUBE connector operates. See Section 3.15.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Guidance: HTN, HTNC, and HTNS ties may be made to a REDUCER or EXPANDER, but they contribute zero heat transfer area to such a tie.

Guidance: In addition to tubes and STUBE connectors, REDUCER and EXPANDER connectors are accepted as part of extended duct macros in FloCAD and Sinaps (Section 3.9.2.2).

Guidance: If liquid enters the reducer and flashes in the throat, or if vapor enters and a fog forms in the throat, or if volatile two-phase fluid enters the reducer, use of MCH=-3 is strongly recommended.

Examples:

```
PA CONN,1,2,3, DEV=REDUCER, MODEC=1  $ sudden contraction
      DHI = 2./12., DHJ = 1./12.
PA CONN,44,55,66, FR=2.0, STAT=DLS, DEV=REDUCER
        DHI = 0.01,  AFJ = 0.005^2
      MODEC=2, RAD_A = 0.25*DHJ#this  $ rounded contraction
PA CONN,1213,12,13, DEV = REDUCER
      AFI = 0.05*0.07, AFJ = 0.01*0.07, TLEN = 0.1
      MODEC = 4    $ smooth transition both ends
```

3.5.4 EXPANDER Option

The EXPANDER option provides various models for handling the irrecoverable losses associated with an increase in flow area. (Recovery of static pressure associated with velocity changes are handled automatically, as described in Section 3.15.)

When flow reverses in an EXPANDER, it essentially becomes a REDUCER (Section 3.5.3). However, the losses and vena contracta are all very different if the flow reverses. There is no vena contracta* for an EXPANDER in the forward flow direction: AFTH=AFI. But in the reverse direction, AFTH<AFI, as explained below. Irrecoverable losses (total pressure changes) are large for flow in the positive flow direction of an EXPANDER, but are low in the reverse direction. Also, a compressible flow correction to the irrecoverable losses (see Γ factor below) is applied for forward expanding flow, but not for reversed contracting flow.

An expander is completely described by an inlet diameter DHI and an outlet diameter DHJ, providing the default option of “sudden expansion” (MODEC=1) is chosen (Section 3.5.4.1).

If the device is not circular in cross section, the flow areas AFI and AFJ may be substituted at either or both ends.

For all EXPANDER connectors, the AF is output as the average flow area: AF=0.5*(AFI+AFJ)). The DH is output as the effective circular diameter at that average area.

The MODEC=2 option (Section 3.5.4.2) is suitable for curved wall diffusers. The parameters RAD_A and RAD_R may be used to describe the elliptical curvature of such a contraction, but these parameters only influence irrecoverable losses and vena contracta calculations in the reversed flow

* Actually, if a REDUCER (Section 3.5.3) is upstream, its vena contracta is inherited by the EXPANDER but is assumed to exist in the inlet plane. In other words, for an inherited constriction, the EXPANDER starts at the higher velocity and reduced cross sectional area corresponding to the vena contracta.

direction. This is a rare choice, and is mostly present to support the symmetry of reversed flow in a nozzle. For this option as well as most other options, the length TLEN is needed, and the wall roughness parameter WRF is optional.

A conical expander corresponds to MODEC=3 (Section 3.5.4.3). For this option, the cone length TLEN is needed, and the wall roughness parameter WRF is optional. For long enough cone length (such that the whole cone angle is less than about 15 to 20 degrees), the expander becomes a diffuser, whose intent is to recover the maximum static pressure. Such an expander can be used to model the exit of a venturi.

A smooth expander corresponds to MODEC=4 (Section 3.5.4.4). This option should be chosen for butt-weld pipe fittings: a piping reducer used as an expander.

Depending on the mode chosen via the MODEC parameter, the inlet K-factor FKI is predicted by the program as described in subsequent subsections, as is the throat area AFTH. FKJ is left as a user input if augmented losses are required, and FK is output for informational purposes as the effective K-factor at the average flow area AF. FK includes the effects of both FKI and FKJ.

The user may chose to specify both AFTH and FKI directly (as well as the optional FKJ) using the MODEC=0 option.

The default choking method for EXPANDER connectors is Dyer's method (MCH=-3, see Section 3.18).

Both REDUCER and EXPANDER connectors may be used with HTN, HTNS, and HTNC ties, though they contribute no heat transfer area to such ties. They also both may be part of extended duct macros generated in Sinaps or FloCAD (Section 3.9.2.2).

3.5.4.1 Sudden Expansion (MODEC=1)

The simplest option is also the default: a sudden contraction. Rennels* Section 11.1 provides the model used, with a compressible flow correction added:

$$FKI = \Gamma(1-\beta^2)$$

$$\Gamma = (2 \cdot DL_{in-tot})/(DL_{in-tot} + DL_{out})$$

$$\beta^2 = AFI/AFJ = DHI^2/DHJ^2$$

The incompressible K-factor from Rennels is augmented by a Γ factor which is the ratio of inlet to average density. This factor represents a first-order attempt to include extra losses for compressibility and phase change effects.† The inlet total (stagnation) density $DL_{in\text{-}tot}$ may be adjusted for

* Donald C. Rennels and Hobart M Hudson, Pipe Flow, John Wiley & Sons, First Edition (with Errata), 2012.

† "Duct Systems," Section 405.4, Fluid Flow Division of General Electric, November 1969.

phase suction, and the outlet density DL_{out} may not exactly correspond to the DL of the downstream lump.* This correction is roughly equivalent to the ASME "Y expansion factor" used for perfect gas flows,† but allows the more general case of real gases and multiple phases.

For forward flow, the throat area for choking is calculated as AFTH = AFI (as long as a REDUCER is not upstream, in which case the EXPANDER inherits its AFTH).

If the flow reverses, the methods described in Section 3.5.3.1 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, the compressible correction disappears (Γ=1) but there is a strong vena contracta (λ>1, so AFTH=AFI/λ).

3.5.4.2 Curved Wall Diffuser (MODEC=2)

A rounded edge expansion is also available. A single radius can be defined by specifying *either* RAD_A *or* RAD_R, leaving the other entry zero. To specify an elliptical shape, use *both* RAD_A *and* RAD_R as depicted in Figure 3-4 (with the flow direction reversed from that diagram). The length of the expander TLEN is also required, and the wall roughness ratio WRF is optional.

Rennels Section 11.4 provides the correlation used. It is based on a curve fit to a solution assuming that the axial pressure gradient is constant along the diffuser: dP/dx = constant.

$$FKI = \Gamma \cdot \varphi_o \cdot (1.43 - 1.3\beta^2)(1 - \beta^2)^2$$

$$\varphi_o = 1.01 - 0.624\delta + 0.3\delta^2 - 0.074\delta^3 + 0.0070\delta^4$$

$$\delta = \min(4.0, TLEN/DHI)$$

Note that the required input of radii (RAD_A, RAD_R) is missing in the above equations. Those dimensions are required for reversed flow, which betrays the real purpose of this option: it is mostly present to preserve symmetry with a REDUCER, and it is a rare choice otherwise.

For forward flow, the throat area for choking is calculated as AFTH = AFI (as long as a REDUCER is not upstream, in which case the EXPANDER inherits its AFTH).

If the flow reverses, the methods described in Section 3.5.3.2 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, the compressible correction disappears (Γ=1) but there is a small vena contracta (λ>1, so AFTH=AFI/λ). The radii have an effect in this reversed mode, which is why they are required inputs even though they have no effect on forward flow.

---

* To account for reversed phase suction, ports, transient lags in downstream tanks, and other streams mixing into the downstream, an adiabatic expansion from the upstream state is calculated. Additional adjustments may limit this expansion to account for choked flow and property range limits in mixtures.

† ASME: $Y = 1 - (0.41 + 0.35*\beta^4)*(\Delta P/Pup)/\gamma$, where ΔP is the pressure drop (at least to the throat), γ is the specific heat ratio C_p/C_v. This expansion correction factor applied to $1/\lambda \sim C_d$ (not to the K-factor as is the case with Γ), so $K = [(1-K_{incompressible}^{1/2})/Y - 1]^2$

3.5.4.3 Conical Diffuser (MODEC=3)

Although MODEC=3 is intended to cover the case of a slight-angle "diffuser," whose purpose is recovery of static pressure, the formulation is general enough to include wide-angle cones which can have substantial irrecoverable pressure losses.

For a cone of length TLEN, entrance diameter DHI, and exit diameter DHJ, the full angle of the cone can be calculated as α. For α in radians,* and defining f_i as the friction factor at the i^{th} (smallest) end, the frictional resistance can be estimated as:

$$K_{frict} = \frac{f_i(1-\beta^4)}{8\sin(\alpha/2)}$$

$$\alpha = 2 \cdot \text{atan}((DHJ - DHI)/(2 \cdot TLEN))$$

For true diffusers, where $\alpha_d < 20$ degrees, Rennels (Section 11.2) suggests that the inertial losses can be estimated as follows, with the usual compressible correction factor Γ added:

$$K_{inertial} = 8.30[\tan(\alpha/2)]^{1.75}(1-\beta^2)^2$$

$$FKI = \Gamma(K_{inertial} + K_{frict})$$

For intermediate cone angles, $20 < \alpha_d < 60$ degrees, the $K_{inertial}$ term is dependent on β. For $\beta > 0.5$:

$$K_1 = 1.366 \cdot \sin\sqrt{\frac{2\pi(\alpha_d - 15)}{180}} - 0.170$$

$$K_{inertial} = [K_1](1-\beta^2)^2$$

For these intermediate cone angles, if $\beta < 0.5$, then the above $K_{inertial}$ term is decremented slightly:

$$K_{inertial} = \left[K_1 - 3.28(0.0625 - \beta^4)\sqrt{\frac{\alpha_d - 20}{40}}\right](1-\beta^2)^2$$

* In some formula, α is in degrees instead of radians. This will be denoted as "α_d"

For large cone angles, $60 < \alpha_d < 180$ degrees, if $\beta>0.5$ then the $K_{inertial}$ term is:

$$K_2 = 1.205 - 0.2\sqrt{\frac{\alpha_d - 60}{120}}$$

$$K_{inertial} = [K_2](1-\beta^2)^2$$

Once again, this term is modified for $\beta<0.5$:

$$K_2 = 1.205 - 3.28(0.0625 - \beta^4) - 12.8\beta^6\sqrt{\frac{\alpha_d - 60}{120}}$$

For these larger cone angles with $\alpha_d > 60$ degrees, K_{frict} is neglected, so FKI = $\Gamma * K_{inertial}$

For forward flow, the throat area for choking is calculated as AFTH = AFI, (as long as a REDUCER is not upstream, in which case the EXPANDER inherits its AFTH).

If the flow reverses, the methods described in Section 3.5.3.3 are applicable, substituting the i^{th} end of the path for the j^{th} end. In that reversed mode, the compressible correction disappears ($\Gamma=1$) but there is a small vena contracta ($\lambda>1$, so AFTH=AFI/λ).

3.5.4.4 Reducer used as Expander (MODEC=4)

For a smooth expansion, such as a butt-weld piping reducer used as an expander, Rennels Sections 11.5 suggest breaking the expander into a straight entrance (20% of length), an effective cone (60% of length) and a straight exit (20% of length). Using the entrance friction factor f_i and the exit friction factor f_j, the friction term then becomes:

$$K_{frict} = \frac{f_i(0.2 \cdot TLEN)}{DHI} + \frac{f_i(1-\beta^4)}{8\sin(\alpha_{eff}/2)} + \frac{f_j(0.2 \cdot TLEN)}{DHJ} \cdot \beta^4$$

$$\alpha_{eff} = 2 \cdot \text{atan}[(DHI - DHJ)/(2 \cdot (0.6 \cdot TLEN))]$$

Notes that the effective cone angle for the middle section, α_{eff}, is often rather large. The diffuser relationships described in Section 3.5.4.3 apply to this middle section, using this effective cone angle α_{eff}.

For forward *or* reversed flow, the throat area for choking is calculated as AFTH = AFI, (as long as a REDUCER is not upstream, in which case the EXPANDER inherits its AFTH).

If the flow reverses, the methods described in Section 3.5.3.4 are applicable, substituting the i^{th} end of the path for the i^{th} end. In that reversed mode, the compressible correction disappears (Γ=1). Irrecoverable losses are significantly reduced in the reversed flow direction, while in the forward direction, a MODEC=4 expander can have almost as large losses as does a MODEC=1 sudden expansion.

3.5.4.5 EXPANDER Formats in FLOW DATA

EXPANDER Format in CONN Subblocks:

```
... 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]
```

where:

MODEC..... Expansion/contraction device mode
- 0 ..... User-input FKI, FKJ, and AFTH
- 1 ..... Sudden expansion (sudden contraction in reverse)
- 2 ..... Rounded edge (defined by RAD_A, RAD_R, TLEN)
- 3 ..... Cone (single-stage conic diffuser)
- 4 ..... Smooth (pipe fitting reducer in reverse)

DHI,DHJ ... diameter at the defined inlet and outlet, respectively.

AFI,AFJ ..... flow area at the defined inlet and outlet, respectively. Use either diameter or area at any one end, not both. One end can be defined by diameter and the other by flow area, however.

RAD_R..... radial wall radius. See Figure 3-4 above. Either or both of RAD_R and/or RAD_A is required if MODEC=2.

RAD_A..... axial wall radius if the curvature is elliptical. See Figure 3-4 above, with the flow direction reversed. Either or both of RAD_R and/or RAD_A is required if MODEC=2.

TLEN ..... length. Required if MODEC=2, 3, or 4.

WRF ..... wall roughness fraction: ε/DHI. Optional if MODEC=2, 3, or 4

AFTH ..... throat flow area for choking calculations (user-specified if MODEC=0, otherwise this is automatically calculated for other MODEC values)

FKI ..... K-factor based on the dynamic pressure at the defined inlet, optional.

FKJ ..... K-factor based on the dynamic pressure at the defined outlet (optional if MODEC=0, otherwise this is automatically calculated for other MODEC values).

HC..... initial head coefficient (impressed pressure gradient)

FF..... FF=FILL or FF=PURGE sets appropriate values of STAT and IPDC as described in Section 3.27 for flat-front two-phase modeling

defaults (if not previously defined in a PA DEF subblock):

MODEC . . . . . 1 (sudden contraction)
DHI . . . . . based on AFI assuming circular (for Ports and other rare uses)
AFI . . . . . based on DHI assuming circular
DHJ . . . . . based on AFJ assuming circular (for Ports and other rare uses)
AFJ . . . . . based on DHJ assuming circular
RAD_R . . . . . zero
RAD_A . . . . . zero
WRF . . . . . zero
AFTH. . . . . autocalculated unless MODEC=0
FKI . . . . . autocalculated unless MODEC=0
FKJ . . . . . zero
HC . . . . . zero

Guidance: The parameters AF, DH, and FK are normally inputs for most other path types. For REDUCERs and EXPANDERs, they are outputs.

Guidance: The AC factor is also updated automatically for a REDUCER or EXPANDER connector, nearly identically to how a tube or STUBE connector operates. See Section 3.15.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Guidance: HTN, HTNC, and HTNS ties may be made to a REDUCER or EXPANDER, but they contribute zero heat transfer area to such a tie.

Guidance: In addition to tubes and STUBE connectors, REDUCER and EXPANDER connectors are accepted as part of extended duct macros in FloCAD and Sinaps (Section 3.9.2.2).

Examples:

```
PA CONN,1,2,3, DEV=EXPANDER, MODEC=1  $ sudden expansion
      DHI = 1./12., DHJ = 2./12.
PA CONN,44,55,66, FR=2.0, STAT=VLS, DEV=EXPANDER
        AFI = 0.005^2,  DHJ = 0.01, TLEN = 0.005
      MODEC = 4    $ piping reducer as expander
PA CONN,1213,12,13, DEV = EXPANDER
      DHI = 0.01, DHJ = 0.02, TLEN = 0.1, WRF = 1.0e-4
      MODEC = 3    $ diffuser
```

```
C Herschel Venturi tube example: reducer->expander
C
PA CONN,1,1,2,DEV=REDUCER
     MODEC = 4  $ Smooth. Cavitating venturis often use MODEC=2
    DHI = 1./12., DHJ = 0.5/12., TLEN = 0.1/12.
LU JUNC,2
PA CONN,2,2,3,DEV=EXPANDER
    MODEC = 3  $ Diffuser, single-stage conic
    DHI = 0.5/12., DHJ = 1.0/12., TLEN = 4.25/12., WRF=3.0e-5
```

3.5.5 CAPIL Option

Surface tension forces can become dominant through small passages if both vapor (or gas) and liquid are present. FLUINT allows simulation of such passages with the CAPIL connector device. The basic operation of such a passage is defined as follows: liquid may always pass through the path, but vapor and gas cannot pass if the capillary limit (or bubble point) of the passage is not exceeded and any liquid is present to block the passage of vapor. All flow through the connector is assumed to be laminar because of the typically small effective diameter of such devices. There is no preferred direction: blockage may occur in either direction depending on conditions.

CAPIL connectors may be used to model wicks, grooves, tubules, thin slots, etc. They might be used as part of a model for gas traps, liquid acquisition devices, liquid/vapor separators, etc. However, there is one restriction on the use of this device: the STAT flag is disabled. This is a result of the internal use of liquid/vapor suction action to simulate the device. Also, it should be noted that a multi-element model (CAPPMP) exists which performs the same basic function as CAPIL, but allows heat addition/rejection and therefore can simulate capillary pumping (see Section 3.9.3). CAPIL connectors can stop vapor flow but cannot simulate pumping.

The following paragraphs describe CAPIL descriptors and their use. As with all connector models, all descriptors are available to the user in the logic blocks unless noted.

CAPIL has four descriptors: RC, CFC, XVH, and XVL. The last two parameters have default values, and will be discussed in Section 3.11.2. The first two, RC and CFC, have no defaults and will be described here. First, the definitions of RC and CFC are presented to aid in understanding them, and then methods of calculating them are given.

RC is the effective capillary radius of the meniscus if it were two-dimensional, defined such that the maximum capillary pressure that can be developed is:

$$\Delta P_c = 2\,\sigma / RC$$

where:

ΔP_c . . . . . Maximum capillary pressure gain or bubble point (psf in English units)
RC . . . . . Effective two-dimensional capillary radius (units of length)
σ . . . . . Fluid surface tension (units of force per length)

A spherical meniscus that forms in a circular passage has a radius RC. These menisci are typically found in tubules, although most wicking material is also described by an effective spherical pumping radius. A one-dimensional meniscus in the shape of a cylinder of radius R, would have an RC value of 2*R. Such one-dimensional menisci are typically found in open grooves and slots. In general, if the meniscus shape is described by two orthogonal radii R_1 and R_2, the input RC value should be $2\bullet((R_1)^{-1} + (R_2)^{-1})^{-1}$. Thus, a square groove of width 1.0 units cut in the inside the circumference of a pipe of radius 10.0 units would have an RC of 2.0/(1.0 + 0.1) = 1.82 units.

Nonpositive RC values are interpreted as perfect capillary devices capable of withstanding or producing an infinite pressure gradient. This is useful for preliminary design work or for controlling steady-state solutions to prevent deprime.

CFC is the capillary flow conductance term, used to find the effective flow resistance of the path depending on the fluid and the phase of the fluid within the path. CFC is defined (for a single phase within the path) as:

$$CFC = FR * \mu / (g_c * DL * \Delta P_f)$$

where:

FR . . . . . Mass flow rate
μ. . . . . Fluid dynamic viscosity
g_c . . . . . Unit conversion factor for English units
DL . . . . . Fluid density
ΔP_f. . . . . Pressure drop due to friction

The usage of CFC is slightly more complicated for two-phase flow, which may only occur when the capillary limit is exceeded. As is typical with other paths, homogeneous phase distribution is assumed, but a minimum pressure drop corresponding to the capillary limit applies if vapor or gas is passing through the device.

The following sections describe how to calculate RC and CFC for specific cases: tubules (and grooves), a uniform wick, and a thin slot.

For circular tubules, RC is simply one half of the diameter. CFC is then calculated using:

$$CFC = n * AF * DH**2 / (32.0 * TLEN)$$

where:

n. . . . . Number of tubules in parallel (duplication factors may also be used for multiple tubules, as described in a later section)
AF . . . . . Flow area of each tubule
DH. . . . . Hydraulic diameter of each tubule
TLEN . . . Length of each tubule

A single cylindrical tubule having the same laminar flow resistance as a CAPIL would have a diameter and length that follows the relationship $DH^2/TLEN = 128 \bullet CFC/\pi$.

Note that DH and TLEN are arbitrary names chosen here because of their familiarity—unlike tubes and STUBE connectors, CAPIL connectors do not accept these keywords as parameters. Also note that the formula given above can be converted into one applicable to open grooves by using appropriate values of AF, DH, and TLEN. In this case, the value of RC would be equal to the width of the open side of the groove (not half the width since a groove meniscus is one-dimensional).

For a uniform (isotropic) wick, the capillary radius should be available from test data, vendor data, or estimates. The same is true for the properties defining CFC. These are:

$$CFC = A * P / X \qquad \text{(flat plate)}$$

$$CFC = 2.0 * \pi * P * Y / ALOG(R_o/R_i) \qquad \text{(cylinder)}$$

where:

A. . . . . Wick total frontal area (perpendicular to the flow direction)
P . . . . . Wick permeability (units of area)
X. . . . . Wick thickness (in direction of flow)
Y. . . . . Wick length (perpendicular to the flow direction)
R_o . . . . . Wick outer radius
R_i . . . . . Wick inner radius
ALOG . . . Natural logarithm

The reader might notice that both of these equations are direct analogs to the equation for net conductance in a solid, with the permeability substituted for the thermal conductivity. Thus, the CFC for other more complicated wick geometries can be derived accordingly.

For a thin slot, RC is equal to the slot width W (because of the one-dimensional meniscus). CFC is then:

$$CFC = A * W**3 / (12.0 * X)$$

where:

A. . . . . Slot breadth (perpendicular to the flow direction—equals frontal area divided by W)
W . . . . . Slot width (gap) size
X. . . . . Slot length (in flow direction)

The user should be wary of the underlying assumptions in such a simplified model, and their impact on modeling a real capillary device, especially in off-design simulations:

1) The CAPIL, like any path, is fundamentally a one-dimensional component: it must be either primed or not. Real devices can exhibit more two-dimensional effects such as preventing vapor backflow in some regions of the wick, while simultaneously leaking vapor back into the liquid space in other locations.

2) The endpoint lumps are perfectly mixed, having a single thermodynamic state. The WICK iface (Section 3.8.3.4) has been used to model the spring-like meniscus, special care is needed to model behavior during deprime, or during clearing of the vapor lump.

3) While real wicks can exhibit considerable hysteresis (on the order of 50%) between the pressure differences at bubble formation and cessation, only a very small hysteresis (about 4%) is employed internally, purely for numerical stability.

CAPIL Format in CONN Subblocks:

```
... DEV=CAPIL, RC=R, CFC=R [,XVH=R][,XVL=R][,AF=R][,AFTH=R]
```

where:

RC . . . . . effective two-dimensional capillary radius
CFC . . . . . capillary flow conductance through connector
XVH . . . . . capillary priming dryness, upper quality limit
XVL . . . . . capillary priming dryness, lower quality limit
AF . . . . . flow area for kinetic energy and choking calculations
AFTH. . . . . throat area for choking calculations

defaults:

XVH 0.99
XVL 0.95
AF . . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if AF is nonpositive then no kinetic energy can be extracted nor choking calculations applied). For porous materials, consider a value of the frontal area multiplied by the porosity.
AFTH. . . . . AF

Examples:

```
PA CONN,10,1,15, DEV=CAPIL, RC=10.0E-6, CFC=1.0E-11
PA CONN,1,5,10, FR=0.05, DEV=CAPIL
      RC  = PoreSize*Margin
      CFC = porosity*(RC#this)^2
```

Guidance: Nonpositive values of RC will be interpreted as a perfect, infinite pressure difference capillary structure. This is useful in reducing the chance of deprime during steady-state runs.

Guidance: XVH and XVL represent the limits of a hysteresis cycle. An adjacent lump with a quality higher than the current value (XVH or XVL, depending on past history) is considered dry enough to permit the device to prime, or perhaps too dry to permit priming, depending on whether the lump has higher or lower pressure than the opposite lump.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Caution: A CAPIL connector does not by itself simulate dynamic wicking or capillary pumping; it can only prevent vapor flow when not deprimed. Refer to the CAPPMP macrocommand to satisfy capillary pumping simulation requirements.

Restrictions: STAT is overridden for this device, and so cannot be set independently.

3.5.6 LOSS Option

In addition to frictional losses in pipes, pressure head may be lost due to fittings such as bends, tees, orifices, valves, filters, etc. FLUINT provides a connector option called LOSS to simulate such head losses, which are characterized by a loss factor that is independent of flow direction through the path.

LOSS connectors are characterized by four parameters (FK, AF, AFTH, and TPF), all of which are available to the user to modify within logic blocks. (This capability can be used to simulate an in-line control valve according to the user's control logic.) With the exception of TPF, all LOSS connector parameters are also available in STUBE connectors.

FK is the loss element resistance factor, commonly called a *K factor*. (See also the description of FK provided in Section 3.4.1.2). AF is the effective flow area of the surrounding ducts (used as a basis for calculating velocity and head). AFTH is the effective flow area of any internal throat at which choking might occur (used only in optional routines CHKCHP, CHKCHL, and CHOKER, as described in Section 7.11.9.5). TPF is an additional loss factor for two-phase flow effects. Each of these will be discussed below. The defining relationship for LOSS elements in single-phase flow is:

$$\Delta P = FK * (FR/AF)**2 / (2.0 * g_c * DL)$$

where:

- ΔP. . . . . Pressure drop caused by element
- FK . . . . . K factor
- FR . . . . . Mass flow rate
- AF . . . . . Flow area (based on diameter used for K factor)
- g_c . . . . . Unit conversion factor for English units
- DL . . . . . Fluid density

For two-phase flow through the connector, the relationship is expanded to include phase densities and TPF:

$$\Delta P = FK * (FR/AF)**2 \quad \frac{(1.0 + TPF * XL * (DL_l / DL_v - 1.0))}{(2.0 * g_c * DL_l)}$$

where:

- TPF . . . . . Two-phase loss factor
- XL . . . . . Fluid quality
- DL_l . . . . . Liquid density
- DL_v . . . . . Vapor density

FK can be any nonzero number. Tables of these values are available for common pipe fittings in any elementary fluid mechanics text,* and in many engineering handbooks.†Section 7 offers several routines for calculating K-factors due to area changes (FKCONE) as well as losses in merging and diverging flows (YTKALI, YTKALG). K-factors can also be found using software such as piping system analyzers, several of which can be used as FLUINT model builders.

A few words of caution should be noted here. First, AF should correspond to the pipe size used to find FK in a table. Second, K factors are only approximate, and this approximation can be poor for laminar flow (specifically, below Reynolds numbers of about 10,000, where losses are highly geometry dependent). FK is available in user logic and expressions to be changed as needed to fit the analysis requirements.

K factors are based on flow velocity. At a given mass flow rate, the velocity increases as the quality increases for two-phase flows (according to a homogeneous flow assumption). This is taken into account in the above equation. However, additional losses usually result beyond that predicted by a homogeneous assumption. This may be taken into account using a value of TPF greater than 1.0. Fitzsimmons‡ gives information on finding TPF, and Rohsenow and Harnett** contains a short excerpt of this information. However, the default value for TPF is 1.0 (corresponding to only homogeneously predicted losses), and this should be used unless appropriate information is available. Once again, *if you don't understand it, ignore it.*

LOSS devices (and the related devices that follow) normally only represent pressure losses. **Negative values of FK are allowed, but should be used with caution since they can be mathematically unstable.** (In the CTLVLV option, the negative sign is used as a flag to close the valve.)

The user is also cautioned against using absolute values of FK that are too small (less than say 1.0e-3), and against using LOSS-type devices in passages with very small flow rates—**the flow conductance is infinite at zero flow rate**. Both uses can adversely affect program accuracy and stability analogous to the disruption caused in SINDA by the use of conductors with very large G. Such strong connections blur the necessary distinction between adjacent lumps or adjacent nodes, violating the basic assumptions of finite networks.

The analyst should consider using the FK term in tubes and STUBE connectors instead of LOSS connectors. Combining both attributes into one path not only helps eliminate unnecessary lumps, it helps avoid some of the aforementioned limitations.

LOSS Format in CONN Subblocks:

```
... DEV=LOSS, FK=R [,AF=R][,AFTH=R][,TPF=R]
```

---

* See for instance: F. M. White, *Fluid Mechanics,* 1979.

† *Flow of Fluids through Valves, Fittings, and Pipe* (Crane Technical Paper 410)
D.S. Miller, *Internal Flow Systems*, ISBN 0-87201-020-1 (2nd edition, 1990)
I.E. Idelchik, *Handbook of Hydraulic Resistance*, ISBN 1-56700-074-6 (3rd edition, 1993)
Donald C. Rennels and Hobart M Hudson, Pipe Flow, John Wiley & Sons (1st Edition, 2012)

‡ D.E. Fitzsimmons, *Two-Phase Pressure Drop in Piping Components,* HW-08970 Rev. 1, March 20, 1964. (A report by GE for the AEC available from the Office of Technical Services, Department of Commerce.)

** W.M. Rohsenow & J.P Harnett, *Handbook of Heat Transfer,* 1973.

where:

FK . . . . . .K factor (head loss factor) through connector
AF . . . . . .Effective flow area (basis of velocity)
AFTH. . . . . .throat flow area for choking calculations
TPF . . . . . .Two-phase factor (for nonhomogeneous losses)

defaults:

AF . . . . . .AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH. . . . . .AF
TPF . . . . . .1.0 (homogeneous)

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

Restrictions: FK must be nonzero.

Examples:

```
PA CONN,40,1,5, FR=22.0, DEV=LOSS, FK=2.5
PA CONN,1,1,2, DEV=LOSS, FK=1.0, AF=5.0E-4, TPF=1.5
PA CONN,23,123,124, DEV = LOSS, AF = AF5019
      FK = (case = worst)? margin*3.1 : 0.34
      AFTH = sqrt(ratio)*AF(#this)
```

3.5.7 LOSS2 Option

LOSS2 is the same as LOSS, but loss coefficients are allowed to differ according to flow direction. Thus, only one new parameter is required, FKB, which is the K factor corresponding to backward (negative FR) flow rates. FK is the forward K factor, and the default for FKB is FK, in which case the connector is indistinguishable from a LOSS device.

LOSS2 may be used for certain flow control devices such as check valves (that do not completely close) or any other "diode" action device.

LOSS2 Format in CONN Subblocks:

```
... DEV=LOSS2, FK=R [,FKB=R][,AF=R][,AFTH=R][,TPF=R]
```

where:

FK. . . . . K factor (head loss factor) through connector if flow is positive (forward)
FKB . . . . . K factor (head loss factor) through connector if flow is negative (backward)
AF. . . . . Effective flow area (basis of velocity)
AFTH . . . . . throat flow area for optional choking calculations
TPF . . . . . Two-phase factor (for nonhomogeneous losses)

defaults:

FKB . . . . . FK
AF. . . . . AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH . . . . . AF
TPF . . . . . 1.0 (homogeneous)

Examples:

```
PA CONN,4,5,8, STAT=VS, DEV=LOSS2, FK=1.0, FKB=10.0
PA CONN,66,12,13, DEV=LOSS2, FK=12.0, AF=-1.0
      FKB=0.5, TPF=2.0
PA CONN,31,30,32, FR = 10.0
      DEV = LOSS2,FK = 3.0, FKB = 2.0*FK#this
      AF = 0.25*pi*DH(293)**2
```

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Restrictions: FK and FKB must be nonzero.

3.5.8 CHKVLV Option

CHKVLV may be used to simulate a check valve. A CHKVLV connector has all the same options and operations as does a LOSS connector, except that the valve will close automatically if the flow rate reverses *or if the pressure gradient reverses*, signaling impending flow rate reversal. The positive flow rate direction for the connector is assumed to be the nominal (open) flow direction. The valve will behave like a LOSS element when open (with a nonzero FK), and like an MFRSET connector with zero flow rate when shut. For this reason, a shut valve has all the same restrictions as an MFRSET or VFRSET connector (see below).

CHKVLV Format in CONN Subblocks:

```
... DEV=CHKVLV, FK=R [,AF=R][,AFTH=R][,TPF=R]
```

where:

FK . . . . . .K factor (head loss factor) through open check valve (nonzero)
AF . . . . . .Effective flow area (basis of velocity when open)
AFTH. . . . . .throat flow area for choking calculations
TPF . . . . . .Two-phase factor (for nonhomogeneous losses when open)

defaults:

AF . . . . . .AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH. . . . . .AF
TPF . . . . . .1.0 (homogeneous)

Guidance: Check valve will close if the flow rate through the valve is negative. For positive flow, the device corresponds to a LOSS connector in every way.

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

Restriction: A valve cannot close when used in series with an MFRSET or VFRSET connector, and a junction cannot be surrounded only by closed valves and MFRSET/VFRSET connectors. FK must be nonzero.

Examples:

```
PA CONN,1011,10,11, DEV=CHKVLV, FK=0.05, AF=AF1029
PA CONN,1213,12,13, FR=2.0,STAT=NORM, DEV=CHKVLV
      FK=1.0
```

3.5.9 CTLVLV Option

CTLVLV may be used to simulate a control valve. With one exception, a CTLVLV connector has all the same options and operations as does a LOSS connector—the user is expected to describe the control valve operation via FK manipulations in logic blocks. The one exception is that the user may close the valve by using a negative FK. The valve may be opened and closed as desired, but FK can never be zero (i.e., there must exist some head loss across the valve, however small, when it is not shut). The valve will therefore behave like a LOSS element when open (i.e., FK is positive), and like an MFRSET connector with zero flow rate when shut. For this reason, a shut valve has all the same restrictions as an MFRSET or VFRSET connector (see below).

In many engineering applications, the performance of a control valve is given in terms of "C_v." This parameter is specific to English units and is usually used for water. To convert to unitless FK, use the following relationship (where AF is the valve area *in standard FLUINT English units* of ft^2):*

$$FK = 3.0E7 * (AF/CV)**2$$

Finally, a word of caution regarding changing K factors (for valves or loss elements) within logic blocks: unlike changes to thermal T, C, Q, and G factors, changes to FLUINT factors may result in instantaneous and complete changes throughout the fluid network. K factors should be adjusted gradually. The user must also avoid changes that result in oscillations; the value sensed should not be too dependent on the control valve reaction. Most control valve logic will require some sort of numeric damping. For these reasons, *the user may find that STUBE connectors with negative HC may make simulations of certain control valves easier, especially pressure regulating valves. The MODA option for ORIFICE connectors also allows autonomous regulation of an up or downstream pressure.*

Also, for flow losses that are not proportional to the square of the flow rate, consider CAPIL connectors for laminar losses (i.e., losses linearly proportional to flow rate) and STUBE connectors if the loss is proportional to some other exponent of flow rate. In the latter case, the user would set IPDC=0, and then assume responsibility for the manipulation of FC and FPOW in FLOGIC 0.

CTLVLV Format in CONN Subblocks:

```
... DEV=CTLVLV, FK=R [,AF=R][,AFTH=R][,TPF=R]
```

where:

FK. . . . . K factor (head loss factor) through open control valve (nonzero)
AF. . . . . Effective flow area (basis of velocity when open)
AFTH . . . . . throat flow area for choking calculations
TPF . . . . . Two-phase factor (for nonhomogeneous losses when open)

defaults:

AF. . . . . AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH . . . . . AF
TPF . . . . . 1.0 (homogeneous)

Guidance: Control valve will close if the FK is negative. For positive FK, the device corresponds to a LOSS connector in every way. It is the user's responsibility to manipulate FK (and/or AF if desired) within user logic blocks according to the desired control algorithm. Note that the control algorithm should mimic reality; FK changes slowly and gradually.

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

* Crane Technical Paper 410, Flow of Fluids Through Valves, Fittings, and Pipe. December 2001 printing, page A-31: $K = 891d^4/C_v^2$, where d is in inches.

Restriction: A valve cannot close when used in series with an MFRSET or VFRSET connector or an operating capillary device, and a junction cannot be surrounded only by closed valves and MFRSET/VFRSET connectors.

Examples:

```
PA CONN,12,13,14, DEV=CTLVLV, FK=-1.0
PA CONN,112,44,19, FR=0.01, DEV=CTLVLV, AF=-1.0
        FK=0.01, TPF=1.1
```

3.5.10 UPRVLV Option (*Upstream* Pressure Regulator)

UPRVLV is similar to the LOSS element, except that the K-factor resistance (FK) is not set by the user, but rather is calculated by the program as needed to maintain the pressure at the *defined upstream* lump to be PSET (the set point pressure). The user may place upper and lower limits on the K-factor (FKH and FKL, respectively), and may set its initial value. Otherwise, the FK value becomes an output variable available for inspection but which cannot be directly specified by the user.

As an alternative to UPRVLV connectors which can include more realistic lags in transients, consider also the MODA=-1 mode for a controllable ORIFICE connector (Section 3.5.12.2).

UPRVLV Format in CONN Subblocks:

```
... DEV=UPRVLV, PSET=R [,FKH=R][,FKL=R][,FK=R][,RLAG=R]
      [,AF=R][,AFTH=R][,TPF=R]
```

where:

PSET . . . . . Pressure set-point for defined *upstream* lump (in user's pressure units)
FKH . . . . . Maximum K factor (head loss factor) through connector
FKL . . . . . Minimum K factor (head loss factor) through connector
FK . . . . . *Initial* K factor
RLAG. . . . . Lag factor (from zero to one, inclusive)
AF . . . . . Effective flow area (basis of velocity)
AFTH. . . . . throat flow area for choking calculations
TPF . . . . . Two-phase factor (for nonhomogeneous losses)

defaults:

FKH 1.0E15
FKL 1.0E-7
FK. . . . . 1.0E4 (if FKH and FHL defaulted, otherwise logarithmic average of those)
RLAG . . . . . 0.0 (no additional lag)
AF. . . . . AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH . . . . . AF
TPF . . . . . 1.0 (homogeneous)

Examples:

```
PA CONN,14,25,68, DEV=UPRVLV, PSET = 14.7
PA CONN,33,44,55, FR = 1.0, DEV= UPRVLV
      FKH = 20000.0, FHL = 0.1, PSET = 1.0E6
PA CONN,3000,4000,5000, DEV = UPRVLV, PSET = PL2000 + 10.0
```

Guidance: *The maintenance of the desired pressure is not perfect nor immediate.* The code will seek the correct resistance iteratively. By default (RLAG=0.0), it will seek and maintain the required resistance as fast as possible, but inevitably there will be some delay, lag, and overshoot (or undershoot). If needed for stability, additional lag can be added by using nonzero values of RLAG: the higher the value, the greater the lag. A value of 1.0, for example, means infinite lag: no changes to the valve position. Generally, lag should only be added as needed to assist convergence of steady state runs, or to avoid jittery responses in transients if junctions or hard-filled tanks are used at the end-points. For more realistic time-dependent lags, consider also the MODA=-1 mode for a controllable ORIFICE connector (Section 3.5.12.2).

Guidance: The valve will continue to operate to control the pressure of the designated lump even if the flow rate reverses. In other words, the *defined* upstream lump is regulated, not the *current* upstream lump.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Restrictions: In some cases, the desired set point cannot be achieved, in which case the valve will typically hit the FKL or FKH limits. The control valve does not act like a pump: it can only regulate via variable flow resistance.

Restrictions: FKH and FKL must be positive, and FKH must be greater than or equal to FKL. RLAG must be between 0.0 and 1.0, inclusive.

3.5.11 DPRVLV Option (*Downstream* Pressure Regulator)

DPRVLV is similar to UPRVLV, except that it regulates the pressure on the defined *downstream* end.

DPRVLV is similar to the LOSS element, except that the K-factor resistance (FK) is not set by the user, but rather is calculated by the program as needed to maintain the pressure at the *defined downstream* lump to be PSET (the set point pressure). The user may place upper and lower limits on the K-factor (FKH and FKL, respectively), and may set its initial value. Otherwise, the FK value becomes an output variable available for inspection but which cannot be directly specified by the user.

As an alternative to DPRVLV connectors which can include more realistic lags in transients, consider also the MODA=-2 mode for a controllable ORIFICE connector (Section 3.5.12.2).

DPRVLV Format in CONN Subblocks:

```
... DEV=DPRVLV, PSET=R [,FKH=R][,FKL=R][,FK=R][,RLAG=R]
      [,AF=R][,AFTH=R][,TPF=R]
```

where:

PSET. . . . . Pressure set-point for defined *downstream* lump (in user's pressure units)
FKH . . . . . Maximum K factor (head loss factor) through connector
FKL . . . . . Minimum K factor (head loss factor) through connector
FK . . . . . *Initial* K factor
RLAG. . . . . Lag factor (from zero to one, inclusive)
AF . . . . . Effective flow area (basis of velocity)
AFTH. . . . . throat flow area for choking calculations
TPF . . . . . Two-phase factor (for nonhomogeneous losses)

defaults:

FKH 1.0E15
FKL 1.0E-7
FK . . . . . 1.0E4 (if FKH and FHL defaulted, otherwise logarithmic average of those)
RLAG. . . . . 0.0 (no additional lag)
AF . . . . . AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH. . . . . AF
TPF . . . . . 1.0 (homogeneous)

Examples:

```
PA CONN,14,25,68, DEV=DPRVLV, PSET = 14.7
PA CONN,33,44,55, FR = 1.0, DEV= DPRVLV
      FKH = 20000.0, FHL = 0.1, PSET = 1.0E6
PA CONN,3000,4000,5000, DEV = DPRVLV
      PSET = PL2000 + 10.0
      FKH = 10.0*FK301, FKL = 0.01*FK301
      RLAG = MAX(0.0,MIN(1.0,100*(0.01-DTIMUF)))
```

Guidance: *The maintenance of the desired pressure is not perfect nor immediate.* The code will seek the correct resistance iteratively. By default (RLAG=0.0), it will seek and maintain the required resistance as fast as possible, but inevitably there will be some delay, lag, and overshoot (or undershoot). If needed for stability, additional lag can be added by using nonzero values of RLAG: the higher the value, the greater the lag. A value of 1.0, for example, means infinite lag: no changes to the valve position. Generally, lag should only be added as needed to assist convergence of steady state runs, or to avoid jittery responses in transients if junctions or hard-filled tanks are used at the end-points. For more realistic time-dependent lags, consider also the MODA=-2 mode for a controllable ORIFICE connector (Section 3.5.12.2).

Guidance: The valve will continue to operate to control the pressure of the designated lump even if the flow rate reverses. In other words, the *defined* downstream lump is regulated, not the *current* downstream lump.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Restrictions: In some cases, the desired set point cannot be achieved, in which case the valve will typically hit the FKL or FKH limits. The control valve does not act like a pump: it can only regulate via variable flow resistance.

Restrictions: FKH and FKL must be positive, and FKH must be greater than or equal to FKL. RLAG must be between 0.0 and 1.0, inclusive.

3.5.12 ORIFICE Option

The ORIFICE connector device can be used to model a square or circular orifice flow restriction.* It is also a useful option for modeling leaks, imperfect seals (e.g., a single tooth of a labyrinth seal), and partially opened valves including pressure and back-pressure regulators.

The user must supply both the upstream flow area AF and the opening (hole) area AORI. Although the upstream and downstream flow areas are assumed equal, and although compressibility effects may be optionally included, the calculated K-factor (FK) uses the upstream density for the dynamic head basis.† If AORI is zero, the orifice is considered closed to flow (e.g., a valve that has been shut).

To estimate the resistance of the orifice, a built-in correlation may be used by selecting MODO=1, where MODO is the integer mode flag for the orifice connector. Alternatively, the user may supply either the discharge coefficient (C_d, using MODO=2) or the K-factor (MODO=3). In all three cases, the size of the vena contracta‡ (the point of minimum static pressure and maximum velocity) is estimated and is applied as the throat area (AFTH) for choking calculations. If MODO is negative, the ORIFICE connector will include an estimation of compressible flow effects as explained below.

The default choking method for ORIFICE connectors is Dyer's method (MCH=-3, see Section 3.18).

The built-in correlation, which is applicable to both sharp-edged and long orifices in incompressible flow, is based on Hooper.** The K-factor is a strong function of the area restriction (d^2/D^2 for a circular orifice, or AORI/AF in FLUINT allowing for the more general case of noncircular orifices) and is a weak function of the upstream Reynolds number (Re). For Re < 2500:

$$K = \left(2.72 + \frac{AORI}{AF}\left(\frac{120}{Re} - 1\right)\right)\left(1 - \frac{AORI}{AF}\right)\left(\left(\frac{AF}{AORI}\right)^2 - 1\right)\left(0.584 + \frac{0.0936}{ELLD^{1.5} + 0.225}\right)$$

where ELLD is the ratio of hole length to diameter for a long orifice (L/d). ELLD is zero for a sharp-edged orifice.

---

* Orifices are also used as flow metering devices. However, the ORIFICE model in FLUINT concentrates on the overall resistance of the device as would be evidenced by a static pressure measured several diameters below the restriction, and not at 1.5 diameters (near the vena contracta or point of minimum static pressure).

† This K-factor is to be contrasted with a different K-factor based on the dynamic head at the throat (K_t), such as is used in SAE AIR 1168/1, Section 3.3.5 pp.39-40. Typically, K_t is on the order of 1 to 2, whereas FK for a FLUINT orifice is typically several orders of magnitude larger.

‡ The vena contracta is the point of minimum static pressure and maximum velocity that exists a short distance downstream of a sharp orifice, and usually within a long orifice. The vena contracta area is smaller than the orifice opening by about 60 to 85%. The ratio of the vena contracta area to the orifice opening area is the contraction coefficient, C_c (nearly equal to the discharge coefficient, C_d). The assumption that the throat area for choking is the same as the size of the vena contracta is an approximation, though reasonable and common.

** W.B. Hooper, "Calculate Head Loss Caused by Change in Pipe Size," Chemical Engineering, Nov 7, pp 89-92, 1988.

For Re > 2500:

$$K = \left(2.72 - \frac{AORI}{AF}\left(\frac{4000}{Re}\right)\right)\left(1 - \frac{AORI}{AF}\right)\left(\left(\frac{AF}{AORI}\right)^2 - 1\right)\left(0.584 + \frac{0.0936}{ELLD^{1.5} + 0.225}\right)$$

Figure 3-5 presents the diagram corresponding to the above equations for two cases (two values of ELLD). Note that the underlying correlation is restricted to ELLD < 5.

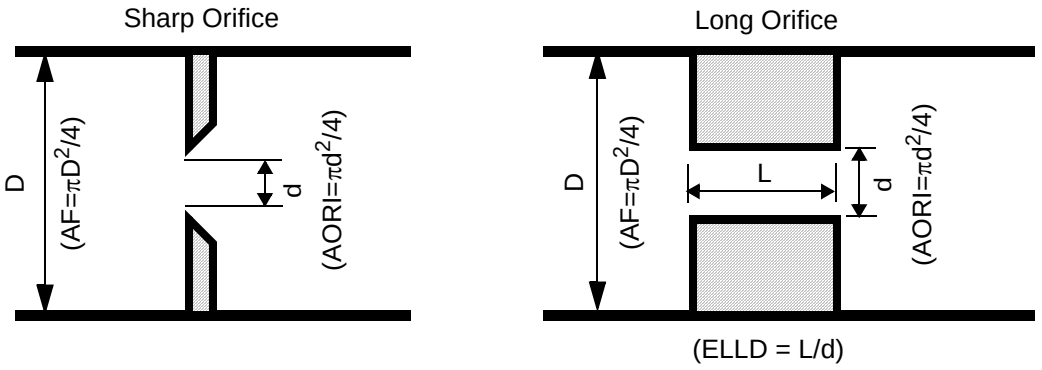


Figure 3-5 Basis for Built-in ORIFICE K-factor Correlation
(Circular Case Depicted)

The above correlation applies if MODO=1 (and in modified form if MODO=-1). If instead MODO=2 or MODO=-2, then the user is expected to input the coefficient of discharge, CDIS, defined in the following equation (SI units assumed):

$$FR = \frac{AORI \cdot (CDIS/C_v)\sqrt{2 \cdot \rho \cdot \Delta PL}}{1 - (CDIS/C_v) \cdot (AORI/AF)} \quad \left\{ = \frac{AFTH\sqrt{2 \cdot \rho \cdot \Delta PL}}{1 - (AFTH/AF)} \right\}$$

where C_v is the velocity coefficient which represents an adjustment for nonuniform velocity profiles (and is assumed to be equal to 0.98 for the ORIFICE connector).*

* The equation above to the right {in brackets} is for reference, showing that the key factor for orifices is the throat area AFTH. In a nutshell, *almost* one dynamic head is lost at the vena contracta represented by AFTH: $K_{AFTH} \approx 1$ if the head basis had been that hypothetical constriction, and so $K_{AORI} = K_t \approx 1/CDIS^2$ if the head is based on the physical restriction at AORI. Nonetheless, choking can occur for downstream pressures that are higher than the throat pressure (PLTH): some recovery of static pressure can occur.

If CDIS is input (|MODO|=2), the K-factor (FK) and the throat area (AFTH) are both calculated. The discharge coefficient CDIS is related to the K-factor FK via:

$$CDIS = C_v C_c$$

$$C_c = \frac{AFTH}{AORI} = \frac{1}{\left(\frac{AORI}{AF}\right) + \sqrt{K_t}} = \left(\frac{AF}{AORI}\right)\left(\frac{1}{1 + \sqrt{FK}}\right)$$

where C_c is the contraction coefficient (the ratio of the vena contracta area to the orifice area AORI). FLUINT's K-factor FK is based upon the flow area AF, which is to be contrasted with a K-factor based on the throat area AORI (K_t).

If MODO=3 or MODO=-3, the user can specify the K-factor (FK) in the above equations directly, and CDIS and AFTH are calculated. In the case of MODO=-3, the effective K-factor will be somewhat larger than the specified FK for compressible flows (see Section 3.5.12.1).

To close an ORIFICE connector, specify AORI as zero. To open or close it in a finite time span during a transient, see MODA=1 or MODA=2 in Section 3.5.12.2. To open or close it automatically as needed to maintain an upstream or downstream pressure, set MODA=-1 or MODA=-2 as also described in Section 3.5.12.2.

3.5.12.1 Compressible Flow Effects

If MODO is positive, then the above incompressible equations are applied without modification. If MODO is negative, then *an estimation* of compressible flow effects is included.* The flow is assumed to accelerate isentropically into the throat (vena contracta), with a resulting decrease in static pressure and density.

When expansion is included, the above equations apply except that the K-factor is augmented by the ratio FK = $K_{incompressible}$*(ρ_{up}/ρ_{avg}) where ρ_{avg} is the average density: ρ_{avg} = 0.5*(ρ_{up} + ρ_{down}).† This correction is roughly equivalent to the "Y expansion factor" used for perfect gas flows, but allows the more general case of real gases and multiple phases. For all-gas flows, as a check the empirically-based ASME Y expansion factor‡ is calculated and the largest of both corrections (lowest resulting flow) is used.

This correction causes a corresponding reduction in C_d and C_c (and therefore the throat area AFTH).

---

* Whether or not MODO is negative, the orifice is subject to choked flow calculations per the values of MCH and HCH. However, the election of positive or negative MODO can affect the estimation of the throat area, AFTH.

† "Duct Systems," Section 405.4, Fluid Flow Division of General Electric, November 1969.

‡ Y = 1 - (0.41 + 0.35*β^4)*(ΔP/Pup)/γ, where β^2 = AORI/AF, ΔP is the pressure drop (at least to the throat), γ is the specific heat ratio C_p/C_v. Y is the expansion correction factor applied to C_d (not to the K factor), so K = [(1-$K_{incompressible}^{1/2}$)/Y - 1]^2. If the flow is all gas (and does not change phase as the pressure drops in the orifice), then the large of this FK and the one based on density-only corrections is applied.

The downstream density ρ_{down} is not necessarily the same as the density of the downstream lump (which may be a plenum, or have nonzero QDOT, etc.). Therefore, ρ_{down} is calculated based on the fluid leaving the orifice: at the upstream enthalpy but downstream pressure. If the flow is choked, then to be consistent ρ_{down} is calculated at the downstream pressure for which choking is incipient: at a pressure somewhat higher than that of the actual downstream pressure.

Because of the likelihood of choking with an orifice, the user is strongly urged not to suspend choked flow calculations for ORIFICE connectors: *do not use MCH=0 for these paths.* Also, note that the default of MCH=-3 is overridden, choking can occur for downstream pressures that are higher than the throat pressure (PLTH): the static pressure typically recovers somewhat in the downstream section (seemingly in contradiction to the common approximation that "one dynamic head is lost irrecoverably at the vena contracta").

In the case of MODO=-3, expansion calculations are invoked yet the user has supplied a value of FK. In this special case, the input FK is treated as the incompressible loss, and the actual resistance will be augmented: its pressure drop will be higher than $\Delta P = FK*0.5*\rho_{up}*V_{up}^2$ (SI units assumed).

Similarly, if MODO=-2, expansion calculations (compressible flow corrections) will cause the actual or effective C_d to be slightly lower than the CDIS input by the user.

The above two paragraphs mean that MODO=-1 cannot be used to generate FK or CDIS values that, when input using MODO=|2| or |3|, will generate equivalent results. In fact, both the FK and CDIS values would need to be generated using MODO=+1 and applied using MODO=+2 or +3 for such "reversibility" to be verified by excluding compressibility corrections in both runs.

3.5.12.2 Controllable Orifices

If detailed valve performance data is missing, an ORIFICE device is often a good model of a partially closed valve (needle valve, pressure or backpressure regulating valve, even a fast-acting solenoid valve in mid stroke). This section describes such "valve as an orifice" usage.

By default, the orifice hole size AORI is user-defined, and can be updated during the run (though it can never be greater than or equal to the flow area, AF). This corresponds to the MODA=0 mode. When MODA (AORI mode) is nonzero, the program assumes control of AORI, adjusting it as needed to perform first-order simulations of an opening or closing valve, a back-pressure (upstream pressure) regulator, or a (downstream) pressure regulator.

MODA . . . . . 0 = no control (default). AORI is user specified, and limits are ignored.
1 = *opening*, AORI grows to maximum (AORI_H)
2 = *closing*, AORI shrinks to minimum (AORI_L)
-1 = adjusts AORI to maintain *upstream* pressure (to set point PSET)
-2 = adjusts AORI to maintain *downstream* pressure (to set point PSET)

To constrain the possible range of AORI values, lower and upper limits (AORI_L and AORI_H) may be set: the positions of a fully closed or fully opened valve. AORI_L defaults to zero (fully closed) and AORI_H defaults to 90% of AF (*nearly* fully opened, since AORI=AF is illegal).

This range of possible hole areas (AORI_L ≤ AORI ≤ AORI_H) is used to set the maximum allowable change in AORI either per steady-state iteration, or per transient time step.

In steady state (STEADY) solutions, the control parameter AORI_C does not allow AORI to change more than 10% of the allowed range by default. For example, when MODA=2 (signaling a valve to close), if AORI starts at AORI_H it will take at least 10 solution steps (LOOPCT or time steps) to close the valve. AORI_C may be set to 1.0 to allow the full range of motion per step, though this setting is not advised for steady-state solutions.

In transients (TRANSIENT, as well as STDSTL), an additional constraint may be placed on AORI changes: a maximum rate of change. This is regulated using AORI_T: the time constant for the valve. AORI_T is defined as the time it takes to move from the fully closed position (AORI_L) to the fully open position (AORI_H), or the reverse. AORI_T defaults to 10 milliseconds (i.e., a fast-acting valve, such as a solenoid valve). AORI_T can be set to zero, disabling any time lag (which can lead to small time steps and oscillations).

For pressure regulating modes (MODA=-1 or MODA=-2), by default FLUINT will use the current state (e.g., choked, or current resistance, etc.) to try to better estimate a new AORI. Such adjustments aren't perfect since AORI is assumed constant during the solution. The pre-solution adjustments therefore can't foresee all system reactions to this AORI change or to other changes.

This anticipation of best-estimate AORI, rather than a simpler "make the maximum change allowed by AORI_C and AORI_T" approach (which is the basis of MODA=1 and MODA=2), allows the program to converge on a steady-state answer, and to avoid oscillations about a set point during transients. However, this predictive method can also cause an additional lag beyond that prescribed by AORI_C or AORI_T. To avoid this additional lag in transients, a negative value of AORI_T may be used as a signal, with the absolute value (|AORI_T|) still employed as a rate change limit for AORI. When using negative values of AORI_T to disable the internal prediction methods, small magnitudes of AORI_T often lead to oscillations about the set point.

Using nonzero MODA, the time step may be reduced ("ORIFICE OPENING RATE LIMIT" or "ORIFICE CLOSING RATE LIMIT") but small time steps won't cause unrealistically fast changes as long as AORI_T is not zero.

Note that AORI_C is active in both steady and transient solutions, whereas AORI_T is active in transients.

Also note that use of MODO values other than 1 or -1 with a controllable orifice implies that the user must be able to update their supplied values of K or C_d values as the AORI value changes. Since this is either unlikely or difficult, caution should be used with nonzero MODA values in combination with MODO values other than 1 or -1.

3.5.12.3 FLOW DATA Format

ORIFICE Format in CONN Subblocks:

```
... DEV=ORIFICE, AORI=R [,AF=R] [,AFTH=R]
      [CDIS=R] [,FK=R] [,ELLD=R] [,MODO=I]
      [,MODA=I] [,PSET=R]
      [,AORI_L=R] [,AORI_H=R]
      [,AORI_C=R] [,AORI_T=R]
```

where:

AORI . . . . . Actual flow area of orifice aperture (AORI < AF)

AF. . . . . Upstream/downstream flow area (basis of velocity)

AFTH . . . . . throat flow area for optional choking calculations (may be initialized, but otherwise *this is an output* of the ORIFICE model, with AFTH < AORI)

ELLD . . . . . L/D ratio for long orifice (optional input if |MODO|=1)

CDIS . . . . . Discharge coefficient (input only if |MODO|=2)

FK. . . . . K-factor (input only if |MODO|=3)

MODO . . . . . Orifice solution mode:

- 1 . . . . . Use built-in sharp or long orifice correlation
- 2 . . . . . User input CDIS
- 3 . . . . . User input FK

Negative MODO signals the use of a compressible flow correction.

MODA . . . . . AORI control mode (integer)

- 0 . . . . . no control (default)
- 1 . . . . . opening, to maximum (AORI_H)
- 2 . . . . . closing, to minimum (AORI_L)
- -1. . . . . controlling defined upstream PL to PSET
- -2. . . . . controlling defined downstream PL to PSET

PSET . . . . . pressure set point for negative MODA (pressure regulator mode) in user's PL units (static or total according to lump controlled).

AORI_L . . . . lowest value of AORI allowed. Cannot be less than 1.0e-15 unless zero. Must be less than AORI_H

AORI_H . . . . highest value of AORI allowed. Cannot be greater than AF, nor less than AORI_L.

AORI_C . . . . maximum change in AORI from AORI_L to AORI_H that can occur in one time step or one steady state iteration. May be 1.0 (100%) for no added change lag.

AORI_T . . . . time constant for fastest motion from AORI_L to AORI_H or reverse. May be zero for no added time lag.

Negative signals fastest rate be used (subject to |AORI_T|) without any lag due to predictive methods. When using negative AORI_T to disable the lags of the internal prediction methods, consider large magnitudes of AORI_T for stability if the actual valve time constant is not known.

defaults:

AF . . . . . AF from PA DEF subblock, or if no AF defined or if negative, calculated using DH from PA DEF subblock assuming circular cross section
AFTH. . . . . (calculated by ORIFICE model)
ELLD. 0.0
MODO. . . . . -1 (built-in correlation plus compressible flow expansion)
MODA. . . . . 0 (no control: user-defined AORI, and AORI_L and AORI_H are ignored)
AORI_L . . . . . 0.0 (closed)
AORI_H . . . . . 0.9*AF (mostly open)
AORI_C 0.1 (10%)
AORI_T . . . . . 10ms (0.01 seconds, or 2.77778e-6 hours)

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

Caution: This device model is *not* validated for use in flashing flows *in long orifices* (ELLD>0), such as orifice tubes used as expansion devices in vapor compression cycles or Joule-Thomson (JT) devices. In any two-phase flow model with short orifices, nonequilibrium expansion (negative MCH, specifically the default of MCH=-3) is recommended, as is adequate consideration for the uncertainties involved in such calculations. Effects such as hysteresis and persistence of superheated liquid are not included in the ORIFICE connector model.

Caution: Use of MODO values other than 1 or -1 with a controllable orifice (nonzero MODA) implies that the user must be able to update their supplied values of K or C_d values as the AORI value changes. Since this is either unlikely or difficult, caution should be used with nonzero MODA values in combination with MODO values other than 1 or -1.

Guidance: If liquid enters the orifice and flashes in the throat, or if vapor enters and a fog forms in the throat, or if volatile two-phase fluid enters the orifice, use of the default MCH=-3 is strongly recommended.

Examples:

```
PA CONN,401,12,25, DEV=ORIFICE, AORI = 0.2*AF#this
PA CONN,1,1,2, MCH = -3, DEV=ORIFICE, AF=0.25*pi*Diam^2
      MODO = -2, CDIS = 0.62, AORI=0.25*pi*(beta*Diam)^2
PA CONN, 33, 34, 3334, DEV = ORIFICE
      AF = Flow_Area, AORI = 0.0
      MODA = (nsol == 2)? 1 : 0       $ open when transient starts
      AORI_T = 100.E-3/3600.0   $ 100ms (if UID=ENG)
PA CONN, 3001, 3301, 3302, DEV = ORIFICE, AF=Farea
      MODA = -2, PSET = 1.15e5  $ control PL of lump 3302
      AORI_L = 0.01*Farea     $ doesn't completely close
      AORI_T = 100.E-3*3600.0   $ 100ms (if UID=SI)
```

3.5.13 MFRSET Option

MFRSET means “set mass flow rate.” All an MFRSET connector does is maintain the given mass flow rate through the connector, independent of pressure gradient or fluid state. This device model has only one simple parameter: SMFR, the flow rate to be maintained. This parameter is accessible in user logic blocks for inspection or modification.

MFRSET is a powerful modeling tool. It can be used to simulate an ideal pump or flow control device. It can even be used to control boundary conditions for open systems, or it can represent a completely closed valve (i.e., SMFR=0.0).

However, it must be used carefully or fatal errors will result. For example, using two MFRSET connectors in the same series line with different flow rates will eventually (if not immediately) cause an error. The user should be careful to avoid similar overspecifications in more complicated networks with parallel passages: at least one flow passage should be allowed to adjust itself without an MFRSET prescribing the flow rate. For this reason, MFRSET connectors cannot be the only type of path attached to a junction.

Paths are intrinsically adiabatic and therefore isenthalpic. If the MFRSET connector is used to simulate a pump or compressor, and if the energy added to the network by this process is nonnegligible, the user should add the appropriate energy rate to the downstream lump.

MFRSET Format in CONN subblocks:

```
... DEV=MFRSET [,SMFR=R][,AF=R]
```

where:

SMFR. . . . . Mass flow rate to maintain
AF . . . . . flow area for kinetic energy calculations

defaults:

SMFR. . . . . FR from PA DEF subblock, or if no default FR defined, zero. Note: to be consistent with other connectors, an FR value defined earlier in this subblock is taken to be an initial value unrelated to SMFR.
AF . . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if AF is nonpositive then kinetic energy flow will be disrupted and energy will not conserved for high speed flows).

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Restriction: *Do not use more than one in series.* In general, do not put this device in any situation where maintaining the given flow rate contradicts a condition such as a closed valve or a primed capillary device.

Examples:

```
PA CONN,6,5,4, DEV=MFRSET
PA CONN,12,1,118, FR=12.0, DEV=MFRSET
PA CONN,12,1,118, DEV=MFRSET, SMFR=12.0
```

3.5.14 VFRSET Option

VFRSET means "set volumetric flow rate." It performs similar functions as MFRSET and should be used with similar caution. However, instead of maintaining a constant mass flow rate, VFRSET maintains a constant volumetric flow rate, independent of pressure gradient or fluid density. SVFR is the only parameter, and is used similar to SMFR (above).

VFRSET can be used in similar ways as MFRSET. However, it is a better approximation of a positive displacement pump, and has different behavior than MFRSET when used in compressible flows. Note that the mass flow rate through a VFRSET connector will drop considerably if there is any vapor in the inlet because of the density change.

Paths are intrinsically adiabatic and therefore isenthalpic. If the VFRSET connector is used to simulate a pump or compressor, and if the energy added to the network by this process is nonnegligible, the user should add the appropriate energy rate to the downstream lump.

VFRSET Format in CONN subblocks:

```
... DEV=VFRSET, SVFR=R [,AF=R]
```

where:

SVFR . . . . . Volumetric flow rate to maintain
AF. . . . . flow area for kinetic energy calculations

defaults:

SVFR . . . . . zero
AF. . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if AF is nonpositive then kinetic energy flow will be disrupted and energy will not conserved for high speed flows).

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

Restriction: *Do not put more than one in series unless compressible (at least part vapor) tanks will* always *be present between them and STEADY is not used.* In general, do not put this device in any situation where maintaining the given volumetric flow rate contradicts a condition such as a closed valve or a primed capillary device.

Examples:

```
PA CONN,118,119,120, DEV=VFRSET, SVFR=0.5
PA CONN,115,16,17, STAT=LS, DEV=VFRSET
      SVFR=12.0/DL#up
PA CONN,116,17,16, STAT=VS, DEV = VFRSET
      SVFR = -SVFR(115)
```

3.5.15 PUMP Option

A PUMP connector is a model of a fan or a centrifugal, axial, or mixed-flow pump: a pump whose performance varies as a function of volumetric flow rate for a given rotational speed.

In order to simulate such a pump, the user must specify pressure head (H) as a function of volumetric flow rate (G) and perhaps rotational speed (SPD). *Note that H is by default defined in terms of the density of the working fluid itself, not water, mercury, or some other reference fluid.* (Alternate units, such as "inches of water" or "mm of Hg" are also available, but are actually units of pressure, not head.) Thus, one foot of head is the weight of a column of one foot of the working fluid under one gravity.

Generally, the head produced by a pump within a typical system* is proportional to the pump speed squared. The head coefficient Ψ can be defined as Ψ = HCF*H/SPD**2, where HCF is a user-supplied conversion factor (defaults to 1.0). In common English engineering (US Customary) units ... so-called "lazy" units ... with H in feet and SPD in rpm, then HCF=g/D^2 where D is the rotor diameter and g is the acceleration of gravity (32.174). In that case, Ψ is not truly nondimensional, it is "quasi-nondimensional."

The volumetric flow rate G (more commonly referred to as "Q" in pump literature) is based on the fluid density at the inlet by default. (Alternate units are available, including mass flow rate units.) The flow rate produced by a pump within a typical system is proportional to the pump speed. The flow coefficient can be defined as Φ = GCF*G/SPD, where GCF is a user-supplied conversion factor (defaults to 1.0). In US Customary units, with G in gpm (US gallons per minute) and SPD in RPM, then either GCF=1/(A*D) or GCF=1/D^3 where D is the rotor diameter and A is the flow area. In that case, Φ is once again quasi-nondimensional.

In an ideal pump, "similarity rules" apply: the H scales perfectly with SPD^2 and the G scales linearly with SPD (with other variations fixed, such as pump rotor diameter). If these rules hold true, then a single curve of Ψ-Φ completely defines the map. (Figure 3-6)

Throughout the PUMP options, users may elect to use dimensional head and flow rates (H and G) to define performance parameters, or by using COEFS=YES they may instead elect to use nondimensional or quasi-nondimensional (depending on the values of HCF and GCF) head and flow rate *coefficients* (Ψ and Φ). The choice of whether or not to use coefficient-based performance parameters can vary from pump to pump (if more than one exist in the model), but otherwise the rule is all-or-nothing within a single pump: either use H and G, or else use Ψ and Φ, but the two forms cannot be mixed within a single device.

A specialized output routine, PUMPMAP (Section 7.11.8), is available for PUMP devices. PUMPMAP includes performance information as well an echo of key inputs.

* "Typical system" meaning a system that exhibits a pressure drop which scales with the square of velocity.

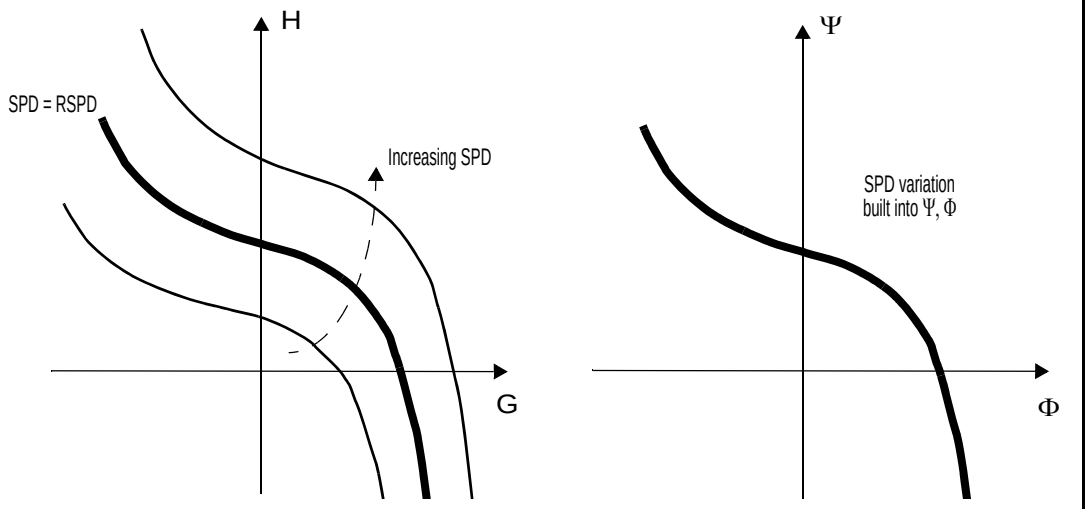


Figure 3-6 Similarity Rules for Dimensional and Nondimensional Pump Curves

3.5.15.1 Head-Flow Rate Curves and Maps

There are various ways to represent the head-flow performance map depending on information available:

1. If the pump similarity rules apply, the user can either:

   a. Specify a single Ψ-Φ curve (with COEFS=YES, and HCF and GCF specified along with the units of the underlying H and G values, per the right-hand side of Figure 3-6)

   b. Specify a single H-G curve along with a reference speed (RSPD) upon which that curve is based (along with the default COEFS=NO and the units of H and G specified, per the left-hand side of Figure 3-6). For pumps with no speed (SPD) variation, RSPD is optional.

   These methods will be collectively referred to as “the HG= method.”

2. Otherwise, if the similarity rules either don’t apply or don’t apply perfectly or globally, then the user can elect to input a full map: a series of Ψ-Φ or H-G curves at various speeds. (In the case of Ψ-Φ maps, an option exists to use a single curve of Φ with only Ψ being dependent on SPD.)

   When supplying a full map, the points in the curves are assumed to be corresponding at each speed, meaning that flow coefficients (whether or not coefficients are used in the map itself) are at least roughly equal. This requirement, which will be described in more detail later, is met by the output generated from most pump analysis codes.

   This method will be referred to as “the HLIST= method.”

3. If the Ψ-Φ or H-G maps exist as functions or are provided via calls to (perhaps COM-linked external) pump codes, then the current value of H and G (or Ψ and Φ) can provided directly. Although data is often perceived as H(G) ... head as a function of current flow ... SINDA/FLUINT actually requires the reverse: G(H) or Φ(Ψ): flow as a function of current head. Furthermore, SINDA/FLUINT requires the slope at the current SPD and head: dG/dH or dΦ/dΨ, which cannot be zero.

   In other words, the user can omit pump arrays and can instead manipulate the pump parameters directly in logic and/or expressions. The variables for H and G (or Ψ and Φ if COEFS=YES) are HPMP and GPMP. The code provides the current value of HPMP (head or head coefficient), and the user must supply the corresponding values for GPMP (flow rate or flow coefficient) and DGDH (the slope).

   This method will be referred to as “the GPMP= method.”

   When using the GPMP= method, automatic corrections such as two-phase, cavitation, and viscosity are omitted: they are assumed to have already been taken into account by the user in the provided values of GPMP and DGDH and the efficiency EFFP (see Section 3.5.15.2).

Arrays (method #1 or #2 above, the HG= or the HLIST= method) should have large enough range to cover all anticipated values. This includes operation with negative H or G. However, under the assumption that any answer is better than none, FLUINT will extrapolate from the available information to cover any excursions outside of the defined range and will signal such an excursion with an indicator flag, LIMP (a translatable path output variable):

LIMP = 0: nominal (within range)
LIMP = -1 or -2: head too low or too high, respectively
LIMP = 1 or 2: flow too low or too high, respectively
LIMP = -11 or -12: speed too low or too high, respectively

G or Φ must increase monotonically, but H or Ψ need not decrease monotonically although zero slopes (dH/dG or dΨ/dΦ) are not allowed. At least two data pairs are needed to define the simplest pump curve: a straight line.

Zero Speed Pumps: For all options, zero speed is problematic, and negative speeds can be problematic for coefficient-based approaches. In almost all cases, zero speed (SPD=0.0) is equivalent to a shut valve to be consistent. The one exception is when COEFS=NO and the HLIST= method is used, in which case a zero-speed curve (as well as negative speed curves) may be input. Otherwise, avoid zero and negligibly small speeds. For example, if pump start-up to 10,000 rpm is being modeled, consider starting at 100 rpm. Also, consider using LOSS or CTLVLV elements in parallel to represent the stopped condition, with logic or expressions employed to alternate between the PUMP model and the LOSS or CTLVLV model.

3.5.15.2 Defining Pump Efficiencies

Pump efficiencies are optional but should be specified or at least estimated (typical values are between 0.6 and 0.8, for example). Efficiencies are used to calculate heating (Section 3.5.15.3) and torque (Section 3.5.15.4). There are three ways to specify pump hydraulic efficiency, η:

1. as a single value, EFFP (updated in logic or expressions)

   This method will be referred to as “the EFFP= method.”

2. as an interpolated array of efficiency vs. G (or Φ if COEFS=YES).

   This method will be referred to as “the AEFFP= method.”

3. if a full map of H-G or Ψ-Φ has been provided via the HLIST= method, then a full map of efficiencies can be provided in parallel (e.g, full maps of η-G or η-Φ at the same operating points as the head-flow relationships).

   This method will be referred to as “the ELIST= method.”

Significant interrelationships exist between efficiencies and other inputs. First, efficiencies might be modified per the cavitation correction (CCE, see Section 3.5.15.5) and/or per the viscosity correction (EFFVF, see Section 3.5.15.6) if H-G relationships have been specified via arrays (i.e., either the HG= or the HLIST= method), and not using the GPMP= method. In such cases, the final EFFP value will be calculated using the arrays (i.e., methods #2 or #3 above, the AEFFP= or the ELIST= method) then modified by multiplying by CCE and EFFVF. If, on the other hand, EFFP is supplied directly (method #1 above, the EFFP= method), then any corrections are assumed to have been applied by the user. Similarly, if the GPMP= method is used, no viscosity nor cavitation corrections will be made to the user-supplied efficiency.

Furthermore, the viscosity correction itself will not be applied unless the efficiency has been defined using arrays (i.e., the AEFFP= or the ELIST= method) since the code must be able to calculate the most efficient point to perform those corrections.

If the AEFFP= method has been used for efficiencies and a reference speed, RSPD, has been specified (in which case the HG= method has been used for the head-flow relationship and COEFS=NO), then the G values in the AEFFP array will be scaled with |SPD/RSPD| to be consistent with the scaling of the G values in the HG array. This means that a single array of efficiencies versus flow will be assumed to apply at the design point, with the peak efficiency occurring at the design flow rate times |SPD/RSPD| for off-design speeds.

Note that pump efficiency, EFFP, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.).

3.5.15.3 QTMK and QTM: Turbomachinery Heat

Even for a perfectly efficient* pump (EFFP=1.0 by default), a heat load QTMK (a translatable path output variable) will be added to the QTM (a translatable lump output variable) of the lump currently downstream of the pump. QTM is one component (along with QL and the sums of tie QTIEs and ftie QFs) of the total lump power input QDOT. This path heat load, QTMK, is proportional to |GPMP|*HPMP/EFFP, taking into account various unit conversion factors.†

Each lump may have only one unique QTM. Therefore, if more than one PUMP or other turbomachine is placed in parallel, the lump QTM will represent to total or net power added by all turbomachinery currently exhausting to that lump.

Pump calculations override the rotating path (ROTR etc.) calculations for QTMK and TORQ (Section 3.5.15.4), and apply factors such as TCF and EFFP independently of analogous path rotation options; pump options generally take precedence over path rotation options.

3.5.15.4 Hydraulic Torque

The code calculates the hydraulic torque as:

TORQ = TCF*|GPMP|*HPMP/(EFFP*SPD)

where GPMP and HPMP are based on user-selected units,‡ and TCF is input by the user to perform whatever unit conversions are necessary. TORQ is a translatable path output variable. TCF defaults to unity. In US Customary units, if HPMP is in feet and GPMP is in gpm and SPD is in rpm, then to covert TORQ to lb_f-ft use:

```
TCF = DL#UP*0.1337/(2*pi)     $ DL#UP is the upstream density,
                              $ 0.1337 converts gallon to ft3
                              $ “pi” is a built-in constant
```

If COEFS=YES in the above case, with GCF=HCF=1.0, then:

```
TCF = DL#UP*0.1337 *SPD#THIS**3 /(2*pi)
```

In the more general case, for COEFS=YES (and HU=101 and GU=105 and SPD in rpm, for TORQ in lb_f-ft):

```
TCF = DL#UP*0.1337 *SPD#THIS**3 /(2*pi)/HCF#this/GCF#this
```

* This additional heat represents a departure from complete backward compatibility with Version 4.7 and earlier.

† For compressible fluids (including compressible liquids), the calculation is more generalized to include the change in enthalpy at constant entropy, which is reported in the PUMPMAP routine.

‡ Or unitless or quasi-unitless if COEFS=YES, in which case GPMP=Φ and HPMP=Ψ.

Hydraulic torque may be used along with co-solved ordinary differential equations (e.g., DIFFEQ1) to calculate the current speed of the pump during transients, or may be used to solve a torque balance equation for steady-states (e.g., for turbopumps), perhaps using ROOT_FIND. Note that TORQ is positive for energy into the fluid system, so the negative of TORQ should be set proportional to $d\omega/dt$, the rate of change of shaft speed.

Torque may also be used to calculate power input to the pump. If TORQ is in English units of lb_f-ft, for example, and speed is in rpm, then the horsepower can be calculated as:

$$HP = TORQ*SPD*2\pi/60.0/550.0$$

Pump power input may also be taken directly from the output lump's QTM (in Watts if UID=SI and in BTU/hr if UID=ENG), noting that other turbomachinery may be contributing to that QTM as well.

Note that pump torque coefficient, TCF, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.). Similarly, the TORQ calculations are independent of rotating path calculations.

3.5.15.5 Cavitation Detection and Modeling

Cavitation can be ignored, flagged, simulated, or both flagged and simulated.

Two inputs govern flagging: RNPSH (required or minimum net positive suction head, NPSH) and RNSS (required or maximum suction specific speed, Nss). If the working fluid is liquid at the inlet and has a defined saturation relationship (either being a two-phase fluid or a 9000 series with AT,PSAT data having been input), then the program will calculate the actual/available NPSH and Nss, designated as PUMP output variables* ANPSH and ANSS.

One of the subpurposes of the CHKFLASH routine (Section 7.11.1.10) is to detect and flag, via an output warning message, when either ANSPH<RNPSH or ANSS>RNSS. Otherwise users can perform their own checks in logic or expressions.

By default, RNPSH is -1.0E30 and RNSS is 1.0E30: no cavitation warnings will be produced.

Ideally, Nss should be unitless, so another conversion factor, SSCF, is required. Thus:

$$ANSS = SSCF * SPD*G^{1/2}/ANPSH^{3/4}$$

Note that ANSS is *not* defined using GPMP and HPMP (which might be Φ and Ψ), but rather by the inlet G and head (NPSH) values defined by the user-selected units, and so is not dependent on COEFS=YES or COEFS=NO. In "lazy" quasi-nondimensional English units with G in gpm, ANPSH in feet, and SPD in rpm, SSCF=1.0.

* Output variables are available for inspection and for use in calculations in either logic or data expressions, but may not be specified.

The user may specify efficiency and head correction factors for cavitation: CCE and CCH. These are applied to the predicted efficiency (corrected = uncorrected*CCE) and head (corrected = uncorrected*CCH) providing neither the EFFP= nor the GPMP= method was used (in which case the code assumes any cavitation corrections have already been made in those factors). If CCE and CCH have been directly specified by the user, they will be applied independent of NPSH and Nss: the user is assumed to be adjusting CCE and CCH as functions of whether or not cavitation is occurring, and by how much.

Alternatively, if neither the EFFP= nor the GPMP= method was used (i.e., if instead either the AEFFP= or the ELIST= method was used to define efficiency *and* either the HG= or the HLIST= method was used to define the head-flow relationships), then CCE and CCH may be provided to the code as arrays of values versus Nss.

Choking: Choking detection and modeling is disabled by default for a pump: MCH=0 (Section 3.18). Otherwise, a word of caution is required regarding pumps with low subcooling at their inlets: they might be flagged as choking if the inlet is stagnant (LSTAT=STAG) or if AFTH<AF. (Note that AFTH=AF by default, or AFTH=AFI if AFI and AFJ are specified instead of AF.) The isentropic expansion calculation might predict flashing at the throat, which can easily result in choking. Use care in defining AFTH and stagnant inlets if choking is enabled.

3.5.15.6 Viscosity Corrections

Users may elect to use Hydraulic Institute (HI, www.pumps.org) corrections* for viscous fluids, to scale those corrections as uncertainties, or to specify the corrections directly.

Three correction factors are available for viscosity: one for head (HVF), one for flow (GVF), and one for efficiency (EFFVF). If the parameter VCOR is positive, these three factors are expected to be user-supplied (input). If VCOR is negative, these factors are calculated by the program (output) as described below.

Note that viscosity corrections are never applied when the GPMP= method is used (e.g., when GPMP and DGDH are defined using expressions or logic instead of arrays), and that *viscosity corrections cannot be automatically calculated unless EFFP has been specified as an array* (i.e., using either the AEFFP= or the ELIST= method).

The reason that efficiency must be a function of flow rate and/or speed is that the HI correlation is based on the best operating point (H_{opt}, G_{opt}) for the current speed. In fact, the correlation only applies in the range ($0.6*G_{opt} < G < 1.2*G_{opt}$); outside of this range the end values are used so caution should be applied for very high or very low (including reversed) flows. The HI correlation

* Based on curve fits by Sorrell, T., Taylor, R.P., Hodge, B.K., “Estimating Pump Curves for High-Viscosity Fluids,” Heating/Piping/Air Conditioning, October 1990, p.75

also has large discontinuities around G_{opt} = 100 gpm (6.31 L/s), which might cause numerical problems in sensitive models. Other recommended (but not enforced) limits of this correlation include:

G: 1 gpm to 10000 gpm (0.063 L/s to 631 L/s)
H: 15 to 600 ft. water (4.6 to 183 m. water)

VCOR=-1.0 by default, which means that the HI correlation is applied directly. If VCOR is any other negative number, it is taken as a scaling factor for the degradation. For example, VCOR=-2.0 means "use twice the degradation in head, flow, and efficiency factors." Specifically:

$HVF = 1 - |VCOR|*(1-HVF_{HI})$
$GVF = 1 - |VCOR|*(1-GVF_{HI})$
$EFFVF = 1 - |VCOR|*(1-EFFVF_{HI})$

Thus, VCOR can be used to test the sensitivity of the results to the predictions of the HI viscosity correction correlation.

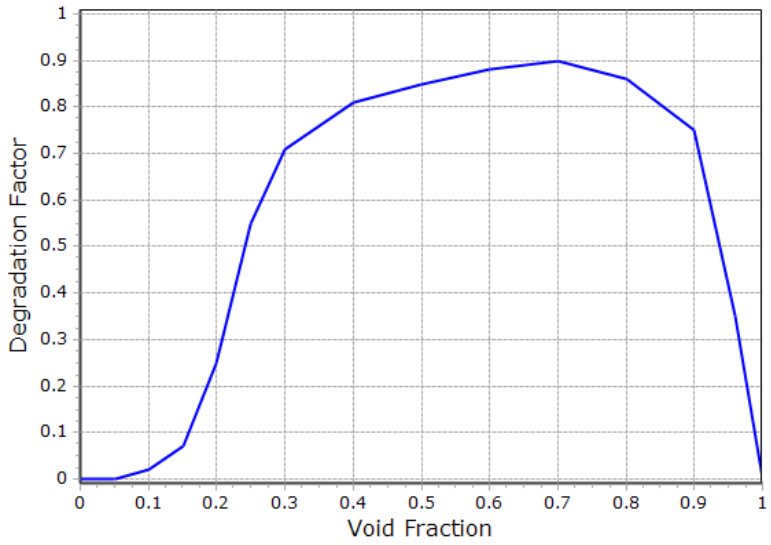
Caution: The HI correlation is not appropriate for all pumps and all fluids. It is especially suspect for cryogens, since it is normally applied to water and oil. When used in inappropriate situations, the degradations it predicts can vary rapidly as a function of slight changes to inputs and boundary conditions. The last section of the PUMPMAP output routine displays current degradations (HVF, GVF, and EFFVF). Set VCOR=0.0 to disable the viscosity correction.

3.5.15.7 Two-Phase Flow Corrections

Cavitation corrections, if specified, apply when liquid *or* two-phase flow is present at the inlet (Section 3.5.15.5). A separate and independent (albeit superimposed) head degradation factor is applied when two-phase flow is encountered at the inlet: $H_{corrected} = H_{uncorrected}*(1-DEG_{\alpha})$.

As a default, FLUINT will automatically use a built-in curve for DEG_{α}, as shown in Figure 3-7. This default curve represents a reasonable average degradation for several pumps reported in the literature*† and can be used as a first approximation unless specific pump characteristics are known. To disable this correction (perhaps in favor of cavitation corrections), specify a DEGVF array with zero degradation values (two such points are required as a minimum to define a line).

* B.E Boyack et al, *TRAC User's Guide,* LA-10590-M, Los Alamos National Laboratory, Nov. 1985.
† J.J. Beyer, *FLASH6: A Fortran IV Computer Program for Reactor Plant Loss-of-Coolant Accident Analysis (LW-BR Development Program),* WAPD-TM-1249 (Westinghouse/Bettis Laboratory), July 1976.



Void Fraction	Deg. Factor
0.0	0.0
0.05	0.0
0.10	0.02
0.15	0.07
0.20	0.25
0.25	0.55
0.30	0.71
0.40	0.81
0.50	0.85
0.60	0.88
0.70	0.90
0.80	0.86
0.90	0.75
0.96	0.35
1.0	0.0

Figure 3-7 Default Pump Degradation Curve for Two-Phase at Inlet

3.5.15.8 FLOW DATA Format

PUMP Format in CONN subblocks:

```
... DEV=PUMP [,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]
```

where:

GU . . . . . Flow rate unit identifier: see Table 3-5. These units will apply to G values, but if COEFS=YES then GPMP and all array flow values will be nondimensional or at least quasi-nondimensional according to GPMP = Φ = GCF*G/SPD (where G is given by GU units).

HU . . . . . Head unit identifier: see Table 3-6.* These units will apply to H values, but if COEFS=YES then HPMP and all array head values will be nondimensional or at least quasi-nondimensional according to HPMP = Ψ = HCF*H/SPD^2 (where H is given by HU units), and DGDH is in units of GPMP/HPMP.

COEFS . . . . . If COEFS=YES, then H is interpreted as head coefficient Ψ instead of head or pressure drop, and G is interpreted as flow coefficient Φ instead of mass or volumetric flow rate. This has an effect not only on the H/Ψ and G/Φ values, but also on HPMP, GPMP, DGDH, and TCF (but *not* SSCF, ANPSH, RNPSH, ANSS, nor RNSS).

GCF . . . . . If COEFS=YES, this is the conversion factor applied to G/SPD to yield Φ: Φ = GCF*G/SPD. For US Customary units, with G in gpm and SPD in rpm, then either GCF = 1/(A*D) or perhaps GCF=1/D^3 where D is the rotor diameter and A is the flow area.

HCF . . . . . If COEFS=YES, this is the conversion factor applied to H/SPD^2 to yield Ψ: Ψ = HCF*H/SPD^2. For US Customary units, with H in feet and SPD in rpm, then HCF = g/D^2 where D is the rotor diameter and g is the acceleration of gravity.

ISTP. . . . . If ISTP=YES (or equivalently, ISTP=TT), then head is defined on the basis of total or stagnation pressures, and either AF or both AFI and AFJ are required inputs. Otherwise (ISTP=NO or equivalently, ISTP=SS), static inlet/outlet pressures are used without accounting for dynamic head. ISTP=TS (total inlet, static outlet) and ISTP=ST (static inlet, total outlet) are also available.

HG . . . . . ID of doublet array (in this submodel) containing a single pump head-flow curve: head (units according to HU) as a function of G (flow rate: units according to GU). *G must increase monotonically, and no two adjacent H values may be equal (i.e., the slope dH/dG cannot be zero).* The array pointed to must contain real values or expressions of the form G1,H1, G2,H2 ... or if COEFS=YES then Φ1,Ψ1, Φ2,Ψ2 ...
Extrapolations will be flagged as nonzero LIMP values.
Note that this “HG= method” is just one way of specifying the H-G relationship. See also SPEEDS/HLIST/GLIST for the “HLIST= method,” and GPMP/DGDH for the “GPMP= method.”

SPD . . . . . current/initial pump speed, in consistent user units. Avoid very small or zero values (the default!) unless the model/inputs are specifically designed

* This table contains the same options used for TABULAR connectors (Section 3.5.16). The last two HU options are only available for TABULAR connectors, and not for PUMP connectors.

for this case. In most cases, zero SPD is equivalent to a closed valve for consistency.

RSPD . . . . . reference pump speed, in consistent user units (same as SPD). This option is only applicable if the HG= method is used *and* COEFS=NO. If the AEFFP method is used as well, its G values will be scaled with |SPD/RSPD|.

GPMP . . . . . for pumps whose head-flow is defined by expressions and/or user logic (i.e., by the "GPMP= method" and not by arrays via either the HG= or the HLIST= method), the flow rate or flow coefficient at the current head, HPMP, in units consistent with COEFS and GU. If H-G arrays are input via the HG= or the HLIST= method, then this becomes an output.*

DGDH . . . . . for pumps defined by the GPMP= method (i.e., no arrays used), the slope (flow rate or flow coefficient to head or head coefficient) at the current head, HPMP, in units consistent with COEFS, GU, and HU. Must not be zero, and negative values are legal but can cause instabilities in some models if persistent. If H-G arrays are input via the HG= or the HLIST= method, then this becomes an output.

SPEEDS . . . . A full map alternative to the single-curve "HG= method," SPEEDS is the array ID containing a list of speeds for which head and flow rate curves (and possibly efficiencies) will be specified via the HLIST, GLIST, and optionally the ELIST arrays (see "Full Map Example" below). Like HG, SPEEDS is an array ID, and the entries in that array must be real numbers or expressions in the same units as SPD.

HLIST. . . . . The ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing head values that correspond to each of the above N speeds. In other words, the first array ID in the HLIST array should contain a series of head values at the first speed in the SPEEDS array. Those values should correspond to the first array of GLIST (and optionally, ELIST): all such arrays must be the same length. The array values themselves should be in units consistent with COEFS and HU.

GLIST. . . . . The ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing flow rate values that correspond to each of the above N speeds. In other words, the first array ID in the GLIST array should contain a series of flow rate values at the first speed in the SPEEDS array. Those values should correspond to the first array of HLIST (and optionally, ELIST): all such arrays must be the same length. The array values themselves should be in units consistent with COEFS and GU. If COEFS=YES and only one array of flow coefficient points is needed, specify that array repeatedly for the number of speeds supplied. For example: GLIST = 102, and in ARRAY DATA: 102 = 34, 34, 34, 34 ....

* Note that the calculated value of GPMP will differ from a value calculated using the current FR except for a converged steady-state. FR is the current flow rate, whereas GPMP (updated after FLOGIC 0) is the current flow that the pump *would* be at if the head were constant at the current HPMP value. The value of GPMP can be compared directly with the output variable GFR, the value of the current flow rate in the same units as GPMP.

ELIST . . . . . The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing pump hydraulic efficiencies that correspond to the above N speeds. In other words, the first array ID in the ELIST array should contain a series of efficiency values (between zero and one) at the first speed in SPEEDS. Those values should correspond to the first array of HLIST and the first array of GLIST. Edge values are used if the provided range is exceeded; no extrapolations are made for this variable.

EFFP. . . . . Overall pump efficiency (real, between 0.0 and 1.0). This is an output if either ELIST or AEFFP arrays are provided.

AEFFP . . . . . ID of the doublet array containing the efficiency as a function of flow G (or flow coefficient Φ if COEFS=YES). End values will be used if the provided range is exceeded; no extrapolations are made for this variable. If RSPD has been specified (above), then G will be scaled with |SPD/RSPD|.

VCOR. . . . . Viscosity correction control and scaling factor. If zero, or if neither ELIST nor AEFFP have been input, then no corrections will be performed (HVF=GVF=EFFVF=1.0). If negative, the viscosity degradation will be scaled with |VCOR|. If positive, HVF, GVF, and EFFVF will be provided by the user.

HVF . . . . . Head correction factor for viscosity (input if VCOR>0, output if VCOR<=0). Apparent head will be divided by this value if the head-flow relationship has been specified using arrays using either the HG= or the HLIST= method.

GVF . . . . . Flow correction factor for viscosity (input if VCOR>0, output if VCOR<=0). Flow will be multiplied by this value if the head-flow relationship has been specified using arrays using either the HG= or the HLIST= method.

EFFVF . . . . . Efficiency correction factor for viscosity (input if VCOR>0.). EFFP will be multiplied by this value if the head-flow relationship has been specified using arrays using either the HG= or the HLIST= method.

TCF . . . . . Correction factor for the PUMP output variable TORQ, such that TORQ = TCF*GPMP*HPMP/(EFFP*SPD) where GPMP and HPMP are in units defined by COEFS, GU, and HU.

SSCF. . . . . Nss (suction specific speed) units correction factor, such that ANSS= SSCF*SPD*$G^{1/2}$/$ANPSH^{3/4}$ where the flow G and the available NPSH (output variable ANPSH) are defined by GU and HU but *not* by COEFS: they are G and H, not Φ and Ψ.

RNPSH . . . . . Required NPSH in units defined by GU but *not* by COEFS (it is H not Ψ). Can be compared with output variable ANPSH, perhaps using CHKFLASH.

RNSS. . . . . Required Nss (suction specific speed) in units defined by SSCF. Can be compared with output variable ANSS, perhaps using CHKFLASH.

CCE . . . . . Efficiency correction factor for cavitation (output if ACCE is specified). Efficiency will be multiplied by this value if the head-flow relationship has been specified using arrays using either the HG= or the HLIST= method.

ACCE . . . . . ID of doublet array containing the efficiency cavitation correction factor as a function of ANSS: ANSS1,CCE1, ANSS2,CCE2, ... with ANSS monotonically increasing. End values will be used if the provided range is exceeded; no extrapolations are made for this variable.

CCH . . . . . Head correction factor for cavitation (output if ACCH is specified). Apparent head will be divided by this value if H-G has been specified using arrays using either the HG= or the HLIST= method.

ACCH . . . . . ID of doublet array containing the head cavitation correction factor as a function of ANSS: ANSS1,CCH1, ANSS2,CCH2, ... with ANSS monotonically increasing. End values will be used if the provided range is exceeded; no extrapolations are made for this variable.

DEGVF . . . . . ID of doublet array containing the head degradation factor as a function of inlet void fraction. $0.0 \leq DEG < 1.0$; VF1,DEG1, VF2,DEG2 ... with void fraction monotonically increasing.

AF . . . . . flow area for kinetic energy, choking, and NPSH calculations, and for calculating dynamic head if ISTP=YES. This input is not required if AFI and AFJ are specified instead.

AFTH . . . . . throat area for choking calculations

AFI . . . . . inlet (suction) flow area, if not constant. If AFI is specified, AFJ must also be specified, and AF will be calculated automatically.

AFJ . . . . . outlet (discharge) flow area, if not constant

UPF . . . . . weighting factor for properties: 1.0 = base H and G on inlet (suction) density, 0.0 = base H and G on outlet (discharge) density. Note that ANSS etc. are always based on current inlet (suction) properties

defaults:

GU . . . . . 0 (m^3/s if UID=SI, else ft^3/hr of working fluid if UID=ENG)

HU . . . . . 0 (meters of working fluid if UID=SI, else feet of working fluid if UID=ENG)

COEFS . . . . . NO (head and flow coefficients are not used)

GCF 1.0

HCF 1.0

ISTP . . . . . NO (pressures based on static values), same as ISTP=SS.

SPD . . . . . 0.0 (zero speed usually means the pump acts like a closed valve)

RSPD . . . . . none

GPMP . . . . . none (calculated if H-G arrays input using either the HG= or the HLIST= method)

DGDH . . . . . none (calculated if H-G arrays input using either the HG= or the HLIST= method)

EFFP . . . . . 1.0 (calculated if AEFFP or ELIST are used; independent of PA DEF defaults)

VCOR . . . . . -1.0 (use HI correlation if efficiency arrays have been input, and the GPMP= method has *not* been used)

HVF 1.0
GVF 1.0
EFFVF 1.0
TCF . . . . . 1.0 (independent of PA DEF defaults)
SSCF. 1.0
RNPSH . . . . . -1.0E30
RNSS. 1.0E30
CCE . . . . . 1.0 (calculated if ACCE is used)
CCH . . . . . 1.0 (calculated if ACCH is used)
DEGVF . . . . . Uses a “standard” curve that is stored internally (Figure 3-7)
AF . . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if neither AF nor AFI/AFJ have been specified, then no kinetic energy can be extracted nor can choking calculations applied, nor can ISTP=YES be used, and NPSH will be based on static instead of total pressures at the inlet).
AFTH. . . . . AF, or AFI if both AFI and AFJ are used instead of AF
AFI . . . . . -1.0 (use AF)
AFJ . . . . . -1.0 (use AF)
UPF . . . . . 1.0 (inlet properties). Note that this default UPF value (unlike that for tubes and STUBE connectors) does not depend on prior PA DEF blocks.
MCH . . . . . 0 (no choking: contained within maps)

Simple Examples:

```
PA CONN, 10,1,2, FR=10.0, DEV=PUMP, HG=1, DEGVF=2
PA CONN,1,111,112, STAT=DLS, DEV=PUMP, HG=114
      HU=2, GU=4 $ Units of cm H2O and L/s
```

A full map example is provided below.

Table 3-5 G (“Flow rate”) Units for PUMP, TABULAR, TURBINE, COMPRESS, COMPPD Connectors

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	

Table 3-5 G (“Flow rate”) Units for PUMP, TABULAR, TURBINE, COMPRESS, COMPPD Connectors

GU	Meaning	Cautions
10	mm^3/s	
11	mm^3/min	
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

Table 3-6 H (“Head”) Units for PUMP, TABULAR, TURBINE, COMPRESS, COMPPD Connectors

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 (3.5.16), TURBINE (3.5.17), COMPRESS (3.5.18), or COMPPD (3.5.19) only
202	Martin Beta Factor	TABULAR only (3.5.16)

Caution: Some of the “volumetric flow rate (G)” units are really mass flow rate units, and most of the “head (H)” units are not length, but rather pressure. Despite the name, “inches of water” has units of pressure, not length or head, for example. In any case where pressure or mass flow rate units are used instead of head and volumetric flow rate units, the relative independence of density will be lost: the pump curve will be less applicable to different fluids, pressures, temperatures, etc. than those for which the data is taken.

Caution: HPMP and GPMP represent a point on the current (or extrapolated) pump curve, whereas GPMP can differ from the current flow (as designated by GFR, an output variable corresponding to the current mass flow rate FR but in the same units as GPMP). If corrections are required that are based on the current flow rate, use GFR, not GPMP.

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Guidance: A summary of common head-flow rate input options is provided below, and summarized in Table 3-7:

1) *A single curve of H vs. G at one speed.* Specify the HG array and SPD and (if the speed is to vary) RSPD variables. Use of AEFFP is recommended, but a single value (EFFP) can be supplied if available (even though use of a single value disables viscosity corrections). If AEFFP is provided, the flows (Gs) in that array will be scaled with the ratio |SPD/RSPD| if RSPD is specified.

2) *A single curve of* Ψ *vs.* Φ. Specify the HG array and SPD (but *not* RSPD), and set COEFS=YES. Specify GCF and HCF values if applicable. Use of AEFFP is recommended, and this array should contain an doublet array of efficiencies versus Φ.

3) *A full map of H and G.* Specify the SPEEDS, HLIST, and GLIST lists along with the current/initial speed SPD. Use of a corresponding ELIST is best, but a single array of efficiencies vs. G (via AEFFP) is adequate.

4) *A full map of* Ψ *and* Φ. Specify the SPEEDS, HLIST, and GLIST lists along with the current/initial speed SPD, and set COEFS=YES. Specify GCF and HCF values if applicable. Use of a corresponding ELIST is best, but a single array of efficiencies vs. Φ (via AEFFP) is adequate.

5) *A map of* Ψ*, but all the* Φ *values are the same.* Similar to the above, but list the same array ID repeatedly inside the GLIST array.*

* For example: “2000 = 2001, 2001, 2001, 2001”

Table 3-7 Summary of PUMP Input Options

Head-Flow Method	COEFS	Efficiency Method	Other notes
Single Point/Slope (GPMP= and DGDH=)	NO (GPMP=G)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ...	Viscosity, cavitation corrections etc. are not automatic: they should be already contained within provided GPMP, DGDH, and EFFP values.
	YES (GPMP=Φ)	EFFP=η or AEFFP=$\Phi_1,\eta_1,\Phi_2,\eta_2$ ...	
Single Curve (HG=)	NO (G1, H1 ...) (RSPD OK)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ... (G scaled if RSPD input)	AEFFP recommended since viscosity and cavitation corrections are then applied automatically. EFFP= disables corrections, in which case CCE and VCOR/EFFVP should be specified.
	YES (Φ1, Ψ1 ...) (no RSPD)	EFFP=η or AEFFP=$\Phi_1,\eta_1,\Phi_2,\eta_2$ ...	
Full Map (HLIST=)	NO (G, H)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ... or ELIST = ...	ELIST recommended, but AEFFP adequate if COEFS=YES. EFFP= disables corrections (see above). Viscosity and cavitation corrections are otherwise applied automatically.
	YES (Φ, Ψ)	EFFP=η or AEFFP=$\Phi_1,\eta_1,\Phi_2,\eta_2$ ... or ELIST =	

3.5.15.9 Full Map Example and Discussion

This section contains an example of the “HLIST= method:” a full map of head, flow rate, and efficiencies as functions of speed, along with a short description of the mathematical treatments of these maps and the resulting requirements for their structure. In this example, 4 speeds are used, with 6 values (head, flow, and efficiency) at each speed: a 4x6 map.

For the full map example listed below, note that the PUMP connector lists for HLIST, GLIST, and ELIST are “lists of lists:” lists of array IDs that point to the arrays containing the actual data in 1D strings. In this example, there are four pump speeds for which data is available, as pointed to in the SPEEDS array (number 1000): 64515 rpm, 86020 rpm, 107525 rpm, and 129030 rpm. For the first speed (64515 rpm), the first flow rate array (number 2001, the first in GLIST array 2000) corresponds to the first head array (number 3001, the first in HLIST array 3000) and also to the first efficiency array 4001 (the first in ELIST array 4000).

While the efficiency array will be interpolated directly and linearly, and will not be extrapolated (i.e., edge values will be used past the range provided), the head and flow rate arrays will be treated differently since COEFS=NO by default. They will be interpolated and extrapolated using locally produced curve fits of the form $y = a*SPD^b$, where b is approximately 1.0 for the flows and approximately 2.0 for heads. The b exponents are *exactly* 1.0 and 2.0, respectively, *if* the similarity rules apply perfectly. Because of this method, the tables are assumed to respect an underlying structure: the flow rate points at each speed are expected to correspond to those same points at other speeds *with roughly constant flow coefficients* (whether or not the G values themselves represent flows or flow coefficients). For example, the second head/flow point at SPD=64515 is (H=16700.7, G=386.6660) whereas the second point at the next highest speed (86020) is (H=29713.7, G=515.5561). The flow coefficient is roughly constant between those two points since (386.6660/515.5561)*(86020/64515) = 1.0 (in other words, about unity). Such structure is assumed because

this is the manner in which outputs are produced for most popular pump analysis software, and corresponds to common data reduction techniques for actual pump performance. At high speeds where internal pump losses are negligible, these points roughly correspond to a constant “system curve” if the system’s losses were to exhibit a perfect velocity-squared loss relationship.

If COEFS=YES, then the head and flow rate arrays would be interpolated directly and linearly (and extrapolated as well). If the flow coefficients are exactly equal, a single G array can be repetitively input in the array identified by GLIST.

```
HEADER FLOW DATA, PUMP ...
...
pa conn,10,1,10,dev=pump
      spd = speed                   $ initial speed (slow but not zero)
      speeds = 1000                 $ list of speed values provided
      glist = 2000, GU=105          $ list of flow arrays; Q in GPM
      hlist = 3000, HU=101          $ list of head arrays; head in FT
            elist = 4000            $ list of efficiency array IDs
      ISTP = yes                    $ total to total pressure used
      af = diam**2                  $ flow area (constant)
      TCF = dl#up*0.1337/(2*pi)     $ to get lbf-ft, 0.1337 = ft3/gal
...
HEADER ARRAY DATA, PUMP
c
c 4 speeds, each speed has 6 pairs of H-G values given in separate arrays
c with 6 matching (optional) EFFP values per curve too
c
c speeds
  1000 = 64515., 86020., 107525., 129030.
c
c glist = 2000
c volumetric flows (gpm)
  2000 = 2001,2002,2003,2004        $ one for each speed
      2001 = 322.2218, 386.6660, 451.1088, 515.5561, 579.9990, 644.4418
      2002 = 429.6290, 515.5561, 601.4814, 687.407, 773.3319, 859.2572
      2003 = 537.0341, 644.4418, 751.8495, 859.2572, 966.664896, 1074.073
      2004 = 644.4418, 773.3320, 902.2221, 1031.108, 1159.998, 1288.888
c
c hlist = 3000
c head (ft)
 3000 = 3001,3002,3003,3004         $ one for each speed
      3001 = 16656.6, 16700.7, 16094., 15156., 13898.4, 12487.1
      3002 = 29632.9, 29713.7, 28634.3, 26965.8, 24728.4, 22216.7
      3003 = 46325.4, 46457., 44769.9, 42162.4, 38664.9, 34736.9
      3004 = 66735.6, 66933.7, 64504.3, 60749.4, 55711.6, 50051.7
c
c elist = 4000
c efficiency
 4000 = 4001,4002,4003,4004         $ one for each speed
      4001 = 0.688793, 0.734547, 0.754449, 0.74962, 0.7156, 0.652818
      4002 = 0.6918, 0.737381, 0.757076, 0.75202, 0.717727, 0.654593
      4003 = 0.694251, 0.739687, 0.7592, 0.75396, 0.719444, 0.656025
      4004 = 0.69635, 0.741665, 0.76102, 0.75562, 0.720908, 0.657248
```

3.5.16 TABULAR Option

The TABULAR connector device allows users to specify flow rate versus head (or pressure drop) relationships in tabular (array) formats. The flow rate "G" can be either volumetric or mass flow rates, perhaps corrected or based on a reference state. The head "H" can be either a true head (in units of length, and therefore dependent on the current fluid density) or can have units of pressure drop or pressure ratio. The same unit options available for PUMP connectors (Section 3.5.15) apply to TABULAR connectors as well, although TABULAR adds a few options that are not available in PUMP connectors (see Table 3-5 and Table 3-6).

In a PUMP connector, the positive head (H) is understood to be a pumping force in the flow-wise (positive flow rate) direction, whereas in the TABULAR connector the head H is understood to be a retarding force in the positive flow direction: a head loss. In both cases, negative values of H are unusual but by no means illegal.*

Unlike a PUMP connector, for a TABULAR connector more generalized meanings of G and H are available that are better suited for compressible flows. For example, H might be absolute pressure *ratio* instead of pressure *difference*, and G might be corrected or equivalent mass flow (containing corrections for inlet temperature and pressure).

Also, H might be the Martin β factor,† a useful relationship for sequential constrictive losses such as labyrinth seals (where ENLT is the number of such losses or labyrinth teeth):

$$\beta = \sqrt{\frac{1-(P_{out}/P_{in})^2}{ENLT - \ln(P_{out}/P_{in})}}$$

For each TABULAR connector device, the user may choose one of four ways to define the structure and format of the H vs. G relationship:

1. HG: A doublet list of H values as a function of G values: G_1, H_1, G_2, H_2, ...
2. GLIST and HLIST: Paired singlet lists of G (G_1, G_2, ...) and H (H_1, H_2, ...) values.
3. HGTABLE: A bivariate table of H as a function of G and some other user variable O. In other words, for each value of O a head-flow rate curve, H(G), exists. O is completely user defined. For example, it might be the open fraction for a valve, or the inlet temperature for a viscosity-dependent device or control valve. The current value of O is "OFAC," and may be specified in logic blocks or expressions. For example, "flow.ofac102" is the O factor for TABULAR connector #102 in fluid submodel *flow.*
4. GHTABLE: A bivariate table of G as a function of H and some other user variable O. In other words, for each value of O a head-flow rate curve, G(H), exists. This option is an inversion of the HGTABLE option.

* Actually, H values must be positive for HU=201 or HU=202, the pressure ratio options.

† H. M. Martin, "Labyrinth Packings," Engineering, January 1908, pp. 35-36.

In all cases, adjacent H values cannot be equal. By default, the G value must monotonically increase, but H need not be monotonic. If instead H *is* monotonic, this can be specified using the MONOH=YES option described below, which then permits (but does not require) G to be nonmonotonic.

Also in all cases except one (noted below), an assumption of symmetry is made by default: H(G) for positive G is applied as -H(-G) if G is negative.* This allows the user to define one curve that applies in both directions if there is no difference in resistance between forward and reverse flows. When such symmetry is assumed, *the initial G value must be zero*, and *the initial H value corresponding to that initial G value must also be zero (or unity if H is a pressure ratio)*. For the bivariate HGTABLE or GHTABLE options, all initial H values must therefore be zero if symmetry is defaulted (or they must all be unity for the pressure ratio option).

Alternatively, if the device is directional (i.e., it behaves differently if flow rates reverse), then the user can state SYM=NO and supply a more complete curve or family of curves that include the negative flow rate regions. In this case, the initial G value will usually be negative, and the initial H value will usually but not necessarily be negative as well for a resistive element. Figure 3-8 provides an illustration of these options. Note that an asymmetric curve need not pass through the origin.

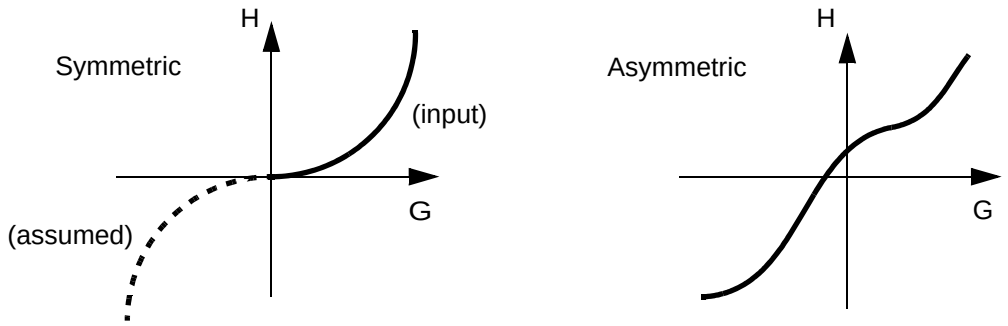


Figure 3-8 Graphical Examples of Symmetric and Asymmetric Head-Flow Rate Curves

* When symmetry applies, positive H represents a pressure loss in either flow direction. Symmetry for HU=201 or HU=202 is applied differently. For example, for HU=201 (pressure ratio), then it is applied as 1/H(-G) if G is negative or H<1, with analogous treatment for HU=202 (Martin β factor).

If H represents pressure ratio (HU=201 as described below), then asymmetry (SYM=NO) is the default in this case. When HU=201 without symmetry, then G values will often be negative for H<1 (defined outlet pressure higher than that at the defined inlet), and G will usually be zero for H=1 (equal pressures), as depicted in Figure 3-9.

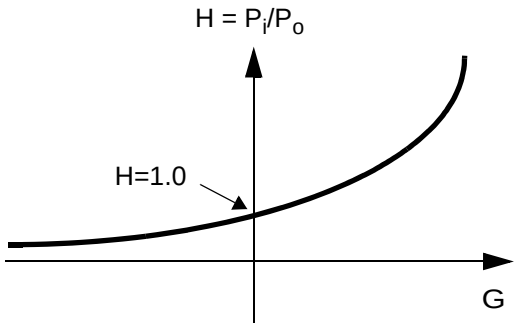


Figure 3-9 Graphical Example of Default Asymmetric Option for Pressure Ratios

With symmetry and pressure ratio (HU=201) used, then only ratios greater than 1.0 will be used: the first head value must be unity and the first flow value must be zero. Therefore, the code will calculate H as the maximum pressure over the minimum pressure, with the flow rates being negated if the pressure ratio is inverted (outlet over inlet) as depicted in Figure 3-10.

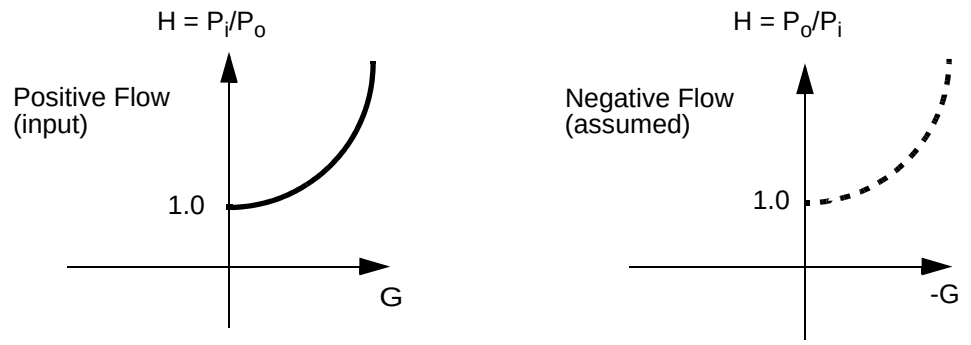


Figure 3-10 Graphical Examples of Symmetric for Pressure Ratio (HU=201)

If H represents the Martin β factor (HU=202, as described below), then symmetry *must* be used and the first H (β) value must always be zero (along with the first G value). Flow rates will be negated if the defined outlet pressure is greater than the defined inlet pressure, and the inverse pressure ratio will be used to evaluate β. The application of the Martin β factor to reverse pressure gradients and flows is depicted in Figure 3-11.

To close a TABULAR connector, independent of the subtype, specify a negative value of OFAC. Because of this convention, O values in the HGTABLE or GHTABLE should be nonnegative.

The flow rate value G provided in the tables may deviate from the units specified by GU, as described next.

First, G may be defined on an areal (flux) basis using the NAF flag. For example, NAF=1 means that the user-provided G values in the tables are actually:

$$\Phi = GCF*G/AF \qquad (NAF = 1)$$

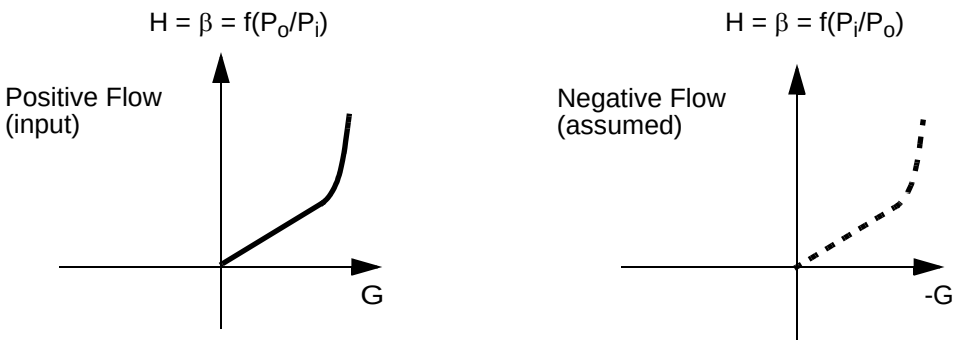


Figure 3-11 Graphical Example of Symmetric for Martin β Factor (HU=202)

where AF is the flow area and GCF is a unit conversion and scaling factor (defaulting to 1.0), and where G is in units specified by the units flag GU.

Second, the "G" values may represent a reference flow rate or corrected mass flow rate using the MREF flag. For example, if MREF=1 then the user-provided G values are actually understood to be:

$$\Phi = GCF*G*T^{1/2}/P \qquad (MREF = 1)$$

where T is the absolute inlet temperature and P is the absolute inlet pressure (in user units according to UID), and GCF is one again an available unit conversion and scaling factor that defaults to unity.

MREF can be used in combination with the NAF flag. For example, if MREF=1 and NAF=1, then the supplied "G" values are understood to be of the form:

$$\Phi = GCF*G*T^{1/2}/(P*AF) \qquad (MREF = 1, NAF = 1)$$

If MREF=2, then the provided "G" values are understood to include a reference state:

$$\Phi = GCF*G*(T/T_{ref})^{1/2}/(P/P_{ref}) \qquad (MREF = 2)$$

The above two MREF options are recommended (but not restricted) to gas flows, and nearly perfect gases as well. A third option, MREF=3, is based on reference or equivalent inlet conditions for turbines and compressors, and is *only* available for gas flows:*

$$\Phi = GCF*G*(\varepsilon_{ref}/\varepsilon)*(V_{cr}^2/V_{cr,ref}^2)^{1/2}/(P/P_{ref}) \qquad (MREF = 3)$$

* If MREF=3 or MREF=4, then the code will abort if 100% liquid is upstream, and will ignore liquid in the calculation of γ and R if a two-phase state is upstream.

where:

$$\varepsilon = \gamma\left\langle\frac{2}{\gamma+1}\right\rangle^{\gamma/(\gamma-1)}$$

$$V_{cr}^2 = \left\langle\frac{2\gamma}{\gamma+1}\right\rangle g_c RT$$

and ε_{ref} and $V_{cr,ref}$ are based on γ_{ref} and T_{ref}, and γ is the ratio of specific heats, $\gamma = C_p/C_v$. R is the gas constant, and g_c is the acceleration of gravity (applicable only to English units).

A fourth option, MREF=4, is similarly restricted to use in gas flows:

$$\Phi = GCF*G*(RT/g_c)^{1/2}/P \qquad (MREF = 4)$$

where R is the gas constant and g_c=1 if UID=SI or g_c=32.174 if UID=ENG.

Note that GCF is not applied unless either NAF or MREF is nonzero.

By default, all temperatures and pressures used for pressure drops, ratios, and MREF corrections are static (if an adjacent lump uses LSTAT=STAG, then its temperature and pressure is *both* static *and* total since the velocity is assumed zero). This choice can be regulated using the ISTP factor, which can be used to change the defined inlet and outlet temperature and pressures used throughout the TABULAR calculations. ISTP=SS (static-static, equivalent to ISTP=NO) by default, but can also be ISTP=TT (total-total, equivalent to ISTP=YES), ISTP=TS (total-static) and ISTP=ST (static-total). These designations do not change the lump states themselves, and the designations are applied to the defined inlet and outlet.

Torque (TORQ) and shaft power or dissipation (QTMK) can be specified for a TABULAR connector in a manner paralleling the definition of G and H. For example, if a list of G values at corresponding H values is given, then a list of corresponding TORQ and/or QTMK values can be specified. QTMK values will be applied as heat sources to the current downstream lump. TORQ is generally ignored by the code and is available mostly for user logic (such as shaft or rotor dynamics or speed determination and torque balancing calculations). However, if ROTR is nonzero and INTORQ=YES is specified, then this TORQ value will be accepted by the path rotation options and QTMK will be calculated on that basis.*

If the tables are symmetric (SYM=YES), then TORQ and QTMK values will be negated for negative values of H.† In other words, if H is negative, then -TORQ and -QTMK will be returned. Discontinuities and other strong variations in TORQ and QTMK at the origin or at any other point are not illegal but may indirectly cause numerical difficulties. (TORQ is not used directly by the program, but QTMK is applied to the downstream lump's QTM sum.)

---

* If either TORQ or QTMK are specified in a TABULAR connector with nonzero ROTR, then the TABULAR values will override any path rotation options. Note, however, that if INTORQ=NO (the default), and QTMK but not TORQ is specified within the TABULAR tables, then the TORQ value will correspond to the rotation value (based on EFFP and the spin pressure force) and will *not* correspond to the user's specified QTMK.

† Since negative values of H are illegal for HU=201 or HU=202, TORQ and QTMK values are never negated when using those definitions of head.

TABULAR Format in CONN Subblocks:

```
... 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]
```

where:

AF..... Effective flow area (basis of velocity)

AFTH..... throat flow area for optional choking calculations

AFI ..... Inlet flow area. If AFI is specified, AFJ must also be specified, and AF will be calculated automatically

AFJ ..... Outlet flow area.

HG..... Array number containing doublet array G_1, H_1, G_2, H_2, ...

G must increase by default, and adjacent H values cannot be equal.

If SYM=YES (default), then $G_1=H_1=0.0$ (unless HU=201, in which case $H_1=1.0$)

If MONOH=YES, then H must increase monotonically, and G can optionally be nonmonotonic

GLIST..... Array number containing singlet array G_1, G_2, ...

G must increase monotonically by default. If SYM=YES (default), then $G_1=0.0$

If MONOH=YES, then H must increase monotonically, and G can optionally be nonmonotonic

HLIST..... Array number containing singlet array H_1, H_2, ... with each value corresponding to the GLIST array.

Adjacent H values cannot be equal. If SYM=YES (default), then $H_1=0.0$

If MONOH=YES, then H must increase monotonically.

HGTABLE. . . . Array number containing a bivariate array of head (H) as a function of flow rate (G) and the "other" or "open" factor (O):

$n, \quad G_1, \quad G_2, \quad \ldots G_n$

$O_1, \quad H_{11}, \quad H_{12}, \quad \ldots H_{1n}$

$O_2, \quad H_{21}, \quad H_{22}, \quad \ldots H_{2n}$

...

$O_m, \quad H_{m1}, \quad H_{m2}, \quad \ldots H_{mn}$

G must increase monotonically by default, O must increase monotonically, and adjacent H values (in each row) cannot be equal. If SYM=YES (default), then G_1=0.0, and H_{i1}=0.0 for all i (unless HU=201, in which case H_{i1}=1.0 for all i). Because negative OFAC signals closure, all O values should be nonnegative.

If MONOH=YES, then H must increase monotonically, and G can optionally be nonmonotonic

GHTABLE. . . . Array number containing a bivariate array of flow rate (G) as a function of head (H) and the "other" or "open" factor (O):

$n, \quad H_1, \quad H_2, \quad \ldots H_n$

$O_1, \quad G_{11}, \quad G_{12}, \quad \ldots G_{1n}$

$O_2, \quad G_{21}, \quad G_{22}, \quad \ldots G_{2n}$

...

$O_m, \quad G_{m1}, \quad G_{m2}, \quad \ldots G_{mn}$

G must increase monotonically by default, O must increase monotonically, and adjacent H values (in each row) cannot be equal. If SYM=YES (default), then H_1=0.0 (unless HU=201, in which case H_1=1.0), and G_{i1}=0.0 for all i. Because negative OFAC signals closure, all O values should be nonnegative.

If MONOH=YES, then H must increase monotonically, and G can optionally be nonmonotonic.

HU . . . . . Head unit identifier: see Table 3-6. In addition to those options, HU=201 may be used to designate H as referring to pressure *ratio* (defined inlet to outlet) instead of pressure *drop*. HU=202 may be used to designate H as referring to the Martin β factor, which is a function of pressure ratio and ENLT (specified separately). The meaning of H can be further modified via the ISTP option.

GU . . . . . Flow rate unit identifier: see Table 3-5. The meaning of G can be further modified via the ISTP, NAF, and MREF options.

ENLT. . . . . Number of sequential constrictive losses, such as labyrinth teeth, used within the calculation of the Martin β factor if HU=202 (per the equation provided in the main text). ENLT is of type real, not integer.

ISTP. . . . . If ISTP=YES (or equivalently, ISTP=TT), then head is defined on the basis of total or stagnation pressures, and either AF or both AFI and AFJ are required inputs. Otherwise by default (ISTP=NO or equivalently, ISTP=SS), static inlet/outlet pressures are used without accounting for dynamic head. ISTP=TS (total-static) and ISTP=ST (static-total) are also

available as options. ISTP also affects the meaning of TREF and PREF.

NAF ..... By default (NAF=0), G values provided are assumed to be based on the units specified by GU. NAF=1 instead means that the G values are actually mass fluxes based on the flow area AF, with the parameter GCF available for conversions or scaling. In other words, for NAF=1 the values supplied as "G" are actually to be interpreted as Φ = GCF*G/AF. For NAF=2, Φ = GCF*G/AFI and for NAF=3, Φ = GCF*G/AFJ.

GCF ..... A G conversion factor (defaulting to unity) that is active if either NAF or MREF is nonzero. *GCF is ignored if MREF=NAF=0.*

MREF ..... By default (MREF=0), G values provided are assumed to be based on the units specified by GU. MREF=1 instead means that the G values are actually corrected mass flows: the values supplied as "G" are actually to be interpreted as $\Phi = GCF*G*T^{1/2}/P$ where T and P are the absolute inlet temperature and pressure, respectively (static or total, depending on ISTP), and G is determined by the unit selector GU. Absolute means the units of T are either K or R (depending on UID), and the units of P are either psia or absolute Pascal. GCF may be used for unit conversions or scaling.

For MREF=2, $\Phi = GCF*G*(T/T_{ref})^{1/2}/(P/P_{ref})$ where the reference temperature and pressure (TREF, PREF) are specified separately. For MREF=3, a more complicated correction is made for equivalent conditions (see main text) using a reference temperature and pressure and also a reference ratio of specific heats, γ_{ref} (GREF).

MREF=4 invokes a slightly more complicated variation of MREF=1 that includes gas properties and unit conversions: $\Phi = GCF*G*(RT/g_c)^{1/2}/P$ where R is the gas constant, and g_c=1 if UID=SI or g_c=32.174 if UID=ENG. MREF may be used in combination with NAF.

TREF ..... The reference temperature (K or R, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

PREF ..... The reference pressure (absolute Pascal or psia, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

GREF ..... The reference ratio of specific heats, $\gamma_{ref} = C_p/C_v$, used for MREF=3.

OFAC ..... O ("other" or "open") factor for bivariate options (HGTABLE or GHTABLE). Negative OFAC signals closure of the connector *whether or not the HGTABLE or GHTABLE options are used*. Any nonnegative value will reopen the path.

OMULT..... One-time conversion multiplier for O units in array: every O value in the bivariate arrays (HGTABLE or GHTABLE) will be multiplied by OMULT to match the model units of OFAC. (The O values themselves are not changed. Rather, the interpolation is scaled according to OMULT. Zero values of OMULT are illegal. If multiple TABULAR devices use the same HGTABLE or GHTABLE array, they all may use distinct values of OMULT without conflict.)

SYM ..... Symmetric H vs. G relationship. If SYM=YES (the default, except when HU=201, and mandatory when HU=202), then the first H and G values

should be zero (the first H must be unity for HU=201), and these values will be applied to both flow rate directions. Otherwise, if SYM=NO, then the G values will normally start with negative values (with the initial H values usually negative as well) to cover this region.

TLIST . . . . . Array number containing singlet array of torque $TORQ_1$, $TORQ_2$, ...

This option can only be used in combination with GLIST and HLIST, with the torque values corresponding to those H and G values.

TGTABLE. . . . Array number containing a bivariate array of torque (TORQ) as a function of flow rate (G) and the "other" or "open" factor (O):

n, G_1, G_2, ... G_n

O_1, $TORQ_{11}$, $TORQ_{12}$, ... $TORQ_{1n}$

O_2, $TORQ_{21}$, $TORQ_{22}$, ... $TORQ_{2n}$

...

O_m, $TORQ_{m1}$, $TORQ_{m2}$, ... $TORQ_{mn}$

This option can only be used in combination with HGTABLE, and the values of O and H must be the same in both the HGTABLE array and this array.

THTABLE. . . . Array number containing a bivariate array of torque (TORQ) as a function of head (H) and the "other" or "open" factor (O):

n, H_1, H_2, ... H_n

O_1, $TORQ_{11}$, $TORQ_{12}$, ... $TORQ_{1n}$

O_2, $TORQ_{21}$, $TORQ_{22}$, ... $TORQ_{2n}$

...

O_m, $TORQ_{m1}$, $TORQ_{m2}$, ... $TORQ_{mn}$

This option can only be used in combination with GHTABLE, and the values of O and H must be the same in both the GHTABLE array and this array.

QLIST . . . . . Array number containing singlet array of turbomachinery heating rates $QTMK_1$, $QTMK_2$, ...

This option can only be used in combination with GLIST and HLIST, with the heating values corresponding to those H and G values.

QGTABLE. . . . Array number containing a bivariate array of turbomachinery heating rate (QTMK) as a function of flow rate (G) and the "other" or "open" factor (O):

n, G_1, G_2, ... G_n

O_1, $QTMK_{11}$, $QTMK_{12}$, ... $QTMK_{1n}$

O_2, $QTMK_{21}$, $QTMK_{22}$, ... $QTMK_{2n}$

...

O_m, $QTMK_{m1}$, $QTMK_{m2}$, ... $QTMK_{mn}$

This option can only be used in combination with HGTABLE, and the values of O and H must be the same in both the HGTABLE array and this array.

QHTABLE. . . . Array number containing a bivariate array of turbomachinery heating rate (QTMK) as a function of head (H) and the "other" or "open" factor (O):

n, H_1, H_2, ... H_n

O_1, $QTMK_{11}$, $QTMK_{12}$, ... $QTMK_{1n}$
O_2, $QTMK_{21}$, $QTMK_{22}$, ... $QTMK_{2n}$
...
O_m, $QTMK_{m1}$, $QTMK_{m2}$, ... $QTMK_{mn}$

This option can only be used in combination with GHTABLE, and the values of O and H must be the same in both the GHTABLE array and this array.

UPF ..... weighting factor for properties: 1.0 = base H and G on inlet density, 0.0 = base H and G on outlet density.

defaults:

AF..... -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if neither AF nor AFI/AFJ have been specified, then no kinetic energy can be extracted nor can choking calculations applied, nor can anything other than ISTP=NO or ISTP=SS be used).

AFTH ..... AF, or AFI if both AFI and AFJ are used instead of AF

AFI ..... -1.0 (use AF)

AFJ ..... -1.0 (use AF)

HU..... 0 (meters of working fluid if UID=SI, else feet of working fluid if UID=ENG)

GU..... 0 (m^3/s if UID=SI, else ft^3/hr of working fluid if UID=ENG)

ISTP ..... NO (pressures etc., based on static values), same as ISTP=SS.

NAF ..... 0 (G is absolute flow, not a flux)

GCF 1.0

MREF ..... 0 (G does not contain corrections for the inlet state, and GCF is ignored)

OFAC 0.0

OMULT..... 1.0

SYM ..... YES (unless HU=201, in which case the default is SYM=NO)

UPF ..... 1.0 (inlet properties). Note that this default UPF value (unlike that for tubes and STUBE connectors) does not depend on prior PA DEF blocks

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Restrictions: Either the HG method, the HLIST/GLIST method, the HGTABLE, or the GHTABLE may be used. TORQ and QTMK values cannot be specified with the HG method. Otherwise, they may be specified using a method that corresponds to the HG method.

Guidance: Extrapolations will be made for the functions H(G) or G(H) beyond the specified end values of H and G. Thus, if only two points (two G,H value pairs per the HG or HLIST/GLIST methods) are defined, a global linear relationship will be assumed.

Guidance: MIXARRAY must be specified in HEADER OPTIONS to use bivariate arrays (any of the *TABLE options), but be sure to watch for preprocessor cautions issued for other arrays.

Guidance: If OFAC values are outside the limits of the O values specified in the HGTABLE, then the limiting values are used: no extrapolation is made. Within the valid range values, linear interpolation is performed. Note that negative OFAC signals closure of the TABULAR connector. Therefore, only nonnegative values of O should be used in the table. If O represents the open fraction of a value, for example, and zero is a valid value, then the head values at O=0 should be specified (and not unreasonably large) to allow interpolation for small but nonzero OFAC values (i.e., nearly closed).

Caution: If either G or H is not monotonic, an unstable region is created.* Steady states will rarely return an answer within this nonmonotonic region, and convergence problems may occur as a result. Similarly, within a transient small time steps and rapid changes are to be expected while the path moves through such a region.

Guidance: **Application to near-perfect gas leakage flows in labyrinth seals:** Martin's formula for unchoked gas flows in a straight labyrinth seal, modified by Vermes, is:†

$$\dot{m}\frac{\sqrt{(RT_{in}/g_c)}}{P_{in}} = \frac{KA_g}{\sqrt{1-\alpha}}\sqrt{\frac{1-(P_{out}/P_{in})^2}{N-\ln(P_{out}/P_{in})}}$$

where the right hand term is the Martin β factor, K is a constriction factor (usually about 0.67 but can approach unity ... it is roughly the same as the coefficient of discharge, C_d, for an orifice), A_g is the geometric flow area of the gap, α is based on the tooth geometry (α = 8.52/[(P-L)/C + 7.23] where C is the clearance, P is the tooth pitch, and L is the tooth thickness). The quantity P_{in}*A_g should have units of force. The quantity K can be decreased for stepped or combination seals.

The above equation can be specified using the HU=202 option, the MREF=4 option, and perhaps (if AF=A_g) the NAF=1 option.‡ The default choking methods should be turned off (MCH=0), and the choking limit supplied separately via a steep section within the H,G tables. An unchoked example of the application of TABULAR connectors to labyrinth seals is provided later in this subsection.

Guidance: The current flow rate, corresponding to FR but in the same units as the G in the TABULAR tables, is available as the output variable GFR.

---

* Specifically, the slope GK = ∂(FR)/∂(ΔPL) becomes negative.

† G. Vermes, "A Fluid Mechanics Approach to the Labyrinth Seal Leakage Problem," Journal of Engineering for Power, April 1961, pp. 161-169.

‡ Caution should be employed, however, in making AF=A_g since this might cause choking or artificially large kinetic energies.

Examples:

```
PA CONN,40,1,5, FR=22.0, DEV=TABULAR, HG = 102
PA CONN,1,1,2, DEV=TABULAR, AF=0.01
      HLIST = 301, HU=4,  SYM=NO
      GLIST = 302, GU=9
PA CONN,23,123,124, DEV = TABULAR, AF = AF1020
      HGTABLE = 103, HU = 12, GU = 15
      OFAC = DL#up, OMULT=1.0
```

Example of a HGTABLE with corresponding TORQ and QTMK arrays:

```
header array data, loopie
 10 = 5, 0.0, 25., 50.0, 75., 100.0   $ glist
      0.0, 0.0, 0.05, 0.25,0.45, 0.5  $ hlist
      0.5, 0.0, 0.1,  0.5, 0.9, 1.0   $ hlist
      1.0, 0.0, 0.3,  1.5, 2.7, 3.0   $ hlist
 20 = 5, 0.0, 25., 50.0, 75., 100.0   $ glist
      0.0, 1.0, 1.05, 1.25,1.45, 1.5  $ tlist
      0.5, 1.0, 1.1,  1.5, 1.9, 2.0   $ tlist
      1.0, 1.0, 1.3,  2.5, 3.7, 4.0   $ tlist
 30 = 5, 0.0, 25., 50.0, 75.,100.0    $ glist
      0.0, 2.0, 2.05, 2.25,2.45,2.5   $ qlist
      0.5, 2.0, 2.1,  2.5, 2.9, 3.0   $ qlist
      1.0, 2.0, 2.3,  3.5, 4.7, 5.0   $ qlist
...

header flow data, loopie
...
pa conn,2,2,1,dev=tabular
      hgtable = 10, hu = 105, gu = 112
      tgtable = 20
      qgtable = 30
      ofac = ohyes
```

Example of a corrected mass flow case:

```
HEADER ARRAY DATA, TABOO
 1 =  0.0             $ corrected mass flow:
      0.004048                $     mdot(kg/s)*sqrt(T in K)/(P in kPa)
      0.005728                $     T and P are total temp, press
      0.007218
      0.008057
      0.008323
      0.008374
      0.008432
 2 =  1.0             $ (total) pressure ratio
      1.06747
      1.17949
      1.34829
      1.57706
      1.93957
      9.8226
      12.1602
...

HEADER FLOW DATA,TABOO, ...
...
pa conn,23,2,3,dev=tabular
      glist = 1, gu = 13      $ kg/s (before correction)
      hlist = 2, hu = 201     $ pressure ratio
      mref = 1                $ simple corrected mass flow
      gcf = 1.e3              $ Pa/kPa
      istp = tt               $ use total temps, pressures everywhere
```

Example of a labyrinth seal, full deck. Note that the “table” consists of a straight line:

```
header options data
title validating labyrinth seal with TABULAR. P 14 of Sect. 405.7, GE manual
        output = labytaby.out
header control data, global
        uid=eng         $ English units
        abszro=0        $ deg R. ... would be -459.67 if F
        patmos=0        $ psia
header register data
        clear = 0.02    $ clearance
        leng = 0.04     $ length of tip
        ess = 0.25      $ tooth pitch
        eee = 0.25      $ tooth depth, not used
        en = 14         $ 14 straight
        pup = 65        $ upstream pres
        pdn = 14.8      $ downstream pres
        tup = 530.      $ upstream temp
       alpha = 8.52/((ess-leng)/clear + 7.23)  $ GE manual had "7.32" (typo)
      FCO = 1/sqrt(1.0-alpha) $ GE manual says alpha**2 but not original ref.
        KAY = 0.85      $ Constriction factor (from GE manual)
        AREA = 1.0E-5   $ ft2. guess: washes out since only flux is know
C                       Just don’t make so small that chokes
insert air8c.inc
header array data,laby
        1 = 0.0, 0.0, KAY*FCO, 1.0         $ constant! Unchoked only
header flow data,laby,fid=8729
lu def, tl=tup,pl=pup,xl=1.0
lu plen,1               $ static: no extra acceleration or expansion
lu plen,2, pl=pdn
c
pa conn,1,1,2,fr=0.01,dev=tabular
        mch = 0         $ turn off choking (should put in curve)
        hg = 1
        hu = 202        $ Martin beta factor
        monoh = yes     $ not really necessary, but better to use if true
        gu = 110        $ lbm/s
        gcf = 1./144.   $ in2/ft2 correction for Pin*Ag
        enlt = 14       $ 14 teeth
        mref = 4        $ use scaling of sqrt(RT/g)/P
        naf = 1         $ use per area, af
        af = area       $ OK in this case, but not always a good idea!
header operations
buildf all
        call steady
        call lmptab(’all’)
        call pthtab(’all’)
        call regtab
C m/A is 76.3 lbm/s-ft2 in GE Manual, 93.2 corrected for the ALPHA typos
        write(nout,*) ’ Mass flow per area = ’,laby.fr1/3600./area
C from output file:
C               Mass flow per area =  92.20761
end of data
```

3.5.17 TURBINE Option

A TURBINE connector* is a component-level model of a gas, steam, or hydraulic turbine (whether axial or radial): a turbine whose performance varies as a function of pressure difference or ratio for a given rotational speed. While a TURBINE connector is normally used to represent an entire turbine containing any number of stages, it may also be used to model either a single turbine stage or a radial portion of an axial turbine or stage (e.g., hub, midline, or tip). Such subset models may be combined in series (for single stages) and/or in parallel (for radial portions).

In order to simulate such a turbine or turbine stage, the user must specify mass flow rate (G, whether actual, corrected, or a "flow parameter") as a function of pressure difference† or ratio (H) and perhaps rotational speed (SPD). The default H is unitless: a pressure ratio (whether total to total, or total to static).

The flow rate G may be a mass or volumetric flow rate, or it may be specified as a flux (per unit area), or as the flow parameter $GCF*G*T_{in}^{1/2}/P_{in}$ based on the current fluid state at the inlet, or as an effective flow based on a reference inlet state (described later).

A single curve of flow at a single speed may be specified as a function of pressure ratio (by default, although pressure differences may be used as well), or a family of curves can be defined as a function of either actual or equivalent speed. For a family of curves, each flow curve (at each speed) may be a different length if a single set of pressure ratios is used for all such flow curves: it is expected that the curves at low speeds will not extend to the highest pressure ratios.‡ This flexibility is contrasted with the PUMP and TABULAR options, which require a rectangular grid of M by N points with no "cut off corners." Therefore, the "GLIST" and "HLIST" parameters for a TURBINE and not completely analogous to those of a PUMP or TABULAR.**

The efficiency may be defined as a constant, or as a function of pressure ratio, or as a function of velocity ratios (U/C) and perhaps both velocity ratio and pressure ratio. As an alternative, the power extracted from the flow can be specified as a function of pressure ratio and perhaps speed, with efficiency being calculated as a result.

Figure 3-12 provides plots of one type of input acceptable to the TURBINE utility: mass flow (as the corrected mass flow $G*T^{1/2}/P$, where G is in kg/s in this case) as a function of speed and pressure ratio, and efficiency as a function of the same speeds and pressure ratios.

A specialized output routine, TURBMAP (Section 7.11.8), is available for TURBINE devices. TURBMAP includes performance information as well an echo of key inputs.

---

* The assistance of D. Mohr of D&E Propulsion & Power in designing this connector device is gratefully acknowledged.

† For pressure differences defines as "heads," note that H is by default defined in terms of the density of the working fluid itself, not water, mercury, or some other reference fluid. (Alternate units, such as "inches of water" or "mm of Hg" are also available, but are actually units of pressure, not head.) Thus, one foot of head is the weight of a column of one foot of the working fluid under one gravity.

‡ This subject is related to choking, which is assumed in a TURBINE device to be contained within the performance curves, as described later (Section 3.5.17.7).

** Indeed, the GLIST and HLIST for a PUMP are not completely analogous to that of a TABULAR connector. The actual means of specifying G and H varies from device to device despite the commonality of input keywords.

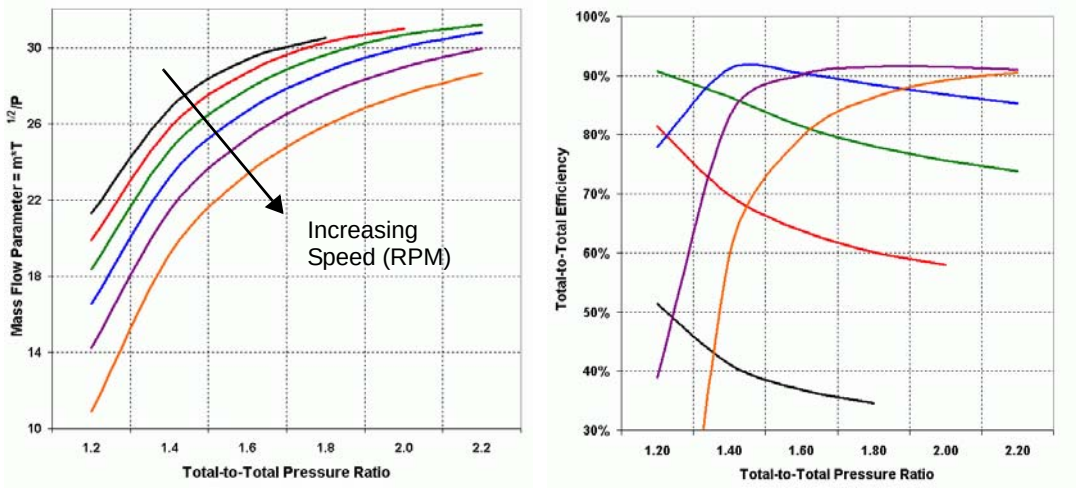


Figure 3-12 Sample Turbine Performance Maps

3.5.17.1 Pressure-Flow Curves and Maps

There are various ways to represent the pressure-flow performance map depending on information available. As will be detailed later, “flow (G)” may actually be a mass flow rate or mass flux, a volumetric flow rate or velocity, a flow parameter corrected for inlet conditions ($T^{1/2}/P$), or a equivalent flow based on a reference state. The choice of G units and method are elected using the unit flag GU combined with the reference method MREF and flow area flag NAF. Similarly, “pressure (H)” may be a head or pressure difference (for a hydraulic turbine perhaps), or a pressure ratio according to the unit flag HU, and these pressures may be optionally chosen to be based on static or total conditions using the ISTP control.

The choices for specifying the pressure-flow relationships include:

1. A single curve of flow versus pressure difference or pressure ratio.

   This method will be referred to as “the GH= method.”

2. A set of curves, each at increasing rotational speed. (If an equivalent flow method is used, this “speed” may optionally be chosen to be an equivalent speed based on a reference state.)

   This method will be referred to as “the GLIST= method.”

   There are two submethods available for the GLIST= method, depending on whether a single set of pressure ratio points is used for all speed curves, or whether a *family* of such H curves is supplied, one for each speed. The need for this complexity is evident in the example turbine performance data plotted in Figure 3-12. The black and red lines (which represent the two lowest speeds) do not extend to the same high pressure ratios as do the characteristic curves for higher speeds. This can be accommodated by either providing a different range of pressure points for each speed curve, or by using a single range for all curves and allowing the low speed curves to be truncated at higher pressure ratios. *Examples of both submethods are provided later in this subsection.*

In the “HLIST= submethod,” a list of pressure arrays is input, with each value corresponding to the same location in the corresponding GLIST array. All of the arrays within both HLIST and GLIST must then be the same size: the same number of data points must be provided for each speed line. HLIST and GLIST then both becomes a “list of lists,” and the map space is rectangular (N speeds by M data points at each speed) with no cut-off corners.

In the contrasting “HLIST1= submethod,” a single array of pressure points can be used to refer to all of the speed lines: HLIST1 is a single list, not a “list of lists” as is HLIST. Using this submethod, the points within the flow arrays (GLIST) must be listed at the same pressure points. For example, the first flow values in each GLIST array should correspond to the first pressure value in the HLIST1 array. To accommodate the missing data for low speeds and high pressure ratios (Figure 3-12), the GLIST arrays need not all be the same length: the points at higher pressure pressures may be omitted if unavailable: the corner may be cut off.

3. If the pressure-flow maps exist as functions or are provided via calls to (perhaps COM-linked external) turbine design codes, then the current value of H and G can provided directly as G(H): flow as a function of current pressure difference or ratio. Furthermore, SINDA/FLUINT requires the slope at the current speed and pressure ratio: dG/dH, the inverse of which (dH/dG) cannot be zero (refer to Section 3.5.17.7 for modeling sonic limits).

   In other words, the user can omit performance arrays and can instead manipulate the turbine parameters directly in logic and/or expressions. The variables for H and G are HTURB and GTURB. The code provides the current value of HTURB (head, pressure drop, or pressure ratio), and the user must supply the corresponding values for GTURB (flow rate, mass flux, flow parameter, equivalent flow, etc.) and DGDH (the slope).

   This method will be referred to as “the GTURB= method.”

Arrays (method #1 or #2 above, the GH= or the GLIST= method) should have large enough range to cover all anticipated values. This includes operation with negative H or G. However, under the assumption that any answer is better than none, FLUINT will extrapolate from the available information to cover any excursions outside of the defined range and will signal such an excursion with an indicator flag, LIMP (a translatable path output variable):

LIMP = 0: nominal (within range)
LIMP = -1 or -2: pressure ratio or difference is too low or too high, respectively
LIMP = 1 or 2: flow rate is too low or too high, respectively
LIMP = -11 or -12: speed is too low or too high, respectively

In the case of multiple excursions, H excursions will be reported instead of any other excursions, which speed excursions will be reported over G excursions. Outside of the supplied range, most values (efficiencies being a notable exception) are extrapolated linearly from the nearest available data.

H must increase monotonically: no zero slopes (dH/dG) are allowed. At least two data pairs are needed to define the simplest turbine curve: a straight line.

Zero Speed Turbines: Zero speed is problematic for the calculation of torque, and should be avoided if possible. If zero speed is truly required, a full-map option (the GLIST= method) should be used with one speed line input that corresponds to zero speed. Otherwise, avoid zero and negligibly small speeds. For example, if turbine start-up to 10,000 rpm is being modeled, consider starting at 100 rpm. Also, consider using LOSS or CTLVLV elements in parallel to represent the stopped condition, with logic or expressions employed to alternate between the TURBINE model and the LOSS or CTLVLV model.

3.5.17.2 Flow Rate (G) Options

The base units for the flow rate, G, are defined using the GU parameter whose meaning is summarized in Table 3-5. However, the actual flow rate value provided in the tables, G_{table}, may deviate from the units specified by GU, G_{GU}, as described next.

First, G as provided in the tables (which corresponds to GTURB) may be defined on an areal (flux) basis using the NAF flag. For example, NAF=1 means that the user-provided G values in the tables (or GTURB and DGDH values if the GTURB= method is used) are actually:

$$G_{table} = GCF*G_{GU}/AF \qquad (NAF = 1)$$

where AF is the flow area and GCF is a unit conversion and scaling factor (defaulting to 1.0), and where G_{GU} is in units specified by the units flag GU.

Second, the “G” values may represent a reference flow rate or corrected mass flow rate using the MREF flag. For example, if MREF=1 then the user-provided G values are actually understood to be:

$$G_{table} = GCF*G_{GU}*T^{1/2}/P \qquad (MREF = 1)$$

where T is the absolute inlet temperature and P is the absolute inlet pressure (in user units according to UID), and GCF is one again an available unit conversion and scaling factor that defaults to unity.

MREF can be used in combination with the NAF flag. For example, if MREF=1 and NAF=1, then the supplied “G” values are understood to be of the form:

$$G_{table} = GCF*G_{GU}*T^{1/2}/(P*AF) \qquad (MREF = 1, NAF = 1)$$

If MREF=2, then the provided “G” values are understood to include a reference state:

$$G_{table} = GCF * G_{GU} * (T/T_{ref})^{1/2} / (P/P_{ref}) \qquad (MREF = 2)$$

The above two MREF options are recommended (but not restricted) to gas flows, and nearly perfect gases as well. A third option, MREF=3, is based on reference or equivalent inlet conditions, and is *only* available for gas or steam flows:*

$$G_{table} = GCF * G_{GU} * (\varepsilon_{ref}/\varepsilon) * (V_{cr}^2/ V_{cr,ref}^2)^{1/2} / (P/P_{ref}) \qquad (MREF = 3)$$

where:

$$\varepsilon = \gamma \left\langle \frac{2}{\gamma + 1} \right\rangle^{\gamma/(\gamma - 1)}$$

$$V_{cr}^2 = \left\langle \frac{2\gamma}{\gamma + 1} \right\rangle g_c RT$$

and ε_{ref} and $V_{cr,ref}$ are based on γ_{ref} and T_{ref}, and γ is the ratio of specific heats, $\gamma = C_p/C_v$.† R is the gas constant, and g_c is the acceleration of gravity (applicable only to English units).

If MREF=3 is selected, and if a full map option is used (via the GLIST= method), the user may elect to define turbine speeds in the tables as equivalent speeds instead of absolute speeds. This selection is controlled by the NSPD flag. By default, NSPD=0 and speeds in the SPEEDS array are actual (e.g., rpm). If instead NSPD=1 (and MREF=3 and the GLIST= method is used), then the speeds in the SPEEDS array are considered to be equivalent speeds as follows:

$$N_{table} = N_{equ} = N_{actual} * (V_{cr,ref} / V_{cr}) \qquad (MREF = 3, NSPD = 1)$$

Furthermore, if MREF=3 and NSPD=1 *and* if power Q has been supplied instead of efficiency (see PLIST option in Section 3.5.17.4), then the input power in the PLIST tables is assumed to be scalable as an equivalent power as well:‡

$$Q_{table} = Q_{equ} = Q_{actual} * (V_{cr,ref} / V_{cr}) * (\varepsilon_{ref}/\varepsilon) / (P/P_{ref}) \qquad (MREF = 3, NSPD = 1 \text{ and PLIST supplied})$$

A fourth option, MREF=4, is similarly restricted to use in gas or steam flows:

$$G_{table} = GCF*G_{GU}*(RT/g_c)^{1/2}/P \qquad (MREF = 4)$$

where R is the gas constant and g_c=1 if UID=SI or g_c=32.174 if UID=ENG.

Note that GCF is not applied unless either NAF or MREF is nonzero.

---

* If MREF=3 or MREF=4, then the code will abort if 100% liquid is upstream, and will ignore liquid in the calculation of γ and R if a two-phase state is upstream.

† The ratio P/P_{ref} is sometimes referred to as "δ," and the ratio $(V_{cr}/V_{cr,ref})^2$ is often referred to as "θ." Using this nomenclature, the equivalent speed is defined as $N_{equ} = N_{actual}/\theta^{1/2}$. If the inverted ratio $\varepsilon_{ref}/\varepsilon$ is referred to as "ε'" then $G_{equ} = G_{actual} * \varepsilon'\theta^{1/2}/\delta$. These factors ($\delta$, θ, ε) are printed in the TURBMAP output.

‡ With P/P_{ref} referred to as "δ," and $(V_{cr}/V_{cr,ref})^2$ referred to as "θ," and the ratio $\varepsilon_{ref}/\varepsilon$ referred to as "ε'," then equivalent power is defined as $Q_{equ} = Q_{actual} *\varepsilon'/(\delta\theta^{1/2})$.

3.5.17.3 Pressure (H) Options

The base units for the pressure gradient or ratio, H, are defined using the HU parameter whose meaning is summarized in Table 3-6. For gas or steam turbines, HU=201 is a common choice, meaning that H is unitless: a pressure ratio. For hydraulic turbines, several head and pressure difference options are available instead. Unlike with the G options, $H_{table} = H_{HU}$ and there is no equivalent of GCF (i.e., "HCF" is not applicable to a TURBINE device).

However, the user has control over whether up or downstream states are to be considered as static or total (stagnation), and this choice affects not only H but also the above MREF corrections and the efficiency (EFFP, TORQ, QTMK) inputs and calculations described later.

By default, all temperatures and pressures used in turbine calculations are total (if an adjacent lump uses LSTAT=STAG, then its temperature and pressure is *both* static *and* total since the velocity is assumed zero). This choice can be regulated using the ISTP factor, which can be used to change the defined inlet and outlet temperature and pressures used throughout the TURBINE calculations. ISTP=TT (total-total, equivalent to ISTP=YES) by default, but can also be ISTP=SS (static-static, equivalent to ISTP=NO), ISTP=TS (total-static) and ISTP=ST (static-total). ISTP=TT and ISTP=TS are common designations for the turbine component. These static/total designations do not change the lump states themselves, and the designations are applied to the *defined* inlet and outlet (and hence the designations don't change if flow rate reverses).

3.5.17.4 Defining Turbine Efficiencies or Power

Turbine isentropic efficiencies are optional but should be specified or at least estimated (typical values are between 0.3 and 0.9, for example). Efficiencies are used to calculate power extracted (Section 3.5.15.3) and torque (Section 3.5.15.4), and may be specified as either total-total or total-static (matching the pressure method described above via the ISTP parameter).

There are five ways to specify efficiency, η, or alternatively, the extracted power:

1. as a single value, EFFP (updated in logic or expressions)

   This method will be referred to as "the EFFP= method."

2. as an interpolated array of efficiency vs. velocity ratio (U/C). In this case, U is the tangential blade tip velocity, which requires DTURB (the turbine diameter) to be specified as an input. C is the isentropic spouting velocity for a gas or steam turbine, calculated assuming an isentropic expansion from a total inlet state to a static outlet state (independent of ISTP). For hydraulic turbines, C is the fluid jet speed, and is calculated based on the Bernoulli equation again assuming a total inlet state and a static outlet state.

   This method will be referred to as "the AEFFP= method."

3. as an interpolated series of arrays of efficiency vs. velocity ratio (U/C), with different curves supplied for each of a series of pressure differences or ratios (corresponding to the same set of pressures used to define the pressure-flow relationship, perhaps, if the GLIST= method were employed)

   This method will be referred to as "the ELISTR= method."

The ELISTR= method requires that the corresponding pressure ratios be supplied as HLIST1 even if pressure ratios are supplied using the HLIST= submethod.

4. if a full map of G-H has been provided via the GLIST= method, then a full map of efficiencies can be provided in parallel (e.g, full maps of η at the same operating points as the head-flow relationships).

   This method will be referred to as "the ELIST= method."

5. if a full map of G-H has been provided via the GLIST= method, then a full map of extracted power (W or BTU/hr, depending on UID) can be provided in parallel (e.g, full maps of power at the same operating points as the head-flow relationships). In this case, EFFP becomes an output variable.

   This method will be referred to as "the PLIST= method."

Note that turbine efficiency, EFFP, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.).

If arrays have been used to define EFFP (e.g., methods 2-5 above, but not method 1), then the efficiency can be modified or scaled before use, using additive and multiplying terms, EFFA and EFFM, respectively, as follows:

$$EFFP_{final} = EFFM*EFFP_{table} + EFFA$$

EFFM defaults to 1.0, and EFFA defaults to 0.0, so no manipulation of EFFP takes place by default. EFFM and EFFA may be used for various purposes, including modifying the efficiency tables produced by a turbine design code as needed to account for separately modeled stators* and moisture degradations in steam turbines.† EFFM and EFFA can also be used as unknowns when correlating to test data.

Note that if the turbine diameter, DTURB, has been supplied and if the rotational speed, SPD, has units of RPM, then the user may inspect or postprocess the turbine output variable UCR (velocity ratio U/C) whether or not this parameter is used in the AEFFP or ELISTR arrays to define efficiency.

3.5.17.5 QTMK and QTM: Turbomachinery Power

Even for a perfectly efficient turbine (EFFP=1.0 by default), a thermal power of QTMK (a translatable path output variable) will be added‡ to the QTM (a translatable lump output variable) of the lump currently downstream of the turbine. QTM is one component (along with QL and the sums of tie QTIEs and ftie QFs) of the total lump power input QDOT.

Each lump may have only one unique QTM. Therefore, if more than one TURBINE or other turbomachine is placed in parallel, the lump QTM will represent to total or net power added by all turbomachinery currently exhausting to that lump.

---

* If the turbine maps contain stators (nozzles) but they have been modeled explicitly in SINDA/FLUINT, the efficiency can be augmented (e.g., EFFM=1.02) so as not to account for stator losses twice.

† As a example of a simple and conservative correction: EFFM = 0.5*(AL#up+AL#down)

‡ For a turbine, QTMK will be negative (power is extracted from the flow).

The value of QTMK is proportional to the change in enthalpy (normally a negative value) from the defined inlet to the defined outlet pressure, holding entropy constant, times the absolute value of the mass flow rate, multiplied by the isentropic efficiency.

Turbine calculations override the rotating path (ROTR etc.) calculations for QTMK and TORQ (Section 3.5.15.4), and apply factors such as TCF and EFFP independently of analogous path rotation options; turbine options generally take precedence over path rotation options.

3.5.17.6 Hydraulic Torque

The code calculates the hydraulic torque as:

TORQ = TCF*QTMK/SPD

where TCF is input by the user to perform whatever unit conversions are necessary. TCF defaults to unity.

TORQ is a translatable path output variable. In SI units, the TCF contains the conversion of rpm (assuming those are the units of SPD) to rad/sec, resulting in TORQ units of N-m:

```
TCF = 60./(2.*pi)       $ “pi” is a built-in constant
```

In US Customary units, QTMK is in BTU/hr, so assuming SPD is in RPM, to get TORQ in lb_f-ft use:

```
TCF = 778./(2.*pi*60.)        $ 778 converts BTU/hr to lbf-ft/hr
```

Hydraulic torque may be used along with co-solved ordinary differential equations (e.g., DIFFEQ1) to calculate the current speed of the turbine during transients, or may be used to solve a torque balance equation for steady-states (e.g., for turbopumps or turbochargers), perhaps using ROOT_FIND. Note that TORQ is positive for energy into the fluid system. It is therefore usually negative for turbines, and so the negative of TORQ should be set proportional to $d\omega/dt$, the rate of change of shaft speed.

Torque may also be used to calculate power extracted by the turbine. If TORQ is in English units of lb_f-ft, for example, and speed is in rpm, then the horsepower extracted can be calculated as:

HP = -TORQ*SPD*2π/60.0/550.0

Turbine power output may also be taken directly from the output lump’s QTM (in Watts if UID=SI and in BTU/hr if UID=ENG), noting that other turbomachinery may be contributing to that QTM as well.

Note that the turbine torque coefficient, TCF, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.). Similarly, the TORQ calculations are independent of rotating path calculations.

3.5.17.7 Modeling Sonic Limitations (Choking)

At low speeds and high pressure ratios, the turbine performance may be limited by internal choking. The general-purpose flow passage methods for detecting and modeling choked flow are inappropriate for handling the case of a gas or steam turbine, and are therefore disabled (MCH=0) by default.

Choking simulation should be contained within the performance maps themselves, which is consistent with the outputs of typical turbine design software. For example, in the left hand plot in Figure 3-12, all speed lines asymptotically approach the sonic limit (corresponding to a flow parameter value of about 31 in that case).

Problems can arise with this “choking built-into map” approach, however, if the G values are based on the default MREF=0, which means that the G values supplied in the GH or GLIST tables are absolute mass or volumetric flow rates and are not otherwise a function of the inlet pressure. In this case, if G values approach a limit representing choking, DGDH approaches zero which is numerically dangerous (MFRSET-like inflexibility): the G value should not be a function of downstream pressure if the turbine is choked, but if DGDH=0 and MREF=0 then the flow is no longer a function of upstream pressure changes either, which leads to solution problems. For the user-defined turbine (i.e., the GTURB= method), DGDH is not allowed to be zero if MREF=0.

Therefore, when modeling gas or steam turbines, nonzero MREF is highly recommended, otherwise the user must assure that the G values in the tables do not contain zero slopes: the pressure ratio values are not based only on downstream pressure changes but on upstream pressure changes as well.

For hydraulic turbines, the built-in choking methods (invoked using nonzero MCH) *might* be appropriate with values of AFTH that are chosen carefully.

3.5.17.8 FLOW DATA Format

TURBINE Format in CONN subblocks:

```
... 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]
```

where:

GU..... Flow rate unit identifier: see Table 3-5. These units will apply to G values. See also MREF and NAF for further modifiers of these basic units.

HU..... Head unit identifier: see Table 3-6. These units will apply to H values.

GCF ..... A G conversion factor (defaulting to unity) that is active if either NAF or MREF is nonzero. *GCF is ignored if MREF=NAF=0.*

ISTP ..... If ISTP=YES (or equivalently, ISTP=TT), then pressures and other state variables are defined on the basis of total or stagnation conditions, and either AF or both AFI and AFJ are required inputs. By default ISTP=TT, and so both inlet and outlet states are based on total (stagnation) conditions and therefore flow areas (AF or both AFI and AFJ) are required. ISTP=TS (total-static) and ISTP=ST (static-total) are also available as options. ISTP also affects the meaning of TREF and PREF.

NAF ..... Integer mass flux basis flag. By default NAF=0, and hence G values provided are assumed to be based on the units specified by GU. NAF=1 instead means that the G values are actually mass fluxes based on the flow area AF, with the parameter GCF available for conversions or scaling. In other words, for NAF=1 the values supplied as "G" are actually to be interpreted as G_{table} = GCF*G_{GU}/AF where G_{GU} is determined by the unit selector GU. For NAF=2, G_{table} = GCF*G_{GU}/AFI and for NAF=3, G_{table} = GCF*G_{GU}/AFJ. (Note "G_{table}" is equivalent to "GTURB" when using the GTURB= method.)

MREF ..... Integer mass flow reference flag. By default MREF=0, and hence G values provided are assumed to be based on the units specified by GU. MREF=1 instead means that the G values are actually corrected mass flows: the values supplied as "G" are actually to be interpreted as G_{table} = GCF*G_{GU}*$T^{1/2}$/P where T and P are the absolute inlet temperature and pressure, respectively (static or total, depending on ISTP), and G_{GU} is determined by the unit selector GU. Absolute means the units of T are either K or R (depending on UID), and the units of P are either psia or absolute Pascal. GCF may be used for unit conversions or scaling.

For MREF=2, G_{table} = GCF*G_{GU}*$(T/T_{ref})^{1/2}/(P/P_{ref})$ where the reference temperature and pressure (TREF, PREF) are specified separately. For MREF=3, a more complicated correction is made for equivalent conditions (see main text) using a reference temperature and pressure and also a reference ratio of specific heats, γ_{ref} (GREF).

MREF=4 invokes a slightly more complicated variation of MREF=1 that includes gas properties and unit conversions: G_{table} = GCF*G_{GU}* $(RT/g_c)^{1/2}/P$ where R is the gas constant, and g_c=1 if UID=SI or g_c=32.174 if UID=ENG.

MREF may be used in combination with NAF.

(Note "G_{table}" is equivalent to "GTURB" when using the GTURB= method.)

TREF. . . . . The reference temperature (K or R, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

PREF. . . . . The reference pressure (absolute Pascal or psia, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

GREF. . . . . The reference ratio of specific heats, $\gamma_{ref} = C_p/C_v$, used for MREF=3.

GH . . . . . ID of doublet array (in this submodel) containing a single turbine head-flow curve: G (flow rate: units according to GU) as a function of pressure ratio or difference (units according to HU). H must increase monotonically. *No two adjacent H values may be equal (i.e., the slope dH/dG cannot be zero: the inverse of this slope, DGDH, cannot be infinite).* The array pointed to must contain real values or expressions of the form H1,G1, H2,G2 ...
Extrapolations will be flagged as nonzero LIMP values.
Note that this "GH= method" is just one way of specifying the H-G relationship. See also SPEEDS/HLIST/GLIST for the "GLIST= method," and GTURB/DGDH for the "GTURB= method."
Supplied G values may be adjusted per the rules invoked via the MREF and NAF variables, and if so they might be scaled by the parameter GCF.

SPD . . . . . Current/initial turbine speed, in consistent user units. Avoid very small or zero values (the default!) unless the model/inputs are specifically designed for this case. In most cases, zero SPD is equivalent to a closed valve for consistency. Note that SPD is in actual units (e.g., rpm) and is not dependent on the value of NSPD, unlike the values in the SPEEDS array. NOTE: To use the ELISTR or AEFFP options, or for the output variable UCR (velocity ratio U/C) to be valid, SPD *must* be in units of rpm.
SPD should be specified even if using the GTURB= method, since SPD is used for TORQ calculations.

DTURB . . . . . Turbine diameter (in feet or meters, depending on UID). This input is required if either the AEFFP= or the ELISTR= method is used to define efficiency based on velocity ratio (U/C). If DTURB is input, SPD should be in units of RPM such that the output variable UCR (the velocity ratio U/C) can be used, whether or not AEFFP or ELISTR is used.

GTURB . . . . . for turbines whose head-flow is defined by expressions and/or user logic (i.e., by the "GTURB= method" and not by arrays via either the GH= or the GLIST= method), the flow rate or flow coefficient at the current head, HTURB, in units consistent with the combined effects of GU, MREF, NAF, and GCF. If H-G arrays are input via the GH= or the GLIST= method, then this becomes an output.*

DGDH. . . . . for turbines defined by the GTURB= method (i.e., no arrays used), the slope (flow rate or flow coefficient to head or head coefficient) at the current head, HTURB, in units consistent with GU, HU, MREF, NAF, and GCF. Must not be zero if MREF=0, and negative values are legal but can cause

* Note that the calculated value of GTURB will differ from a value calculated using the current FR except for a converged steady-state. FR is the current flow rate, whereas GTURB (updated after FLOGIC 0) is the current flow that the turbine *would* be at if the pressure ratio or head were constant at the current HTURB value. The value of GTURB can be compared directly with the output variable GFR, the value of the current flow rate in the same units as GTURB.

instabilities in some models if persistent. If H-G arrays are input via the GH= or the GLIST= method, then this becomes an output.

SPEEDS .... A full map alternative to the single-curve "GH= method," SPEEDS is the array ID containing a list of speeds for which pressure and flow curves (and possibly efficiencies) will be specified via the HLIST (or HLIST1), GLIST, and optionally the ELIST arrays (see "Full Map Example" below). Like GH, SPEEDS is an array ID, and the entries in that array must be real numbers or expressions in the same units as SPD if NSPD=0, or equivalent speeds if NSPD=1.

NSPD ..... Flag indicating whether the values in the SPEEDS array are in actual speed such as rpm (NSPD=0, the default) or whether they has been specified as equivalent speed (NSPD=1, *valid only if MREF=3*) as described in the main text. Note that NSPD has no effect on the current speed, SPD, which is *always* in actual speed units (e.g., rpm).

NSPD=1 also governs the scaling of power if provided by the PLIST option: both speed and power will be scaled if NSPD=1 (see formulas in main text, Section 3.5.17.9).

HLIST1 .... If SPEEDS is used for a full map (the "GLIST= method"), and if only one set of pressures ratios H (or head or pressure drop, per HU) are to be used for all GLIST arrays, then this is the ID of a SINDA/FLUINT array (in this submodel) containing the single list of values of pressure ratios. The first *value* listed in the HLIST1 array should correspond to the first values in each *array* of GLIST (and optionally, ELIST or PLIST), and so forth. HLIST1 must be large enough to cover all of the pressure ratios needed for the longest GLIST (or ELIST or PLIST or ELISTR) array. H must increase monotonically. *No two adjacent H values may be equal.*

HLIST1 may be used in conjunction with ELISTR, even if GLIST is *not* input, or even if GLIST *is* input along with HLIST (in which case HLIST values are used instead of HLIST1 values for the pressure/flow relationship).

HLIST..... If SPEEDS is used for a full map (the "GLIST= method"), this is the ID of a SINDA/FLUINT array (in this submodel) containing the values of pressure ratio H (or head or pressure drop, per HU) for which table values in GLIST correspond. HLIST is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing pressure ratio (H) values that correspond to each of the N speeds set in the SPEEDS input. HLIST is a list of lists. In other words, the first array ID in the HLIST array should contain a series of pressure ratio values at the first speed in the SPEEDS array. The first array listed should also correspond to the first array of GLIST (and optionally, ELIST), and the first value in the first HLIST array should correspond to the first value in the first GLIST array, and so forth. All such arrays within HLIST and GLIST (and ELIST or PLIST, if used), should be the same length. H must increase monotonically within each HLIST array. *No two adjacent H values may be equal.*

GLIST..... The ID of a SINDA/FLUINT array (in this submodel) which itself contains

a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing flow values that correspond to each of the N speeds set in the SPEEDS input. GLIST is a list of lists. In other words, the first array ID in the GLIST array should contain a series of flow rate values at the first speed in the SPEEDS array.

The array values themselves should be in units consistent with GU. Supplied G values may be adjusted per the rules invoked via the MREF and NAF variables, and if so they might be scaled by the parameter GCF.

EFFP. . . . . Overall turbine isentropic efficiency (real, between 0.0 and 1.0). This is an output if either ELIST, ELISTR, PLIST, or AEFFP arrays are provided. The efficiency basis is a function of ISTP, and therefore may be total-total or total-static, for example.

AEFFP . . . . . ID of the doublet array containing the efficiency as a function of flow velocity ratio U/C, where U is the tangential tip velocity (based on SPD and DTURB) and C is the isentropic spouting velocity for a gas turbine (fluid jet speed for a hydraulic turbine). AEFFP is therefore of the form $(U/C)_1$, η_1, $(U/C)_2$, η_2 .... where velocity ratios are monotonically increasing (i.e., $(U/C)_2 > (U/C)_1$).

End values will be used if the provided range is exceeded; no extrapolations are made for this variable. If this option is used, SPD is assumed to be in rpm.

ELISTR . . . . . The ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT doublet arrays (also in this submodel) containing velocity ratio (U/C) and efficiency value pairs that correspond to each of the pressure ratios set in the HLIST1 array. ELISTR is a list of lists. In other words, the first array ID in the ELISTR array should contain a series of U/C and efficiency pairs at the first pressure ratio in HLIST1.

At each pressure ratio (or difference, depending on HU), the efficiency is provided as a function of flow velocity ratio U/C, where U is the tangential tip velocity (based on SPD and DTURB) and C is the isentropic spouting velocity for a gas turbine (or fluid jet speed for a hydraulic turbine). Each array named in ELISTR is therefore a doublet array of the form $(U/C)_1$, η_1, $(U/C)_2$, η_2 .... where velocity ratios are monotonically increasing (i.e., $(U/C)_2 > (U/C)_1$). In other words, ELISTR is like a list of AEFFP arrays, each applicable to a different pressure ratio or head.

End values will be used if the provided range is exceeded; no extrapolations are made for this variable. If this option is used, SPD is assumed to be in rpm. Double linear interpolation will be made for speeds and velocity ratios between available values, so ideally there should be correspondence between data pairs entered at different rotational speeds. However, since no extrapolation is permitted and since each ELISTR array (at any one pressure ratio) may be a different length, the (U/C) value will be interpolated first up to the available limits, then the pressure ratios will be interpolated second.

ELIST . . . . . The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel)

containing turbine isentropic efficiencies that correspond to the N speeds set in the SPEEDS input. ELIST is a list of lists. In other words, the first array ID in the ELIST array should contain a series of efficiency values (between zero and one) at the first speed in SPEEDS. The values in that first *array* should correspond to the first *array* of GLIST. Edge values are used if the provided range is exceeded; no extrapolations are made for this variable.

PLIST..... As an alternative to setting isentropic efficiency, the extracted power itself can be specified using PLIST. The optional PLIST input is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing turbine hydraulic powers that correspond to the N speeds set in the SPEEDS input. *Power units must be either Watts or BTU/hr, depending on UID.* PLIST is a list of lists. In other words, the first array ID in the PLIST array should contain a series of extracted (positive) power values at the first speed in SPEEDS. Like the alternative ELIST option, extrapolations are not made for this variable past the limits.

If MREF=3 and NSPD=1, then the powers supplied in the PLIST arrays are considered to be “equivalent powers” and will be scaled with changes in the inlet conditions per the formula in the main text (Section 3.5.17.2).

EFFM..... Multiplier to use for EFFP values extracted from array options (ELIST, ELISTR, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

EFFA..... Additive term to use for EFFP values extracted from array options (ELIST, ELISTR, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

TCF ..... Correction factor for the TURBINE output variable TORQ (see main text).

AF..... flow area for kinetic energy and choking (if MCH is nonzero), and for calculating dynamic head if ISTP=YES or ISTP=TS or ISTP=ST. This input is not required if AFI and AFJ are specified instead. AF, or AFI and AFJ, is required if AEFFP or ELISTR is specified. AF might be calculated from the prior PA DEF value of DH if no AF has been supplied.

AFTH..... throat area for choking calculations (see Section 3.5.17.7).

AFI ..... inlet (suction) flow area, if not constant. If AFI is specified, AFJ must also be specified, and AF will be calculated automatically.

AFJ ..... outlet (discharge) flow area, if not constant.

UPF ..... weighting factor for fluid properties: 1.0 = base H and G on inlet (suction) density, 0.0 = base H and G on outlet (discharge) density, if appropriate.

defaults:

- GU . . . . . 0 (m^3/s if UID=SI, else ft^3/hr of working fluid if UID=ENG)
- HU . . . . . 201 (pressure ratio, see also ISTP: total pressure ratio by default)
- GCF . . . . . 1.0 (ignored unless either NAF or MREF is nonzero)
- ISTP. . . . . YES (pressures and all other properties are based on total values), same as ISTP=TT (total-total)
- MREF. . . . . 0 (G in units of GU, unmodified)
- NAF . . . . . 0 (G is absolute flow, not a flux)
- SPD . . . . . 0.0 (zero speed usually means the turbine acts like a closed valve)
- NSPD. . . . . 0 (SPEEDS in units of actual speed, not equivalent speed)
- GTURB . . . . . none (calculated if H-G arrays input using either the GH= or the GLIST= method)
- DGDH. . . . . none (calculated if H-G arrays input using either the GH= or the GLIST= method)
- DTURB . . . . . none (only needed if ELISTR or AEFFP options are used)
- EFFP. . . . . 1.0 (calculated if AEFFP, ELIST, ELISTR, PLIST is used; independent of PA DEF defaults)
- EFFM. 1.0
- EFFA. 0.0
- TCF . . . . . 1.0 (independent of PA DEF defaults)
- AF . . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if neither AF nor AFI/AFJ have been specified, then no kinetic energy can be extracted nor can choking calculations applied, nor can the default ISTP=YES be used. In other words, if *both* ISTP and AF are defaulted, an error will occur.)
- AFTH. . . . . AF, or AFI if both AFI and AFJ are used instead of AF
- AFI . . . . . -1.0 (use AF)
- AFJ . . . . . -1.0 (use AF)
- UPF . . . . . 1.0 (inlet properties). Note that this default UPF value (unlike that for tubes and STUBE connectors) does not depend on prior PA DEF blocks.
- MCH . . . . . 0 (no choking: contained within maps)

Simple Examples:

```
PA CONN, 10,1,2, FR=10.0, DEV=TURBINE, GH=1, AF=Aturb
PA CONN,1,111,112, STAT=DLS, DEV=TURBINE
      GH=114, GU=4 $ Units of L/s
      ISTP = TS $ total-static for all properties
      EFFP = 0.8, SPD = 20000., AF = pi*(RO^2 - RI^2)
```

A full map example is provided below.

Caution: Some of the “volumetric flow rate (G)” units are really mass flow rate units, and most of the “head (H)” units are not length, but rather pressure. Despite the name, “inches of water” has units of pressure, not length or head, for example. In any case where pressure or mass flow rate units are used instead of head and volumetric flow rate units, the relative independence of density will be lost for hydraulic turbines: the turbine curve will be less applicable to different fluids, pressures, temperatures, etc. than those for which the data is taken.

Caution: HTURB and GTURB represent a point on the current (or extrapolated) turbine curve, whereas GTURB can differ from the current flow (as designated by GFR, an output variable corresponding to the current mass flow rate FR but in the same units as GTURB). If corrections are required that are based on the current flow rate, use GFR, not GTURB.

Caution: Leaving MCH=0 as the default is strongly recommended (see Section 3.5.17.7).

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Guidance: Unlike MREF=1 and MREF=4, which provide tools for *accepting* a specific style of performance map, MREF=2 and MREF=3 (and perhaps NSPD=1) can be thought of as tools for *extending* the range of a map created at a single inlet state (temperature, pressure, gas species) to other inlet conditions.

Guidance: A summary of common pressure-flow rate input options is provided below, and summarized in Table 3-8:

1) *The point/slope method.* Specify the GTURB value corresponding to the current program-supplied HTURB, along with the slope at that point: DGDH in units per the GU and HU specifications.

2) *A single curve of G vs. H at one speed.* Specify the GH array (usually, mass flow rate as a function of pressure ratio) and SPD. The G values may be generalized for various input states using the MREF option (and perhaps NAF and GCF options). Use of AEFFP or ELISTR is recommended, but a single value of efficiency (EFFP) can be supplied if available.

3) *A full map of H and G.* Specify the SPEEDS, HLIST or HLIST1, and GLIST lists along with the current/initial speed SPD. If MREF=3, the speeds in SPEEDS may be equivalent instead of actual by specifying NSPD=1. Use of a corresponding ELIST is easiest if available, but efficiency can also be specified as a function of pressure ratio (HLIST or HLIST1) and velocity ratio (U/C) using the ELISTR option. Or, a single array of efficiencies vs. U/C is adequate using AEFFP.

Table 3-8 Summary of TURBINE Input Options

Head-Flow Method	Efficiency (or Power) Method	Other notes
Single Point/Slope (GTURB= and DGDH=)	EFFP=η or AEFFP=$(U/C)_1,\eta_1$, $(U/C)_2,\eta_2$ ...	DTURB is required if AEFFP= or ELISTR= is used in order to calculate U (blade tip velocity). (ELISTR and HLIST1 may also be used here, as described in the next subsection.)
Single Curve (GH=)	EFFP=η or AEFFP=$(U/C)_1,\eta_1$, $(U/C)_2,\eta_2$ .... or ELISTR = ... (with HLIST1 too)	The HLIST1 inputs are not normally needed with a single curve, but if ELISTR is used, then HLIST1 must also be used to supply the pressure ratios corresponding to each ELISTR curve.
Full Map (GLIST=) (HLIST1=)	EFFP=η or AEFFP=$VR_1,\eta_1 VR_2,\eta_2$ ... or ELIST = ... or ELISTR = ... or PLIST = ...	The SPEEDS and either the HLIST or the HLIST1 inputs are required with the GLIST= method. The values in the SPEEDS array may be actual (NSPD=0, the default) or equivalent (NSPD=1), while the current speed SPD is always actual (not equivalent). *In this HLIST1= submethod, each of the arrays named within GLIST, PLIST, and ELIST arrays may be shorter at low speeds than the HLIST1 array.*
Full Map (GLIST=) (HLIST=)	EFFP=η or AEFFP=$VR_1,\eta_1 VR_2,\eta_2$ ... or ELIST = ... or ELISTR = ... or PLIST = ...	The SPEEDS and either the HLIST or the HLIST1 inputs are required with the GLIST= method. The values in the SPEEDS array may be actual (NSPD=0, the default) or equivalent (NSPD=1), while the current speed SPD is always actual (not equivalent) *In this HLIST= submethod, each of the arrays named within GLIST, PLIST, and ELIST arrays must be of equal length.* Use of HLIST1 is optional if needed for ELISTR.

3.5.17.9 Full Map Example: GLIST= Method and HLIST1= Submethod

This section contains an example of the "GLIST= method:" a full map of flow rate and efficiencies as functions of speed and pressure ratio, along with a short description of the mathematical treatment of these maps and the resulting requirements for their structure. This section describes the HLIST1= submethod employing a single set of pressure ratios for all curves, whereas the next subsection illustrates the HLIST= submodel employing separate sets of pressure ratios (one for each speed line).

Variations are commented out, including the PLIST= (instead of ELIST=) method and the use of a flow parameter (i.e., corrected flow instead of absolute flow rate). In this example, 6 speeds are used, with as many as 6 values (flow and efficiency) at each speed are provided, but sometimes as few as 4 values with the "missing" values corresponding to large pressure ratios in HLIST1. HLIST1 itself must contain 6 values of pressure ratio, corresponding to the longest list in GLIST and ELIST (or perhaps PLIST or ELISTR). Thus, the map is *nearly* a full 6x6 grid, except that data values are absent in the corner of the grid corresponding to the highest pressure ratios and lowest speeds. See also Figure 3-6.

For the full map example listed below, note that the TURBINE connector lists for GLIST, ELIST, and PLIST are “lists of lists:” lists of array IDs that point to the arrays containing the actual data in 1D strings. HLIST1, on the other hand, is a simple list of pressure ratios that are appropriate for all values in GLIST, ELIST, and PLIST. In this example, there are six turbine speeds for which data is available, as pointed to in the SPEEDS array (number 1): 16000, 32000, 48000, 64000, 81000, and 97000 rpm. For the first speed (16000 rpm), the first flow rate array (number 201, the first in GLIST array 200) corresponds to the first pressure ratio value in HLIST1 array 100, and also to the first efficiency array 401 (the first in ELIST array 400).

While the efficiency array will be interpolated directly and linearly, and will not be extrapolated (i.e., edge values will be used past the range provided), the flow rate arrays (and powers, if input) will be treated differently: they will be interpolated and perhaps extrapolated between speed and pressure ratio points.

In the example below, G is named as a mass flow rate in units of lb_m/hr (GU=112, MREF=0, NAF=0). For a gas or steam turbine, this method is not recommended since (1) the program cannot scale the results as accurately to new inlet conditions, and (2) the sonic limitations encountered at high pressure ratios and low speeds are more problematic with an absolute mass flow rate (see Section 3.5.17.7). Therefore, a better method is shown (but commented out): G inputs using MREF=1.

Efficiencies are input using the ELIST form which matches the GLIST and HLIST1 points. However, powers may be directly input as an alternative using the PLIST method (these are shown, but commented out). Note that the power extracted is normally a positive value, but that the turbine design software used to generate this map specified a negative value of power in the limit of low pressure ratio and high speed. At that point, the turbine is behaving as a compressor, and the corresponding efficiency is negative as well.

```
HEADER FLOW DATA, TURBINE ...
...
pa conn,10,1,10,dev=turbine
      spd = speed                  $ initial speed (slow but not zero)
      speeds = 1                   $ list of speed values provided
      hlist1 = 100, HU=201         $ list of pressure ratios
      glist = 200, GU=112          $ list of flow arrays; G in lb/hr
            elist = 400            $ list of efficiency array IDs
c           plist = 500            $ if powers had been used instead
      ISTP = yes                   $ total to total pressure used
      af = 0.1*dturb#this**2       $ flow area (constant)
      TCF = 2.*pi*60.*0.2161       $ to get lbf-ft, speed is in RPM
```

```
HEADER ARRAY DATA, TURBINE
c
c speeds (SPEEDS) in rpm
c
        1 = 16000., 32000., 48000., 64000., 81000., 97000.
c
c pressure ratios (HLIST1)
c
        100 = 1.2, 1.4, 1.6, 1.8, 2.0, 2.2
c
c list of GLIST arrays followed by data
  200 = 201, 202, 203, 204, 205, 206
c M lb/hr (MREF=0, default, but less scalable, has sonic problems)
      201=19301.3, 24268.1, 26597.4, 27636.8
      202=18033.0, 23361.7, 26007.3, 27413.2, 28070.3
      203=16627.9, 22287.3, 25187.9, 26845.1, 27793.9, 28273.0
      204=14982.9, 21033.9, 24168.9, 26033.0, 27179.3, 27878.4
      205=12899.0, 19447.0, 22863.6, 24936.5, 26251.0, 27115.8
      206= 9846.2, 17375.9, 21153.6, 23460.3, 24961.7, 25969.1
c Using instead MREF=1 (PREFERRED!) GCF = 1.0
c M*T**(1/2)/P lb/Hr, R, psi
c     201=1877.0, 2753.4, 3448.8, 4031.5
c     202=1753.7, 2650.6, 3372.3, 3998.9, 4549.7
c     203=1617.1, 2528.7, 3266.0, 3916.0, 4504.9, 5040.8
c     204=1457.1, 2386.4, 3133.9, 3797.6, 4405.3, 4970.5
c     205=1254.4, 2206.4, 2964.6, 3637.6, 4254.8, 4834.5
c     206= 957.5, 1971.4, 2742.9, 3422.3, 4045.9, 4630.0
c
c list of Eff arrays followed by data
 400 = 401, 402, 403, 404, 405, 406
      401= 0.51317, 0.41237, 0.36865, 0.34524
      402= 0.81488, 0.69808, 0.63738, 0.60163, 0.57945
      403= 0.90781, 0.86407, 0.81470, 0.77990, 0.75586, 0.73893
      404= 0.77930, 0.91102, 0.90338, 0.88450, 0.86727, 0.85317
      405= 0.38860, 0.82994, 0.90069, 0.91479, 0.91440, 0.90975
      406=-0.38906, 0.59466, 0.79395, 0.86289, 0.89156, 0.90422
c list of Power arrays followed by data BTU/hr
c 500 = 501, 502, 503, 504, 505, 506
c     501=233338., 420348., 560369.,   666839.
c     502=345314., 685009., 947644.,  1152800., 1314164.
c     503=354159., 808913., 1172093., 1462624., 1695803., 1885577.
c     504=273770., 806885., 1248360., 1609511., 1903906., 2146321.
c     505=117652., 679173., 1177428., 1594513., 1937182., 2225008.
c     506=-89764., 434254.,  960180., 1415143., 1798037., 2119134.
```

3.5.17.10 Full Map Example: GLIST= Method and HLIST= Submethod

This section contains an example of the "GLIST= method:" a full map of flow rate and efficiencies as functions of speed and pressure ratio, along with a short description of the mathematical treatment of these maps and the resulting requirements for their structure. This section describes the HLIST= submethod employing separate sets of pressure ratios (one for each speed line), whereas the prior subsection illustrates the HLIST1= submodel employing a single set of pressure ratios for all curves.

For the full map example listed below, note that the TURBINE connector lists for HLIST, GLIST, and ELIST are "lists of lists:" lists of array IDs that point to the arrays containing the actual data in 1D strings. In this example, 5 speeds are used, with 12 values (pressure ratio, flow, and efficiency) provided at each speed. Thus, the map is a full 5x12 grid, noting that the pressure ratios have been shifted at each speed. In the HLIST= submethod, all of the arrays listed in HLIST and GLIST (and optionally ELIST or PLIST) must have the same lengths (12, in this example).

In this example, there are five turbine speeds for which data is available, as pointed to in the SPEEDS array (number 1): 40000, 70000, 110000, 130000, and 150000 rpm. For the first speed (40000 rpm), the first pressure ratio array (number 201, the first in HLIST array 200), the first flow rate array (number 301, the first in GLIST array 300), and the first efficiency array (number 401, the first in ELIST array 400) are all used.

To calculate the flow rate and efficiency values at other speeds and pressure ratios than those provided, an adjustment for pressure ratio is first made. The values *along* the nearest two speeds are calculated by the program along the speed lines using linear interpolation of the points nearest the current pressure ratio. Linear extrapolation may be used for flow rate and power but not for efficiency (i.e., edge values will be used past the pressure ratio range provided).

Now that the values have been adjusted for pressure ratio, speed is taken into account. Once again the efficiency array will be interpolated linearly but will not be extrapolated (i.e., edge values will be used past the speed range provided). The flow rate arrays (and powers, if input) might be extrapolated using data points from the table cell nearest the current operating point.

In the example below, G is named as a mass flow rate in units of lb_m/sec (GU=110, MREF=2, NAF=0). Since MREF is 2, the actual values in the table (G_{table}) are not lb_m/sec but are instead corrected for a reference inlet temperature and pressure (*total* since ISTP=TS) as described in Section 3.5.17.2. For a gas or steam turbine, this method (nonzero MREF) is recommended since (1) the program can scale the results as accurately to new inlet conditions, and (2) the sonic limitations encountered at high pressure ratios and low speeds will not be problematic (as they are when using with an absolute mass flow rate, as discussed in see Section 3.5.17.7).

Efficiencies are input using the ELIST form which matches the GLIST and HLIST points.

```
HEADER FLOW DATA, TURBO ...
...
pa conn,10,1,10,dev=turbine
      spd = sturb                  $ initial speed (slow but not zero)
      speeds = 100                 $ list of speed values provided
      hlist = 200, HU=201          $ list of pressure ratio arrays
      glist = 300, GU=110          $ list of flow arrays; G in lb/sec
            mref = 2               $ corrected flow used
            tref = Tbase           $ ref state for flow
            pref = Pbase
      elist = 400                  $ list of efficiency array IDs
      ISTP = TT                    $ total to total pressure used
      af = 0.304                   $ flow area (constant)


HEADER ARRAY DATA, TURBO
...
C speeds (rpm)
C
 100 = 40000.0, 70000.0, 110000.0, 130000.0, 150000.0
C
C press ratios (HLIST)
C
 200 = 201,203,205,207,209
 201 = 1.0750, 1.2946, 1.5189, 1.7339, 1.9582, 2.1727, 2.3973,
       2.6227, 2.8445, 3.0664, 3.2882, 3.5000
 203 = 1.2000, 1.4008, 1.6117, 1.8230, 2.0340, 2.2445, 2.4555,
       3.0118, 3.2664, 3.5009, 3.7555, 4.0000
 205 = 1.4500, 1.7775, 2.0945, 2.4118, 2.7491, 3.0664, 3.3836,
       3.7227, 4.0445, 4.3664, 4.6882, 5.0000
 207 = 1.6450, 1.9551, 2.2509, 2.5664, 2.8618, 3.1773, 3.4727,
       3.7882, 4.0836, 4.3991, 4.6945, 5.0000
 209 = 2.0000, 2.2771, 2.5441, 2.8118, 3.0091, 3.3664, 3.6336,
       3.9009, 4.1882, 4.4555, 4.7227, 5.0000
C
C flow arrays (GLIST, corrected, mref=2)
C
 300 = 301,303,305,307,309
 301 = 0.0840, 0.2016, 0.2392, 0.2527, 0.2684, 0.2684, 0.2684,
       0.2684, 0.2684, 0.2684, 0.2684, 0.2684
 303 = 0.1322, 0.2054, 0.2398, 0.2535, 0.2576, 0.2559, 0.2559,
       0.2559, 0.2559, 0.2559, 0.2559, 0.2559
 305 = 0.1597, 0.2105, 0.2404, 0.2404, 0.2404, 0.2404, 0.2404,
       0.2404, 0.2404, 0.2404, 0.2404, 0.2404
 307 = 0.1891, 0.2243, 0.2333, 0.2333, 0.2333, 0.2333, 0.2333,
       0.2333, 0.2333, 0.2333, 0.2333, 0.2333
 309 = 0.1802, 0.2088, 0.2215, 0.2268, 0.2268, 0.2268, 0.2268,
       0.2268, 0.2268, 0.2268, 0.2268, 0.2268
C
```

```
C efficiencies (ELIST)
C
 400 = 401,403,405,407,409
 401 = 0.5594, 0.6923, 0.6059, 0.5492, 0.4999, 0.4672, 0.4336,
       0.4103, 0.3990, 0.3781, 0.3698, 0.3589
 403 = 0.6630, 0.7506, 0.7107, 0.6777, 0.6344, 0.6057, 0.5796,
       0.5462, 0.5222, 0.5081, 0.4973, 0.4848
 405 = 0.5632, 0.7107, 0.7376, 0.7151, 0.6872, 0.6529, 0.6330,
       0.6118, 0.5934, 0.5829, 0.5768, 0.5521
 407 = 0.5676, 0.6996, 0.7116, 0.6914, 0.6791, 0.6564, 0.6408,
       0.6220, 0.6179, 0.5962, 0.5847, 0.5717
 409 = 0.4627, 0.5709, 0.6109, 0.6278, 0.6210, 0.6116, 0.6083,
       0.5993, 0.5987, 0.5883, 0.5789, 0.5611
```

3.5.17.11 ELISTR Example

This section contains an example of the ELISTR option used as an alternative to the ELIST (or PLIST) option for full maps. Instead of being based on speeds, ELISTR is based on velocity ratios (U/C). The full inputs, which include pressure and flow specifications, are not shown below since the ELISTR method may be used with or without a full map (i.e., the GLIST= method). If a full map is elected, the ELISTR method maybe used whether or not the HLIST= or the HLIST1= submethods were selected.*

Note that each array named in the ELISTR list is a bivariate array with pairs of U/C and η values. These arrays need not be the same length, but since interpolations will be made within cells, there should be a correspondence between adjacent values in each array. In the inputs provided below, for example, each of the first pair of values in each array correspond to the same rotational speed (2684 rpm), the second pair to another rotational speed (5368 rpm), etc. Speed is not otherwise directly specified in the ELISTR method.

As an illustrative example of the interpolation methods used, assume that the current velocity ratio was U/C=0.45 current pressure ratio were 1.5 (i.e., half way between arrays 14 and 16). In this case, the value in array #14 is first interpolated to yield $\eta_{1.4}$ = (0.7762-0.7549)*(0.45-0.4242)/(0.5091-0.4242) + 0.7549 = 0.7614. But since the value of U/C=0.45 exceeds the limits in array 16 and no extrapolations are allowed, the end value of $\eta_{1.6}$ = 0.7603 is used. The returned value is interpolated by pressure ratio, yielding $\eta_{1.5}$ = 0.7608.

* If the GLIST= and HLIST= methods are selected, then the HLIST1= array must also be specified to use the ELISTR= method, in which case the program will use HLIST= for flow rate but HLIST1= for efficiency.

```
HEADER FLOW DATA, TURBEE ...
...
pa conn,3334,33,34,dev=turbine
      hlist = 1001, HU=201    $ list of pressure ratios
      elistr = 2001           $ e(PR,VR)
      ...
      dturb = 0.58            $ turbine tip diameter (in feet if UID=ENG)
      ISTP = TS               $ total to static pressure, eff used
      afi = 3.5e-4            $ inlet flow area
      afj = 4.59e-4           $ outlet flow area
c


HEADER ARRAY DATA, TURBEE
...
c pressure ratios (HLIST1)
      1001 = 1.1, 1.2, 1.3, 1.4, 1.6
c
c lists of efficiency (T-S) vs. U/Co for each pressure ratio:
      2001 = 11, 12, 13, 14, 16     $ 13 means press ratio 1.3, etc.
c
 11 = 0.1564,0.3652,  0.3127,0.5900,  0.4691,0.7650,
      0.6255,0.6983,  0.7676,0.7152,  0.9211,0.5987
 12 = 0.1138,0.2774,  0.2277,0.4933,  0.3416,0.6813,
      0.4554,0.7635,  0.5693,0.7744,  0.6831,0.7492
 13 = 0.0955,0.2432,  0.1910,0.4352,  0.2866,0.6072,
      0.3821,0.7282,  0.4776,0.7705,  0.5731,0.7759
 14 = 0.0848,0.2263,  0.1697,0.3908,  0.2546,0.5583,
      0.3394,0.6828,  0.4242,0.7549,  0.5091,0.7762
 16 = 0.0726,0.2144,  0.1451,0.3486,  0.2177,0.4896,
      0.2902,0.6172,  0.3628,0.7134,  0.4353,0.7603
```

3.5.18 COMPRESS Option

A COMPRESS connector* is a component-level model of an axial or radial compressor: a compressor whose performance varies as a function of flow rate or pressure difference ratio for a given rotational speed. While a COMPRESS connector is normally used to represent an entire compressor containing any number of stages, it may also be used to model either a single compressor stage or a radial portion of an axial compressor or stage (e.g., hub, midline, or tip). Such subset models may be combined in series (for single stages) and/or in parallel (for radial portions). For positive displacement compressors (e.g., scroll, vane, or piston), see COMPPD in Section 3.5.19.

In order to simulate such a compressor or compressor stage, the user must specify pressure difference† or ratio (H) as a function of mass flow rate (G, whether actual, corrected, or a "flow parameter") and perhaps rotational speed (SPD). The default H is unitless: a pressure ratio (whether total to total, or total to static).

The flow rate G may be a mass or volumetric flow rate, or it may be specified as a flux (per unit area), or as the flow parameter $GCF*G*T_{in}^{1/2}/P_{in}$ based on the current fluid state at the inlet, or as an effective flow based on a reference inlet state (described later).

A single curve of pressure ratio at a single speed may be specified as a function of flow, or a family of curves can be defined as a function of either actual or equivalent speed

The isentropic efficiency may be defined as a constant, or as a function of flow, or as a function of both speed and flow. As an alternative, the power added to the flow can be specified.

Figure 3-13 provides plots of one type of input acceptable to the COMPRESS utility: pressure ratio as a function of speed and mass flow (as the corrected mass flow $G*T^{1/2}/P$, where G is in kg/s in this case). Efficiency is provided separately a function of the same speeds and flows.

A specialized output routine, COMPRMAP (Section 7.11.8), is available for COMPRESS devices. COMPRMAP includes performance information as well an echo of key inputs.

3.5.18.1 Pressure-Flow Curves and Maps

There are various ways to represent the pressure-flow performance map depending on information available. As will be detailed later, "flow (G)" may actually be a mass flow rate or mass flux, a volumetric flow rate or velocity, a flow parameter corrected for inlet conditions ($T^{1/2}/P$), or a equivalent flow based on a reference state. The choice of G units and method are elected using the unit flag GU combined with the reference method MREF and flow area flag NAF. Similarly, "pres-

* The assistance of D. Mohr of D&E Propulsion & Power in designing this connector device is gratefully acknowledged.

† For pressure differences defines as "heads," note that H is by default defined in terms of the density of the working fluid itself, not water, mercury, or some other reference fluid. (Alternate units, such as "inches of water" or "mm of Hg" are also available, but are actually units of pressure, not head.) Thus, one foot of head is the weight of a column of one foot of the working fluid under one gravity.

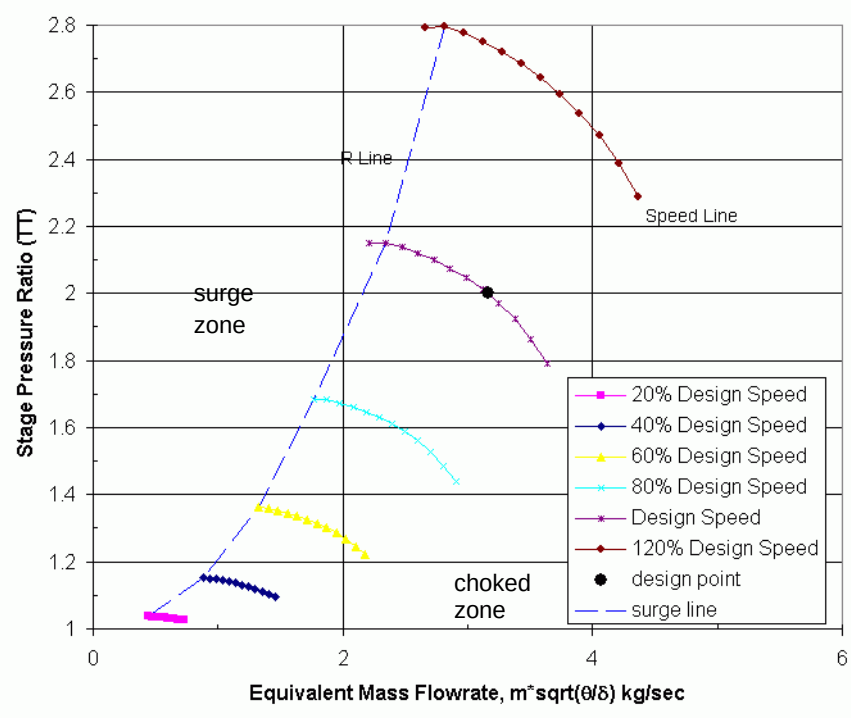


Figure 3-13 Sample Compressor Performance Map

sure (H)" may be a head or pressure difference, or a pressure ratio according to the unit flag HU, and these pressures may be optionally chosen to be based on static or total conditions using the ISTP control.

The choices for specifying the pressure-flow relationships include:

1. A single curve of pressure difference or pressure ratio versus flow rate. The pressure ratios need not decrease monotonically, but the flow values must increase monotonically.

   This method will be referred to as "the HG= method."

   The initial point is assumed to represent the surge or stall line, and the last point is assumed to represent the choke line.

2. A set of curves, each at increasing rotational speed. (If an equivalent flow method is used, this "speed" may optionally be chosen to be an equivalent speed based on a reference state. Also, if data is supplied along "R-lines" it can be transposed into equivalent data along speed lines as described later.) The pressure ratios need not decrease monotonically, but the flow values must increase monotonically.

   This method will be referred to as "the HLIST= method."

   The initial points at each speed are assumed to represent the surge or stall line, and the last points at each speed are assumed to represent the choke line. However, the surge line may be specified independently using the GSURGE or HSURGE methods.

3. If the pressure-flow maps exist as functions or are provided via calls to (perhaps COM-linked external) compressor design codes, then the current value of H and G can provided directly as G(H): flow as a function of current pressure ratio. Although data is often perceived as H(G) ... pressure ratio as a function of current flow ... SINDA/FLUINT actually requires the reverse: G(H): flow as a function of current pressure ratio. Furthermore, SINDA/FLUINT requires the slope at the current speed and pressure ratio: dG/dH, the inverse of which (dH/dG) cannot be zero (refer to Section 3.5.18.8 for modeling sonic limits).

   In other words, the user can omit performance arrays and can instead manipulate the compressor parameters directly in logic and/or expressions. The variables for H and G are HCOMP and GCOMP. The code provides the current value of HCOMP (head, pressure drop, or pressure ratio), and the user must supply the corresponding values for GCOMP (flow rate, mass flux, flow parameter, equivalent flow, etc.) and DGDH (the slope).

   This method will be referred to as “the GCOMP= method.”

Arrays (methods #1 or #2 above) should have large enough range to cover all anticipated values, especially for speed. However, under the assumption that any answer is better than none, FLUINT will extrapolate from the available information to cover any excursions outside of the defined speed range and will signal such an excursion with an indicator flag, LIMP (a translatable path output variable):

LIMP = 0: nominal (within range)
LIMP = -11 or -12: speed is too low or too high, respectively

Outside of the supplied range, most values (efficiencies being a notable exception) are extrapolated linearly from the nearest available data. Whether interpolated or extrapolated, scaling laws are applied to values of pressure ratio and power, but not to efficiency which is not extrapolated, and which is interpolated linearly.

Treatment of choking and surging is described in a separate section (Section 3.5.18.8). For completeness, the values of LIMP can be used to detect these states:

LIMP = -1 compressor is choking
LIMP = -2 compressor is surging

The pressure ratios (H) need not decrease monotonically: positive slopes (dH/dG) are allowed, but zero slopes (adjacent equal H values) are illegal. At least two data pairs are needed to define the simplest compressor curve: a straight line.

Zero Speed Compressors: Zero speed is problematic for the calculation of torque, and should be avoided if possible. If zero speed is truly required, a full-map option (the HLIST= method) should be used with one speed line input that corresponds to zero speed. Otherwise, avoid zero and negligibly small speeds. For example, if compressor start-up to 10,000 rpm is being modeled, consider starting at 100 rpm. Also, consider using LOSS or CTLVLV elements in parallel to represent the stopped condition, with logic or expressions employed to alternate between the COMPRESS model and the LOSS or CTLVLV model.

Using R-Line Map Inputs: Within each speed line, the pressure ratio (HLIST), flow (GLIST), and power or efficiency (PLIST or ELIST) values are normally chosen at increments of specific speed (Section 3.5.18.7). In other words, the second point (for example, as depicted in Figure 3-13) on each speed line will normally correspond to one specific speed, and the third point on each speed line will correspond to a slightly higher value of specific speed. Each such line of constant specific speed is frequently called an "R-line," and some compressor maps are formatted along R-lines instead of speed lines. Using this R-line map, the line of lowest specific speed usually corresponds to the surge or stall line, and the line corresponding to the highest specific speed usually corresponds to the choked flow line. Each point within an R-line is taken at a distinct rotational speed: the third point (for example) in each R-line corresponds to one rotational speed, and the fourth point on each R-line will correspond to a slightly higher value of rotational speed.

It should be evident that R-line inputs represent a transposition of data from speed-line inputs. If there are M points specified along N speed lines, then for R-line style inputs there would be N points specified along M R-lines. Therefore, if the user is presented with data in R-line format, the data will need to be transposed (rows become columns, columns become rows) to satisfy the SINDA/FLUINT input requirements. The current compressor specific speed is available as an output variable (Section 3.5.18.7), perhaps to verify that the data has been transposed correctly.

3.5.18.2 Flow Rate (G) Options

The base units for the flow rate, G, are defined using the GU parameter whose meaning is summarized in Table 3-5. However, the actual flow rate value provided in the tables, G_{table}, may deviate from the units specified by GU, G_{GU}, as described next.

First, G as provided in the tables (which corresponds to GCOMP) may be defined on an areal (flux) basis using the NAF flag. For example, NAF=1 means that the user-provided G values in the tables (or GCOMP and DGDH values if the GCOMP= method is used) are actually:

$$G_{table} = GCF*G_{GU}/AF \qquad (NAF = 1)$$

where AF is the flow area and GCF is a unit conversion and scaling factor (defaulting to 1.0), and where G_{GU} is in units specified by the units flag GU.

Second, the "G" values may represent a reference flow rate or corrected mass flow rate using the MREF flag. For example, if MREF=1 then the user-provided G values are actually understood to be:

$$G_{table} = GCF*G_{GU}*T^{1/2}/P \qquad (MREF = 1)$$

where T is the absolute inlet temperature and P is the absolute inlet pressure (in user units according to UID), and GCF is one again an available unit conversion and scaling factor that defaults to unity.

MREF can be used in combination with the NAF flag. For example, if MREF=1 and NAF=1, then the supplied "G" values are understood to be of the form:

$$G_{table} = GCF*G_{GU}*T^{1/2}/(P*AF) \qquad (MREF = 1, NAF = 1)$$

If MREF=2, then the provided "G" values are understood to include a reference state:

$$G_{table} = GCF * G_{GU} * (T/T_{ref})^{1/2} / (P/P_{ref}) \qquad \text{(MREF = 2)}$$

The above two MREF options are recommended (but not restricted) to gas flows, and nearly perfect gases as well. A third option, MREF=3, is based on reference or equivalent inlet conditions, and is *only* available for gas or steam flows:*

$$G_{table} = GCF * G_{GU} * (\varepsilon_{ref}/\varepsilon) * (V_{cr}^2/ V_{cr,ref}^2)^{1/2} / (P/P_{ref}) \qquad \text{(MREF = 3)}$$

where:

$$\varepsilon = \gamma \left\langle \frac{2}{\gamma + 1} \right\rangle^{\gamma/(\gamma - 1)}$$

$$V_{cr}^2 = \left\langle \frac{2\gamma}{\gamma + 1} \right\rangle g_c RT$$

and ε_{ref} and $V_{cr,ref}$ are based on γ_{ref} and T_{ref}, and γ is the ratio of specific heats, $\gamma = C_p/C_v$.† R is the gas constant, and g_c is the acceleration of gravity (applicable only to English units).

If MREF=3 is selected, and if a full map option is used (via the HLIST= method), the user may elect to define compressor speeds in the tables as equivalent speeds instead of absolute speeds. This selection is controlled by the NSPD flag. By default, NSPD=0 and speeds in the SPEEDS array are actual (e.g., rpm). If instead NSPD=1 (and MREF=3 and the GLIST= method is used), then the speeds in the SPEEDS array are considered to be equivalent speeds as follows:

$$N_{table} = N_{equ} = N_{actual} * (V_{cr,ref} / V_{cr}) \qquad \text{(MREF = 3, NSPD = 1)}$$

Furthermore, if MREF=3 and NSPD=1 *and* if power Q has been supplied instead of isentropic efficiency (see PLIST option in Section 3.5.18.4), then the input power in the PLIST tables is assumed to be scalable as an equivalent power as well:‡

$$Q_{table} = Q_{equ} = Q_{actual} * (V_{cr,ref} / V_{cr}) * (\varepsilon_{ref}/\varepsilon) / (P/P_{ref}) \qquad \text{(MREF = 3, NSPD = 1 and PLIST supplied)}$$

A fourth option, MREF=4, is similarly restricted to use in gas or steam flows:

$$G_{table} = GCF*G_{GU}*(RT/g_c)^{1/2}/P \qquad \text{(MREF = 4)}$$

where R is the gas constant and g_c=1 if UID=SI or g_c=32.174 if UID=ENG.

---

* If MREF=3 or MREF=4, then the code will abort if 100% liquid is upstream, and will ignore liquid in the calculation of γ and R if a two-phase state is upstream.

† The ratio P/P_{ref} is sometimes referred to as "δ," and the ratio $(V_{cr}/V_{cr,ref})^2$ is often referred to as "θ." Using this nomenclature, the equivalent speed is defined as $N_{equ} = N_{actual}/\theta^{1/2}$. If the inverted ratio $\varepsilon_{ref}/\varepsilon$ is referred to as "ε'" then $G_{equ} = G_{actual} * \varepsilon'\theta^{1/2}/\delta$. These factors ($\delta$, θ, ε) are printed in the COMPRMAP output.

‡ With P/P_{ref} referred to as "δ," and $(V_{cr}/V_{cr,ref})^2$ referred to as "θ," and the ratio $\varepsilon_{ref}/\varepsilon$ referred to as "ε'," then equivalent power is defined as $Q_{equ} = Q_{actual} *\varepsilon'/(\delta\theta^{1/2})$.

Note that GCF is not applied unless either NAF or MREF is nonzero.

3.5.18.3 Pressure (H) Options

The base units for the pressure gradient or ratio, H, are defined using the HU parameter whose meaning is summarized in Table 3-6. HU=201 is a common choice, meaning that H is unitless: a pressure ratio. It is therefore the default option. Unlike with the G options, $H_{table} = H_{HU}$ and there is no equivalent of GCF (i.e., "HCF" is not applicable to a COMPRESS device).

However, the user has control over whether up or downstream states are to be considered as static or total (stagnation), and this choice affects not only H but also the above MREF corrections and the isentropic efficiency (EFFP, TORQ, QTMK) inputs and calculations described later.

By default, all temperatures and pressures used in compressor calculations are total (if an adjacent lump uses LSTAT=STAG, then its temperature and pressure is *both* static *and* total since the velocity is assumed zero). This choice can be regulated using the ISTP factor, which can be used to change the defined inlet and outlet temperature and pressures used throughout the COMPRESS calculations. ISTP=TT (total-total, equivalent to ISTP=YES) by default, but can also be ISTP=SS (static-static, equivalent to ISTP=NO), ISTP=TS (total-static) and ISTP=ST (static-total). ISTP=TT and ISTP=TS are common designations for the compressor component. These static/total designations do not change the lump states themselves, and the designations are applied to the *defined* inlet and outlet (and hence the designations don't change if flow rate reverses).

3.5.18.4 Defining Compressor Efficiencies or Power

Compressor isentropic efficiencies are optional but should be specified or at least estimated (typical values are between 0.5 and 0.8, for example). Efficiencies are used to calculate power added (Section 3.5.18.5) and torque (Section 3.5.18.6), and may be specified as either total-total or total-static (matching the pressure method described above via the ISTP parameter).

There are four ways to specify isentropic (or "power") efficiency, η, or alternatively, the extracted power:

1. as a single value, EFFP (updated in logic or expressions)

   This method will be referred to as "the EFFP= method."

2. as an interpolated array of isentropic efficiency vs. flow rate (G).

   This method will be referred to as "the AEFFP= method."

3. if a full map of H-G has been provided via the HLIST= method, then a full map of isentropic efficiencies can be provided in parallel (e.g, full maps of η at the same operating points as the head-flow relationships).

   This method will be referred to as "the ELIST= method."

4. if a full map of H-G has been provided via the HLIST= method, then a full map of added or absorbed power (W or BTU/hr, depending on UID) can be provided in parallel (e.g, full maps of power at the same operating points as the head-flow relationships). In this case, EFFP becomes an output variable.

   This method will be referred to as “the PLIST= method.”

Note that compressor power efficiency, EFFP, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.).

If arrays have been used to define EFFP (e.g., methods 2-4 above, but not method 1), then the isentropic efficiency can be modified or scaled before use, using additive and multiplying terms, EFFA and EFFM, respectively, as follows:

$$EFFP_{final} = EFFM*EFFP_{table} + EFFA$$

EFFM defaults to 1.0, and EFFA defaults to 0.0, so no manipulation of EFFP takes place by default. EFFM and EFFA may be used for various purposes, including modifying the degradation in performance when the compressor is choking or surging (Section 3.5.18.8). EFFM and EFFA can also be used as unknowns when correlating to test data.

3.5.18.5 QTMK and QTM: Turbomachinery Power

Even for a perfectly efficient compressor (EFFP=1.0 by default), a thermal power of QTMK (a translatable path output variable) will be added to the QTM (a translatable lump output variable) of the lump currently downstream of the compressor. QTM is one component (along with QL and the sums of tie QTIEs and ftie QFs) of the total lump power input QDOT.

Each lump may have only one unique QTM. Therefore, if more than one COMPRESS or other turbomachine is placed in parallel, the lump QTM will represent to total or net power added by all turbomachinery currently exhausting to that lump.

The value of QTMK is proportional to the change in enthalpy from the defined inlet to the defined outlet pressure, holding entropy constant, times the absolute value of the mass flow rate, divided by the isentropic efficiency.

Compressor calculations override the rotating path (ROTR etc.) calculations for QTMK and TORQ (Section 3.5.15.4), and apply factors such as TCF and EFFP independently of analogous path rotation options; compressor options generally take precedence over path rotation options.

3.5.18.6 Hydraulic Torque

The code calculates the hydraulic torque as:

$$TORQ = TCF*QTMK/SPD$$

where TCF is input by the user to perform whatever unit conversions are necessary. TCF defaults to unity.

TORQ is a translatable path output variable. In SI units, the TCF contains the conversion of rpm (assuming those are the units of SPD) to rad/sec, resulting in TORQ units of N-m:

```
TCF = 60./(2.*pi)       $ “pi” is a built-in constant
```

In US Customary units, QTMK is in BTU/hr, so assuming SPD is in RPM, to get TORQ in lb_f-ft use:

```
TCF = 778./(2.*pi*60.)        $ 778 converts BTU/hr to lbf-ft/hr
```

Hydraulic torque may be used along with co-solved ordinary differential equations (e.g., DIFFEQ1) to calculate the current speed of the compressor during transients, or may be used to solve a torque balance equation for steady-states (e.g., for turbobrayton cycles or turbochargers), perhaps using ROOT_FIND. Note that TORQ is positive for energy into the fluid system. It is therefore usually positive for compressors, and the negative of TORQ (usually a positive number) should be set proportional to $d\omega/dt$, the rate of change of shaft speed.

Torque may also be used to calculate power required by the compressor. If TORQ is in English units of lb_f-ft, for example, and speed is in rpm, then the horsepower extracted can be calculated as:

$$HP = -TORQ*SPD*2\pi/60.0/550.0$$

Compressor power input may also be taken directly from the output lump’s QTM (in Watts if UID=SI and in BTU/hr if UID=ENG), noting that other turbomachinery may be contributing to that QTM as well.

Note that the compressor torque coefficient, TCF, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.). Similarly, the TORQ calculations are independent of rotating path calculations.

3.5.18.7 Specific Speed

The current specific speed for the compressor is available as an output variable, CNS, defined as follows:

$$CNS = SSCF*SPD*GCOMP^{1/2}/\Delta h^{3/4}$$

In the above definition, SSCF is a user-provided unit conversion factor which defaults to 1.0, SPD is the current (not equivalent) speed, GCOMP is the current flow rate in units consistent with GU, GCF, MREF, and NAF (in other words, in the same units as any table values), and Δh is the ideal minimum increase in enthalpy across the compressor (e.g., at constant entropy, as if EFFP were unity). The Δh term has units of either J/kg or BTU/lb_m depending on UID.

In “lazy” or quasi-non-dimensional English units, where SPD is in rpm, GCOMP is in ft^3/sec (GU=101, NAF=0, MREF=0,2, or 3 ... but not 1, and GCF=1.0), then:

```
SCCF  = 778.**(-0.75) $ converts BTU to ft-lbf for traditional units
```

3.5.18.8 Modeling Surging and Choking

At excessively high or low specific speeds, the compressor performance may be limited by internal choking or surging (stalling).

Choking: The general-purpose flow passage methods for detecting and modeling choked flow are inappropriate for handling the case of a choked compressor, and are therefore disabled (MCH=0) by default. Instead, choking simulation should be contained within the performance maps themselves. Otherwise, *by default the highest flow points at each speed line will be assumed to represent the choke line.*[*] For example, any points to the right of the plotted data in Figure 3-13 represent the choked zone. In this zone, LIMP = -1 will be reported by the code (along with printed warnings), and the user can optionally choose to degrade the *isentropic* efficiency using EFFM and/or EFFA.

Hydrodynamically, the code will represent the choked compressor as a nearly vertical line in the H-G plot (such as is represented in Figure 3-13). In other words, the flow will not increase significantly if the pressure ratio decreases in this zone.

This built-in treatment of choking will begin once the user-provided pressure-flow maps have been exceeded. Therefore, the values at the edge of the tables might represent the choke line itself. However, this does not preclude the user from extending the pressure-flow map to include choked behavior within the map itself. The software does not “second guess” or attempt to interpret data provided by the user, so the only difference between choking within the map and choking beyond the map is whether or not LIMP is nonzero. For the same reason, when choking is included in the map but the map is exceeded, the software will invoke the built-in treatment of choking as a nearly vertical line.

Surging: By default, *the lowest flow points at each speed line are assumed to represent the surge line.* For example, any points to the left of the plotted data in Figure 3-13 represent the surge zone.[†] In this zone, LIMP = -2 will be reported by the code (along with printed warnings), and the user can optionally choose to degrade the *isentropic* efficiency using EFFM and/or EFFA.

Hydrodynamically, the code will represent the surging compressor as a nearly horizontal line in the H-G plot (such as is represented in Figure 3-13). In other words, the flow will increase significantly even if the pressure ratio decreases only slightly in this zone. It should be noted that this treatment does not represent any physically-based process: it exists purely to yield stable numeric treatment either to permit “bounces” out of the design map during steady-states, or to survive long enough to produce warnings during transients.

Positively sloped (negative DGDH) regions of the curves are permitted, and hence the user may elect to contain surge modeling within the map itself. However, note that positively sloped regions are numerically intrinsically unstable.

---

* If either H or G at the end of the curve is negative (or H is less than unity for HU=201), then choking is ignored since the curve or performance map extends past the primary quadrant and is therefore assumed to include choking behavior. LIMP will still report the exceeding of any H or G limits.

† If either H or G at the end of the curve is negative (or H is less than unity for HU=201), then surging is ignored since the curve or performance map extends past the primary quadrant and is therefore assumed to include surging behavior. LIMP will still report the exceeding of any H or G limits.

This built-in treatment of surging will begin once the user-provided pressure-flow maps have been exceeded. Therefore, the values at the edge of the tables might represent the surge line itself. However, this does not preclude the user from extending the pressure-flow map to include surge or stall behavior within the map itself. The software does not "second guess" or attempt to interpret data provided by the user, so the only difference between surging within the map and surging beyond the map is whether or not LIMP is nonzero. For the same reason, when surging is included in the map but the map is exceeded, the software will invoke the built-in treatment of surging as a nearly horizontal line with a negative slope (positive DGDH). This might cause a conflict with a user-provided map that includes surge behavior, especially if that zone of the map has a positive slope (negative DGDH). Therefore, the user should either exclude the surge zone from the map, or else include a large region of surge data to avoid conflict with the built-in surge zone methods.

For some small compressors at low speeds, the surge line is encountered at larger G values than a constant specific speed line (R-line) would indicate. For this reason, the user can independently specify a surge line, either as a function of flow (GSURGE) or pressure ratio (HSURGE), with one surge point for every speed in the SPEEDS array. In other words, these surge lines can "cut off" the lower left hand portion of the compressor H-G map, triggering the code to consider the compressor as surging before it has reached the data points that were provided within the cut-off region.

In other words, three possibilities exist for modeling surging: (1) the edge of the map (lowest flows at each speed) represents the surge line, (2) the surge line is contained within the map and the data to the left of that line is provided by the user to model the surge event, or (3) the surge line is contained within the map itself and is delineated by the user via the GSURGE or HSURGE options, which essentially tell the software to ignore portions of the map and to instead invoke its own simplistic surge treatment. *Note that the software itself does not distinguish between cases #1 and #2.*

Note that isentropic efficiency (EFFP) is not extrapolated beyond the edge of the provided table (or beyond the optionally defined surge lines). This means that, by default, the efficiency at the surge line will be assumed to exist throughout the surge regime. If the performance map has not been specifically extended to include the surge zone (that is, if the edge of the map represents the surge line or if HSURGE or GSURGE has been used), then the user is strongly encouraged to add EFFP degradations using perhaps the EFFM and/or EFFA correction factors (Section 3.5.18.4).

For example, to affect a 50% degradation if surging, EFFM could be defined in FLOW DATA as:

```
effm = (limp#this==-2)? 0.5 : 1.0
```

If the current surge flow rate were known (say, as register *Gsur*), a more gradual degradation would be more acceptable numerically:

```
effm = (limp#this==-2)? max(0.01, Gcomp#this/Gsur) : 1.0
```

3.5.18.9 FLOW DATA Format

COMPRESS Format in CONN subblocks:

```
... 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]
```

where:

GU..... Flow rate unit identifier: see Table 3-5. These units will apply to G values. See also MREF and NAF for further modifiers of these basic units.

HU..... Head unit identifier: see Table 3-6. These units will apply to H values.

GCF ..... A G conversion factor (defaulting to unity) that is active if either NAF or MREF is nonzero. *GCF is ignored if MREF=NAF=0.*

ISTP ..... If ISTP=YES (or equivalently, ISTP=TT), then pressures and other state variables are defined on the basis of total or stagnation conditions, and either AF or both AFI and AFJ are required inputs. By default ISTP=TT, and so both inlet and outlet states are based on total (stagnation) conditions and therefore flow areas (AF or both AFI and AFJ) are required. ISTP=TS (total-static) and ISTP=ST (static-total) are also available as options. ISTP also affects the meaning of TREF and PREF.

NAF ..... Integer mass flux basis flag. By default NAF=0, and hence G values provided are assumed to be based on the units specified by GU. NAF=1 instead means that the G values are actually mass fluxes based on the flow area AF, with the parameter GCF available for conversions or scaling. In other words, for NAF=1 the values supplied as "G" are actually to be interpreted as G_{table} = GCF*G_{GU}/AF where G_{GU} is determined by the unit selector GU. For NAF=2, G_{table} = GCF*G_{GU}/AFI and for NAF=3, G_{table} = GCF*G_{GU}/AFJ. (Note "G_{table}" is equivalent to "GCOMP" when using the GCOMP= method.)

MREF. . . . . Integer mass flow reference flag. By default MREF=0, and hence G values provided are assumed to be based on the units specified by GU. MREF=1 instead means that the G values are actually corrected mass flows: the values supplied as "G" are actually to be interpreted as $G_{table} = GCF*G_{GU}*T^{1/2}/P$ where T and P are the absolute inlet temperature and pressure, respectively (static or total, depending on ISTP), and G_{GU} is determined by the unit selector GU. Absolute means the units of T are either K or R (depending on UID), and the units of P are either psia or absolute Pascal. GCF may be used for unit conversions or scaling.

For MREF=2, $G_{table} = GCF*G_{GU}*(T/T_{ref})^{1/2}/(P/P_{ref})$ where the reference temperature and pressure (TREF, PREF) are specified separately. For MREF=3, a more complicated correction is made for equivalent conditions (see main text) using a reference temperature and pressure and also a reference ratio of specific heats, γ_{ref} (GREF).

MREF=4 invokes a slightly more complicated variation of MREF=1 that includes gas properties and unit conversions: $G_{table} = GCF*G_{GU}*(RT/g_c)^{1/2}/P$ where R is the gas constant, and g_c=1 if UID=SI or g_c=32.174 if UID=ENG.

MREF may be used in combination with NAF.

(Note "G_{table}" is equivalent to "GCOMP" when using the GCOMP= method.)

TREF. . . . . The reference temperature (K or R, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

PREF. . . . . The reference pressure (absolute Pascal or psia, depending on UID, and static or total, depending on ISTP) used for MREF=2 or MREF=3.

GREF. . . . . The reference ratio of specific heats, $\gamma_{ref} = C_p/C_v$, used for MREF=3.

HG . . . . . ID of doublet array (in this submodel) containing a single compressor head-flow curve: pressure ratio or difference (units according to HU) as a function of G (flow rate: units according to GU). H need not decrease monotonically, but *no two adjacent H values may be equal (i.e., the slope dH/dG cannot be zero: the inverse of this slope, DGDH, cannot be infinite).* The array pointed to must contain real values or expressions of the form G1,H1, G2,H2 ...

Excursions will be flagged as nonzero LIMP values, and either choking or surging will be simulated if the provided pressure ratios or flow rates are exceeded.

Note that this "HG= method" is just one way of specifying the H-G relationship. See also SPEEDS/HLIST/GLIST for the "HLIST= method," and GCOMP/DGDH for the "GCOMP= method."

Supplied G values may be adjusted per the rules invoked via the MREF and NAF variables, and if so they might be scaled by the parameter GCF.

SPD . . . . . Current/initial compressor speed, in consistent user units. Avoid very small or zero values (the default!) unless the model/inputs are specifically designed for this case. In most cases, zero SPD is equivalent to a closed valve

for consistency. Note that SPD is in actual units (e.g., rpm) and is not dependent on the value of NSPD, unlike the values in the SPEEDS array. SPD should be specified even if using the GCOMP= method, since SPD is used for TORQ calculations.

GCOMP..... for compressors whose head-flow is defined by expressions and/or user logic (i.e., by the "GCOMP= method" and not by arrays via either the HG= or the HLIST= method), the flow rate or flow coefficient at the current head, HCOMP, in units consistent with the combined effects of GU, MREF, NAF, and GCF. If H-G arrays are input via the HG= or the HLIST= method, then this becomes an output.*

DGDH ..... for compressors defined by the GCOMP= method (i.e., no arrays used), the slope (flow rate or flow coefficient to head or head coefficient) at the current head, HCOMP, in units consistent with GU, HU, MREF, NAF, and GCF. Must not be zero if MREF=0, and negative values are legal but can cause instabilities in some models if persistent. If H-G arrays are input via the HG= or the HLIST= method, then this becomes an output.

SPEEDS .... A full map alternative to the single-curve "HG= method," SPEEDS is the array ID containing a list of speeds for which pressure and flow curves (and possibly efficiencies) will be specified via the HLIST, GLIST, and optionally the ELIST or PLIST arrays (see "Full Map Example" below). Like HG, SPEEDS is an array ID, and the entries in that array must be real numbers or expressions in the same units as SPD if NSPD=0, or equivalent speeds if NSPD=1.

SPEEDS is also a required input if either the GSURGE or the HSURGE option is used.

Refer to full-map example at the bottom of this subsection.

NSPD ..... Flag indicating whether the values in the SPEEDS array are in actual speed such as rpm (NSPD=0, the default) or whether they has been specified as equivalent speed (NSPD=1, *valid only if MREF=3*) as described in the main text. Note that NSPD has no effect on the current speed, SPD, which is *always* in actual speed units (e.g., rpm).

NSPD=1 also governs the scaling of power if provided by the PLIST option: both speed and power will be scaled if NSPD=1 (see formulas in main text, Section 3.5.18.2).

HLIST..... If SPEEDS is used for a full map (the "HLIST= method"), this is the ID of a SINDA/FLUINT array (in this submodel) containing the values of pressure ratio H (or head or pressure drop, per HU) for which table values in GLIST correspond. HLIST is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing pressure ratio (H) values that correspond to each of the N speeds set in the SPEEDS input. HLIST is a list of lists. In

* Note that the calculated value of GCOMP will differ from a value calculated using the current FR except for a converged steady-state. FR is the current flow rate, whereas GCOMP (updated after FLOGIC 0) is the current flow that the compressor *would* be at if the pressure ratio or head were constant at the current HCOMP value. The value of GCOMP can be compared directly with the output variable GFR, the value of the current flow rate in the same units as GCOMP.

other words, the first array ID in the HLIST array should contain a series of pressure ratio values at the first speed in the SPEEDS array. The first array listed should also correspond to the first array of GLIST (and optionally, ELIST), and the first value in the first HLIST array should correspond to the first value in the first GLIST array, and so forth. All such arrays within HLIST and GLIST (and ELIST or PLIST, if used), should be the same length. H need not decrease monotonically, but *no two adjacent H values may be equal.*

GLIST . . . . . The ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing flow values that correspond to each of the N speeds set in the SPEEDS input. GLIST is a list of lists. In other words, the first array ID in the GLIST array should contain a series of flow rate values at the first speed in the SPEEDS array. *GLIST values within each array must increase monotonically.*

The array values themselves should be in units consistent with GU. Supplied G values may be adjusted per the rules invoked via the MREF and NAF variables, and if so they might be scaled by the parameter GCF.

GSURGE . . . . . The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of G (flow) values, corresponding to the speeds in the SPEEDs array, which collectively represent the surge or stall line. If the program encounters flows below these values, even if table values are available for lower flows, it will mark the compressor as surging with appropriate warnings, LIMP values (LIMP=1), and simulation techniques as described above. Either GSURGE or HSURGE can be used, or neither (in which case the table limit is interpreted as the surge line), but not both.

HSURGE . . . . . The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of H (pressure ratio) values, corresponding to the speeds in the SPEEDs array, which collectively represent the surge or stall line. If the program encounters pressure ratios above these values, even if table values are available for higher pressure ratios, it will mark the compressor as surging with appropriate warnings, LIMP values (LIMP=1), and simulation techniques as described above. Either GSURGE or HSURGE can be used, or neither (in which case the table limit is interpreted as the surge line), but not both.

SSCF. . . . . A unit conversion factor (defaulting to unity) that applies to the output variable CNS (current compressor specific speed), as described in the main text.

EFFP. . . . . Overall compressor isentropic efficiency (real, between 0.0 and 1.0). This is an output if either ELIST, PLIST, or AEFFP arrays are provided. The efficiency basis is a function of ISTP, and therefore may be total-total or total-static, for example.

AEFFP . . . . . ID of the doublet array containing the isentropic efficiency as a function of flow rate G. AEFFP is of the form G_1, η_1, G_2, η_2 .... where flow rates are monotonically increasing (i.e., $G_2 > G_1$).

End values will be used if the provided range is exceeded; no extrapolations are made for this variable.

ELIST..... The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing compressor isentropic efficiencies that correspond to the N speeds set in the SPEEDS input. ELIST is a list of lists. In other words, the first array ID in the ELIST array should contain a series of efficiency values (between zero and one) at the first speed in SPEEDS. The values in that first *array* should correspond to the first *array* of GLIST. Edge values are used if the provided range is exceeded; no extrapolations are made for this variable.

PLIST..... As an alternative to setting isentropic efficiency, the absorbed power itself can be specified using PLIST. The optional PLIST input is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing compressor hydraulic powers that correspond to the N speeds set in the SPEEDS input. *Power units must be either Watts or BTU/hr, depending on UID.* PLIST is a list of lists. In other words, the first array ID in the PLIST array should contain a series of absorbed (positive) power values at the first speed in SPEEDS. Unlike the alternative ELIST option, extrapolations are made for this variable past the limits, and scaling with speed is assumed.

If MREF=3 and NSPD=1, then the powers supplied in the PLIST arrays are considered to be “equivalent powers” and will be scaled with changes in the inlet conditions per the formula in the main text (Section 3.5.17.2).

EFFM ..... Multiplier to use for EFFP values extracted from array options (ELIST, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

EFFA ..... Additive term to use for EFFP values extracted from array options (ELIST, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

TCF ..... Correction factor for the COMPRESS output variable TORQ (see main text).

AF..... flow area for kinetic energy and choking (if MCH is nonzero), and for calculating dynamic head if ISTP=YES or ISTP=TS or ISTP=ST. This input is not required if AFI and AFJ are specified instead. AF might be calculated from the prior PA DEF value of DH if no AF has been supplied.

AFTH ..... throat area for choking calculations (see Section 3.5.17.7).

AFI ..... inlet (suction) flow area, if not constant. If AFI is specified, AFJ must also be specified, and AF will be calculated automatically.

AFJ ..... outlet (discharge) flow area, if not constant.

UPF ..... weighting factor for fluid properties: 1.0 = base H and G on inlet (suction) density, 0.0 = base H and G on outlet (discharge) density, if appropriate.

defaults:

GU . . . . . 0 (m^3/s if UID=SI, else ft^3/hr of working fluid if UID=ENG)
HU . . . . . 201 (pressure ratio, see also ISTP: total pressure ratio by default)
GCF . . . . . 1.0 (ignored unless either NAF or MREF is nonzero)
ISTP. . . . . YES (pressures and all other properties are based on total values), same as ISTP=TT (total-total)
MREF. . . . . 0 (G in units of GU, unmodified)
NAF . . . . . 0 (G is absolute flow, not a flux)
SPD . . . . . 0.0 (zero speed usually means the compressor acts like a closed valve)
NSPD. . . . . 0 (SPEEDS in units of actual speed, not equivalent speed)
GCOMP . . . . . none (calculated if H-G arrays input using either the HG= or the HLIST= method)
DGDH. . . . . none (calculated if H-G arrays input using either the HG= or the HLIST= method)
EFFP. . . . . 1.0 (calculated if AEFFP, ELIST, PLIST is used; independent of PA DEF defaults)
EFFM. 1.0
EFFA. 0.0
SSCF. 1.0
TCF . . . . . 1.0 (independent of PA DEF defaults)
AF . . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if neither AF nor AFI/AFJ have been specified, then no kinetic energy can be extracted nor can choking calculations applied, nor can the default ISTP=YES be used. In other words, if *both* ISTP and AF are defaulted, an error will occur.)

AFTH. . . . . AF, or AFI if both AFI and AFJ are used instead of AF
AFI . . . . . -1.0 (use AF)
AFJ . . . . . -1.0 (use AF)
UPF . . . . . 1.0 (inlet properties). Note that this default UPF value (unlike that for tubes and STUBE connectors) does not depend on prior PA DEF blocks.
MCH . . . . . 0 (no choking: contained within maps)

Simple Examples:

```
PA CONN, 10,1,2, FR=10.0, DEV=COMPRESS, HG=1, AF=Acomp
PA CONN,1,111,112, DEV=COMPRESS
      HG=114, GU=4 $ Units of L/s
      ISTP = TS $ total-static for all properties
      EFFP = 0.8, SPD = 20000., AF = pi*(RO^2 - RI^2)
```

A full map example is provided below.

Caution: Some of the “volumetric flow rate (G)” units are really mass flow rate units, and most of the “head (H)” units are not length, but rather pressure. Despite the name, “inches of water” has units of pressure, not length or head, for example.

Caution: HCOMP and GCOMP represent a point on the current (or extrapolated) compressor curve, whereas GCOMP can differ from the current flow (as designated by GFR, an output variable corresponding to the current mass flow rate FR but in the same units as GCOMP). If corrections are required that are based on the current flow rate, use GFR, not GCOMP.

Caution: Leaving MCH=0 as the default is strongly recommended (see Section 3.5.18.8).

Guidance: References to processor variables within expressions may refer to “#this” as the path identifier, “#up” as the identifier of the *defined* upstream lump (does not change with flow reversal), and “#down” as the identifier of the *defined* downstream lump.

Guidance: Unlike MREF=1 and MREF=4, which provide tools for *accepting* a specific style of performance map, MREF=2 and MREF=3 (and perhaps NSPD=1) can be thought of as tools for *extending* the range of a map created at a single inlet state (temperature, pressure, gas species) to other inlet conditions.

Guidance: A summary of common pressure-flow rate input options is provided below, and summarized in Table 3-9:
1) *The point/slope method.* Specify the GCOMP value corresponding to the current program-supplied HCOMP, along with the slope at that point: DGDH in units per the GU and HU specifications.
2) *A single curve of H vs. G at one speed.* Specify the HG array (usually, pressure ratio as a function of volumetric flow) and SPD. The G values may be generalized for various input states using the MREF option (and perhaps NAF and GCF options). Use of AEFFP is recommended, but a single value of isentropic efficiency (EFFP) can be supplied if available.
3) *A full map of H and G.* Specify the SPEEDS, HLIST, and GLIST lists along with the current/initial speed SPD. If MREF=3, the speeds in SPEEDS may be equivalent instead of actual by specifying NSPD=1. Use of a corresponding ELIST is easiest if available, but isentropic efficiency can also be specified as a function of flow (G) using the AEFFP option.

3.5.18.10 Full Map Example: HLIST= Method

This section contains an example of the “HLIST= method:” a full map of pressure ratios and efficiencies as functions of speed and flow rate, along with a short description of the mathematical treatment of these maps and the resulting requirements for their structure.

Table 3-9 Summary of COMPRESS Input Options

Head-Flow Method	Efficiency (or Power) Method	Other notes
Single Point/Slope (GCOMP= and DGDH=)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ...	SPD is still needed to calculate TORQ
Single Curve (HG=)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ...	SPD is still needed to calculate TORQ
Full Map (HLIST=) (GLIST=) (SPEEDS=)	EFFP=η or AEFFP=$G_1,\eta_1 G_2,\eta_2$ ... or ELIST = ... or PLIST = ...	The SPEEDS and GLIST inputs are required with the HLIST= method. The values in the SPEEDS array may be actual (NSPD=0, the default) or equivalent (NSPD=1), while the current speed SPD is always actual (not equivalent).

For the full map example listed below, note that the COMPRESS connector lists for HLIST, GLIST, and ELIST are “lists of lists:” lists of array IDs that point to the arrays containing the actual data in 1D strings. In this example, 6 speeds are used, with 12 values (pressure ratio, flow, and efficiency) provided at each speed. Thus, the map is a full 6x12 grid. In the HLIST= submethod, all of the arrays listed in HLIST and GLIST (and optionally ELIST or PLIST) must have the same lengths (12, in this example).

In this example, there are six compressor speeds for which data is available, as pointed to in the SPEEDS array (number 1000): 16000, 32000, 48000, 64000, 80000, and 96000 rpm. For the first speed (16000 rpm), the first pressure ratio array (number 816, the first in HLIST array 800), the first flow rate array (number 916, the first in GLIST array 900), and the first efficiency array (number 716, the first in ELIST array 700) are all used.

To calculate the flow rates and efficiency values at other speeds and pressure ratios than those provided, linear interpolations and extrapolations are made. However, extrapolation is never used for efficiency (i.e., edge values will be used past the range provided). If PLIST had been supplied instead of ELIST, not only would the powers be extrapolated, but both the extrapolations and interpolations would be nonlinear with speed: powers are estimated using the function $QTMK = A*SPD^B$, where A and B are estimated local curve fit parameters, with B normally equal to about 3.

Extrapolations (and in some cases interpolations) are not made in the surge and choked zones. The choked zone is represented by any flow or pressure ratio past the ends of the arrays. By default, the surge zone is represented by any flow or pressure ratio past the start of the arrays, but in this case the first points in the highest two speed lines are outside the surge zone, as can be evidenced by the nonmonotonic progression of H (pressure ratio) in arrays 880 and 896.* The fact that these first points lie outside the range of the normal operating zone is indicated using the optional HSURGE input, which lists limiting pressure ratios at each speed in SPEEDS. (GSURGE could be used instead. It denotes the surge line using flow rates instead of pressure ratios at each speed.)

* The location of the surge line within this particular compressor map is artificial: it has been added for illustrative purposes only and does not represent any physical limitation in an actual compressor.

In the example below, G is named as a mass flow rate in units of kg/sec (GU=13, MREF=2, NAF=0). GCF is left to default to unity. Since MREF is 2, the actual values in the table (G_{table}) are not kg/sec but are instead corrected for a reference inlet temperature and pressure (*total* since ISTP=TS) as described in Section 3.5.18.2. This method (nonzero MREF, preferably MREF=3) is recommended since the program can scale the results as accurately to new inlet conditions.

Efficiencies are input using the ELIST method which matches the GLIST and HLIST points.

```
HEADER FLOW DATA, COMPTIME, ...
...
PA CONN,3,31,32, DEV=COMPRESS, SPD = 30000.0          $ initial speed
      SPEEDS = 1000                                   $ in RPM
      MREF = 2, TREF = 400.0, PREF = 2.07E6           $ reference state
      ISTP = TT                                       $ total-total
      TCF = 60./(2.*pi)                               $ Torque in N-m
      HLIST = 800, GLIST = 900, GU=13, ELIST = 700    $ HU=201 default
      HSURGE = 8001                                   $ or GSURGE = 9001
```

```
HEADER ARRAY DATA, COMPTIME
C
C SPEEDS (RPM)
C
      1000 = 16000., 32000., 48000., 64000., 80000., 96000.
C
C GLIST
C
            900 = 916, 932, 948, 964, 980, 996
 916= 0.4427,0.4686,0.4944,0.5203,0.5462,0.5720,0.5979,0.6238,
      0.6496,0.6755,0.7014,0.7273
 932= 0.8854,0.9371,0.9888,1.0406,1.0923,1.1441,1.1958,1.2476,
      1.2993,1.3510,1.4028,1.4545
 948= 1.3280,1.4057,1.4833,1.5609,1.6385,1.7161,1.7937,1.8713,
      1.9489,2.0266,2.1042,2.1818
 964= 1.7707,1.8742,1.9777,2.0812,2.1847,2.2881,2.3916,2.4951,
      2.5986,2.7021,2.8056,2.9090
 980= 2.2134,2.3428,2.4721,2.6015,2.7308,2.8602,2.9895,3.1189,
      3.2482,3.3776,3.5069,3.6363
 996= 2.6561,2.8113,2.9665,3.1218,3.2770,3.4322,3.5874,3.7427,
      3.8979,4.0531,4.2083,4.3636
C
C HLIST
C
            800 = 816, 832, 848, 864, 880, 896
 816= 1.0368,1.0361,1.0354,1.0346,1.0337,1.0327,1.0317,1.0306,
      1.0294,1.0280,1.0266,1.0251
 832= 1.1527,1.1503,1.1473,1.1439,1.1402,1.1359,1.1310,1.1253,
      1.1189,1.1116,1.1035,1.0943
 848= 1.3623,1.3580,1.3520,1.3446,1.3361,1.3263,1.3149,1.3014,
      1.2855,1.2667,1.2450,1.2200
 846= 1.6855,1.6823,1.6725,1.6608,1.6469,1.6298,1.6105,1.5877,
      1.5602,1.5269,1.4868,1.4393
 880= 2.1502,2.1515,2.1371,2.1195,2.0987,2.0745,2.0454,2.0113,
      1.9718,1.9237,1.8643,1.7912
 896= 2.7917,2.7943,2.7785,2.7514,2.7202,2.6841,2.6425,2.5944,
      2.5366,2.4697,2.3889,2.2874
```

```
C
C T-T Efficiency
C
            700 = 716, 732, 748, 764, 780, 796
 716= 0.8343,0.8374,0.8379,0.8365,0.8330,0.8270,0.8181,0.8059,
      0.7900,0.7699,0.7452,0.7156
 732= 0.8361,0.8412,0.8426,0.8415,0.8376,0.8299,0.8177,0.7999,
      0.7758,0.7442,0.7046,0.6561
 748= 0.8326,0.8399,0.8445,0.8454,0.8433,0.8377,0.8273,0.8106,
      0.7862,0.7526,0.7084,0.6521
 764= 0.8242,0.8335,0.8403,0.8447,0.8463,0.8437,0.8378,0.8268,
      0.8087,0.7811,0.7422,0.6898
 780= 0.8119,0.8222,0.8305,0.8369,0.8412,0.8430,0.8412,0.8354,
      0.8249,0.8070,0.7787,0.7371
 796= 0.7974,0.8076,0.8165,0.8239,0.8298,0.8336,0.8351,0.8336,
      0.8276,0.8166,0.7982,0.7682
C
C HSURGE (pressure ratio vs. speed)
C
      8001 = 1.0368,1.1527,1.3623,1.6855,2.1515,2.7943
C
C GSURGE (NOT USED SINCE HSURGE USED)
C
      9001 = 0.4427,0.8854,1.3280,1.7707,2.3428,2.8113
```

3.5.19 COMPPD Option

A COMPPD connector* is a component-level model of a piston, rotary or reciprocating vane, screw, or scroll compressor. These machines differ from variable-displacement compressors (see COMPRESS in Section 3.5.18) in that a fixed volume of working fluid is, in principle, delivered with each revolution. While a COMPPD connector is normally used to represent an entire compressor containing any number of stages, it may also be used to model a single compressor stage. Such subset models may be combined in series for multi-stage compressors.

In order to simulate such a compressor or compressor stage, the user must specify a volumetric efficiency, η_v, as a function of pressure difference† or ratio (H) and perhaps rotational speed (SPD). The default H is unitless: a pressure ratio (whether total to total, or total to static). Using the parameter name EFFV for η_v, and using DISP as the displacement volume per revolution, the volumetric flow rate is then calculated as:

G = GCF*EFFV*DISP*SPD

with GCF being a unit conversion factor that is specified by the user (and which defaults to unity).

A single curve of volumetric efficiency (at a single speed) may be specified as a function of pressure ratio, or a family of such curves can be defined as a function of speed. A single (constant) value of EFFV may also be specified, perhaps as adjusted in logic or expressions.

The "power efficiency" (*isentropic* efficiency, EFFP) may be defined as a constant, or as a function of pressure ratio, or as a function of both speed and pressure ratio. As an alternative, the power added to the flow can be specified.

A specialized output routine, COMPPDMAP (Section 7.11.8), is available for COMPPD devices. COMPPDMAP includes performance information as well an echo of key inputs.

3.5.19.1 Pressure Ratio vs. Volumetric Efficiency Maps

The base units for the flow rate, G, are defined using the GU parameter whose meaning is summarized in Table 3-5. Normally, G should have units of volumetric flow rate for COMPPD devices.

The program calculates G (available in logic or expressions as "GCOMP‡") as:

G = GCF*EFFV*DISP*SPD

---

* The assistance of D. Mohr of D&E Propulsion & Power in designing this connector device is gratefully acknowledged.

† For pressure differences defines as "heads," note that H is by default defined in terms of the density of the working fluid itself, not water, mercury, or some other reference fluid. (Alternate units, such as "inches of water" or "mm of Hg" are also available, but are actually units of pressure, not head.) Thus, one foot of head is the weight of a column of one foot of the working fluid under one gravity.

‡ Note that the calculated value of GCOMP will differ from a value calculated using the current FR except for a converged steady-state. FR is the current flow rate, whereas GCOMP (updated after FLOGIC 0) is the current flow that the compressor *would* be at if the current EFFV value (and SPD and DISP values) stay constant.

The user supplies, directly or indirectly, all of the values on the right hand side of the above equation. EFFV is the volumetric efficiency (normally unitless), DISP is the volumetric displacement per revolution (normally in units of volume), and SPD is the shaft or rotor speed (normally in RPM). GCF is a unit conversion factor also supplied by the user such that the units of G match those specified via GU.

For example, if GU=1 (m^3/sec), if DISP is in units of cc, if EFFV is unitless, and if SPD is in RPM, then GCF = 1.0e-6/60. If instead DISP were supplied in units of cubic meters, then GCF=1/60 to convert RPM to revolutions per second.

If GU had specified mass flow units instead of volumetric flow units, then GCF would contain the inlet density (perhaps using “DL#up”).

In summary, the units of EFFV, DISP, and SPD are unknown to the code, and can be any consistent set as long as GCF provides the conversions into the units specified by GU. (SPD should be consistent with TCF, an analogous factor used in torque calculations as described in Section 3.5.19.6).

Similarly, “pressure (H)” may be a head or pressure difference, or a pressure ratio according to the unit flag HU, and these pressures may be optionally chosen to be based on static or total conditions using the ISTP control, as described in Section 3.5.19.3.

3.5.19.2 Specifying Volumetric Efficiency

The choices for specifying the pressure-flow relationships, via volumetric efficiency, include:

1. A single curve of volumetric efficiency versus speed.

   This method will be referred to as “the AEFFV= method.”

2. A set of curves, each at increasing rotational speed. Within each speed curve, heads (pressure ratios) must increase monotonically.

   This method will be referred to as “the ELISTV= method.”

3. If the pressure-flow maps exist as functions or are provided via calls to (perhaps COM-linked external) compressor design codes, then the current value of volumetric efficiency can provided directly as the parameter EFFV.

   In other words, the user can omit performance arrays and can instead manipulate the compressor parameters directly in logic and/or expressions. The variable for H is HCOMP. The code provides the current value of HCOMP (head, pressure drop, or pressure ratio), and the user must supply the corresponding values for EFFV. When using this method, the inlet fluid should not be an incompressible liquid, otherwise this COMPPD device will exhibit MFRSET-like insensitivity to pressure changes during the next solution interval.

   This method will be referred to as “the EFFV= method.”

If arrays have been used to define EFFV (e.g., methods 1 or 2 above, but not method 3), then the volumetric efficiency can be modified or scaled before use, using additive and multiplying terms, EFFVA and EFFVM, respectively, as follows:

$$EFFV_{final} = EFFVM*EFFV_{table} + EFFVA$$

EFFVM defaults to 1.0, and EFFVA defaults to 0.0, so no manipulation of EFFV takes place by default. EFFVM and EFFVA may be used for various purposes, including modeling degradations in performance. EFFVM and EFFVA can also be used as unknowns when correlating to test data.

Arrays (methods #1 or #2 above) should have large enough range to cover all anticipated values, especially for speed. No extrapolations will be made: edge values will be used beyond the table or array limits. FLUINT will signal such an excursion with an indicator flag, LIMP (a translatable path output variable):

LIMP = 0: nominal (within range)
LIMP = -1 or -2: pressure ratio or difference is too low or too high, respectively
LIMP = -11 or -12: speed is too low or too high, respectively

Zero Speed Compressors: Zero speed is problematic for the calculation of torque, and should be avoided if possible. If possible, avoid zero and negligibly small speeds. For example, if compressor start-up to 1000 rpm is being modeled, consider starting at 10 rpm. Also, consider using LOSS or CTLVLV elements in parallel to represent the stopped condition, with logic or expressions employed to alternate between the COMPPD model and the LOSS or CTLVLV model.

3.5.19.3 Pressure (H) Options

The base units for the pressure gradient or ratio, H, are defined using the HU parameter whose meaning is summarized in Table 3-6. HU=201 is a common choice, meaning that H is unitless: a pressure ratio. It is therefore the default option. Unlike with the G options, $H_{table} = H_{HU}$ and there is no equivalent of GCF (i.e., “HCF” is not applicable to a COMPPD device).

However, the user has control over whether up or downstream states are to be considered as static or total (stagnation), and this choice affects not only H but also the isentropic efficiency (EFFP, TORQ, QTMK) inputs and calculations described later.

By default, all temperatures and pressures used in compressor calculations are total (if an adjacent lump uses LSTAT=STAG, then its temperature and pressure is *both* static *and* total since the velocity is assumed zero). This choice can be regulated using the ISTP factor, which can be used to change the defined inlet and outlet temperature and pressures used throughout the COMPRESS calculations. ISTP=TT (total-total, equivalent to ISTP=YES) by default, but can also be ISTP=SS (static-static, equivalent to ISTP=NO), ISTP=TS (total-static) and ISTP=ST (static-total). ISTP=TT and ISTP=TS are common designations for the compressor component. These static/total designations do not change the lump states themselves, and the designations are applied to the *defined* inlet and outlet (and hence the designations don’t change if flow rate reverses).

3.5.19.4 Defining Compressor Efficiencies or Power

Compressor isentropic efficiencies are optional but should be specified or at least estimated (typical values are between 0.3 and 0.6, for example). Efficiencies are used to calculate power added (Section 3.5.19.5) and torque (Section 3.5.19.6), and may be specified as either total-total or total-static (matching the pressure method described above via the ISTP parameter).

There are four ways to specify isentropic (or "power") efficiency, η_p, or alternatively, the extracted power:

1. as a single value, EFFP (updated in logic or expressions)

   This method will be referred to as "the EFFP= method."

2. as an interpolated array of isentropic efficiency vs. speed.

   This method will be referred to as "the AEFFP= method."

3. if a full map of *volumetric* efficiency EFFV has been provided via the ELISTV= method, then a full map of *isentropic* efficiencies can be provided in parallel (e.g, full maps of η_p at the same operating points as the pressure-η_v relationships).

   This method will be referred to as "the ELIST= method."

4. if a full map of *volumetric* efficiency EFFV has been provided via the ELISTV= method, then a full map of added or absorbed power (W or BTU/hr, depending on UID) can be provided in parallel (e.g, full maps of power at the same operating points as the pressure-η_v relationships). In this case, EFFP becomes an output variable.

   This method will be referred to as "the PLIST= method."

   Outside of the supplied range, scaling laws are applied to values of power (PLIST), unlike other variables which are interpolated linearly.

Note that compressor power efficiency, EFFP, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.).

If arrays have been used to define EFFP (e.g., methods 2-4 above, but not method 1), then the isentropic efficiency can be modified or scaled before use, using additive and multiplying terms, EFFA and EFFM, respectively, as follows:

$$EFFP_{final} = EFFM*EFFP_{table} + EFFA$$

EFFM defaults to 1.0, and EFFA defaults to 0.0, so no manipulation of EFFP takes place by default. EFFM and EFFA may be used for various purposes, including modeling degradations in performance. EFFM and EFFA can also be used as unknowns when correlating to test data.

3.5.19.5 QTMK and QTM: Turbomachinery Power

Even for a perfectly efficient compressor (EFFP=1.0 by default), a thermal power of QTMK (a translatable path output variable) will be added to the QTM (a translatable lump output variable) of the lump currently downstream of the compressor. QTM is one component (along with QL and the sums of tie QTIEs and ftie QFs) of the total lump power input QDOT.

Each lump may have only one unique QTM. Therefore, if more than one COMPPD or other turbomachine is placed in parallel, the lump QTM will represent to total or net power added by all turbomachinery currently exhausting to that lump.

The value of QTMK is proportional to the change in enthalpy from the defined inlet to the defined outlet pressure, holding entropy constant, times the absolute value of the mass flow rate, divided by the isentropic efficiency.

Compressor calculations override the rotating path (ROTR etc.) calculations for QTMK and TORQ (Section 3.5.15.4), and apply factors such as TCF and EFFP independently of analogous path rotation options; compressor options generally take precedence over path rotation options.

3.5.19.6 Hydraulic Torque

The code calculates the hydraulic torque as:

TORQ = TCF*QTMK/SPD

where TCF is input by the user to perform whatever unit conversions are necessary. TCF defaults to unity.

TORQ is a translatable path output variable. In SI units, the TCF contains the conversion of rpm (assuming those are the units of SPD) to rad/sec, resulting in TORQ units of N-m:

```
TCF = 60./(2.*pi)       $ “pi” is a built-in constant
```

In US Customary units, QTMK is in BTU/hr, so assuming SPD is in RPM, to get TORQ in lb_f-ft use:

```
TCF = 778./(2.*pi*60.)        $ 778 converts BTU/hr to lbf-ft/hr
```

Hydraulic torque may be used along with co-solved ordinary differential equations (e.g., DIFFEQ1) to calculate the current speed of the compressor during transients, or may be used to solve a torque balance equation for steady-states (e.g., for turbobrayton cycles or turbochargers), perhaps using ROOT_FIND. Note that TORQ is positive for energy into the fluid system. It is therefore usually positive for compressors, and the negative of TORQ (usually a positive number) should be set proportional to $d\omega/dt$, the rate of change of shaft speed.

Torque may also be used to calculate power required by the compressor. If TORQ is in English units of lb_f-ft, for example, and speed is in rpm, then the horsepower extracted can be calculated as:

HP = -TORQ*SPD*2π/60.0/550.0

Compressor power input may also be taken directly from the output lump's QTM (in Watts if UID=SI and in BTU/hr if UID=ENG), noting that other turbomachinery may be contributing to that QTM as well.

Note that the compressor torque coefficient, TCF, cannot be defaulted from PA DEF blocks since it has a distinct meaning from that of rotating paths (ROTR etc.). Similarly, the TORQ calculations are independent of rotating path calculations.

3.5.19.7 FLOW DATA Format

COMPPD Format in CONN subblocks:

```
... 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]
```

where:

- `GU` ..... Flow rate unit identifier: see Table 3-5. These units will apply to the calculated GCOMP value.
- `HU` ..... Head unit identifier: see Table 3-6. These units will apply to H values.
- `GCF` ..... A G conversion or scaling factor (defaulting to unity) which is applied to the calculated flow rate: GCOMP = GCF*EFFV*DISP*SPD.
- `ISTP` ..... If ISTP=YES (or equivalently, ISTP=TT), then pressures and other state variables are defined on the basis of total or stagnation conditions, and either AF or both AFI and AFJ are required inputs. By default ISTP=TT, and so both inlet and outlet states are based on total (stagnation) conditions and therefore flow areas (AF or both AFI and AFJ) are required. ISTP=TS (total-static) and ISTP=ST (static-total) are also available as options.
- `SPD` ..... Current/initial compressor speed, in consistent user units. Avoid very small or zero values (the default!) unless the model/inputs are specifically designed for this case. In most cases, zero SPD is equivalent to a closed valve for consistency. SPD is used for TORQ calculations.
- `DISP` ..... Volumetric displacement of the compressor, usually in units of volume (per rotation), but any units can be used as long as the resulting G is consistent with the inputs GCF, GU, SPD, and EFFV (which is normally unitless).

EFFV. . . . . For compressors whose pressure-flow is defined by expressions and/or user logic (i.e., by the "EFFV= method" and not by arrays via either the AEFFV= or the ELISTV= method), then EFFV is the volumetric efficiency at the current head, HCOMP. GCOMP will then be calculated in units consistent with the combined effects of GU and GCF. If the volumetric efficiency is input using arrays (AEFFV= or ELISTV= methods), then EFFV becomes an output.

AEFFV . . . . . ID of the doublet array containing the *volumetric* efficiency as a function of the actual (or equivalent) speed N. AEFFV is of the form N_1, η_{v1}, N_2, η_{v2} .... where speeds are monotonically increasing (i.e., $N_2 > N_1$).
End values will be used if the provided range is exceeded; no extrapolations are made for this variable.

ELISTV . . . . . The ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing *volumetric* efficiency values that correspond to each of the N speeds set in the SPEEDS input. ELISTV is a list of lists. In other words, the first array ID in the ELISTV array should contain a series of volumetric efficiency values at the first speed in the SPEEDS array.
The array values themselves are normally unitless, but otherwise should be in units consistent with GU, GCF, SPD, and DISP.

EFFVM . . . . . Multiplier to use for EFFV values extracted from array options (ELISTV or AEFFV). Does not apply to user-supplied EFFV.

EFFVA . . . . . Additive term to use for EFFV values extracted from array options (ELISTV or AEFFV). Does not apply to user-supplied EFFV.

SPEEDS . . . . . A full map alternative to the single-curve "AEFFV= method," SPEEDS is the array ID containing a list of speeds for which pressure and flow curves (and possibly efficiencies) will be specified via the HLIST, ELISTV, and optionally the ELIST or PLIST arrays (see "Full Map Example" below). SPEEDS is an array ID, and the entries in that array must be real numbers or expressions in the same units as SPD.
Refer to full-map examples at the bottom of this subsection.

HLIST . . . . . If SPEEDS is used for a full map (the "ELISTV= method"), this is the ID of a SINDA/FLUINT array (in this submodel) containing the values of pressure ratio H (or head or pressure drop, per HU) for which table values in ELISTV correspond. HLIST is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing pressure ratio (H) values that correspond to each of the N speeds set in the SPEEDS input. HLIST is a list of lists. In other words, the first array ID in the HLIST array should contain a series of pressure ratio values at the first speed in the SPEEDS array. The first array listed should also correspond to the first array of ELISTV (and optionally, ELIST), and the first value in the first HLIST array should correspond to the first value in the first ELISTV array, and so forth. All such arrays within HLIST and ELISTV (and ELIST or PLIST, if used), should be the same length. *H values must increase monotonically,*

and no two adjacent H values may be equal.

EFFP . . . . . Overall compressor isentropic efficiency (real, between 0.0 and 1.0). This is an output if either ELIST, PLIST, or AEFFP arrays are provided. The efficiency basis is a function of ISTP, and therefore may be total-total or total-static, for example.

AEFFP. . . . . ID of the doublet array containing the *isentropic* efficiency as a function of the actual (or equivalent) speed N. AEFFP is of the form N_1, η_{p1}, N_2, η_{p2} .... where speeds are monotonically increasing (i.e., $N_2 > N_1$).

End values will be used if the provided range is exceeded; no extrapolations are made for this variable.

ELIST. . . . . The optional ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing compressor *isentropic* efficiencies that correspond to the N speeds set in the SPEEDS input. ELIST is a list of lists. In other words, the first array ID in the ELIST array should contain a series of isentropic efficiency values (between zero and one) at the first speed in SPEEDS. The values in that first *array* should correspond to the first *array* of ELISTV. Edge values are used if the provided range is exceeded; no extrapolations are made for this variable.

PLIST. . . . . As an alternative to setting isentropic efficiency, the absorbed power itself can be specified using PLIST. The optional PLIST input is the ID of a SINDA/FLUINT array (in this submodel) which itself contains a list of N IDs of SINDA/FLUINT arrays (also in this submodel) containing compressor hydraulic powers that correspond to the N speeds set in the SPEEDS input. *Power units must be either Watts or BTU/hr, depending on UID.* PLIST is a list of lists. In other words, the first array ID in the PLIST array should contain a series of absorbed (positive) power values at the first speed in SPEEDS. Unlike the alternative ELIST option, extrapolations are made for this variable past the limits, and scaling with speed is assumed.

EFFM . . . . . Multiplier to use for EFFP values extracted from array options (ELIST, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

EFFA . . . . . Additive term to use for EFFP values extracted from array options (ELIST, AEFFP, or PLIST). Does not apply to user-supplied EFFP.

TCF . . . . . Correction factor for the COMPPD output variable TORQ (see main text).

AF. . . . . flow area for kinetic energy and choking (if MCH is nonzero), and for calculating dynamic head if ISTP=YES or ISTP=TS or ISTP=ST. This input is not required if AFI and AFJ are specified instead. AF might be calculated from the prior PA DEF value of DH if no AF has been supplied.

AFTH . . . . . throat area for choking calculations (see Section 3.5.17.7).

AFI . . . . . inlet (suction) flow area, if not constant. If AFI is specified, AFJ must also be specified, and AF will be calculated automatically.

AFJ . . . . . outlet (discharge) flow area, if not constant.

UPF . . . . . weighting factor for fluid properties: 1.0 = base H and G on inlet (suction) density, 0.0 = base H and G on outlet (discharge) density, if appropriate.

defaults:

GU . . . . . .0 (m^3/s if UID=SI, else ft^3/hr of working fluid if UID=ENG)
HU . . . . . .201 (pressure ratio, see also ISTP: total pressure ratio by default)
GCF 1.0
ISTP. . . . . .YES (pressures and all other properties are based on total values), same as ISTP=TT (total-total)
SPD . . . . . .0.0 (zero speed usually means the compressor acts like a closed valve)
EFFV. . . . . .1.0 (calculated if the AEFFV= or ELISTV= method is used)
EFFVM 1.0
EFFVA 0.0
EFFP. . . . . .1.0 (calculated if AEFFP, ELIST, PLIST is used; independent of PA DEF defaults)
EFFM. 1.0
EFFA. 0.0
TCF . . . . . .1.0 (independent of PA DEF defaults)
AF . . . . . .-1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if neither AF nor AFI/AFJ have been specified, then no kinetic energy can be extracted nor can choking calculations applied, nor can the default ISTP=YES be used. In other words, if *both* ISTP and AF are defaulted, an error will occur.)
AFTH. . . . . .AF, or AFI if both AFI and AFJ are used instead of AF
AFI . . . . . .-1.0 (use AF)
AFJ . . . . . .-1.0 (use AF)
UPF . . . . . .1.0 (inlet properties). Note that this default UPF value (unlike that for tubes and STUBE connectors) does not depend on prior PA DEF blocks.
MCH . . . . . .0 (no choking: contained within maps)

Simple Examples:

```
PA CONN, 10,1,2, FR=10.0, DEV=COMPPD, AEFFV=1, AF=Acomp
PA CONN,1,111,112, DEV=COMPPD
      GU=4   $ Units of L/s
      EFFV = 0.9
      ISTP = TS $ total-static for all properties
      EFFP = 0.6, SPD = 20000., AF = pi*(RO^2 - RI^2)
```

A full map example is provided below.

Caution: Some of the “volumetric flow rate (G)” units are really mass flow rate units, and most of the “head (H)” units are not length, but rather pressure. Despite the name, “inches of water” has units of pressure, not length or head, for example.

Caution: Leaving MCH=0 as the default is strongly recommended.

Guidance: References to processor variables within expressions may refer to "#this" as the path identifier, "#up" as the identifier of the *defined* upstream lump (does not change with flow reversal), and "#down" as the identifier of the *defined* downstream lump.

Guidance: A summary of common pressure-flow rate input options is provided below, and summarized in Table 3-10:

1) *The point method.* Specify the EFFV (*volumetric* efficiency) value corresponding to the current program-supplied HCOMP. The *isentropic* or *power* efficiency (EFFP) can be specified by point or by array.

2) *A single curve of H vs.* η_v *at one speed.* Specify the AEFFV array (isentropic efficiency vs. speed) and the current SPD. The flow rate (G) values are specified indirectly via the η_v values. Use of AEFFP is recommended, but a single value of *isentropic* efficiency (EFFP) can be supplied if available.

3) *A full map of H and* η_v. Specify the SPEEDS, HLIST, and ELISTV lists along with the current/initial speed SPD. Use of a corresponding ELIST is easiest if available, but *isentropic* efficiency can also be specified as a function of flow using the AEFFP option.

Table 3-10 Summary of COMPPD Input Options

Head-Flow Method	**Efficiency (or Power) Method**	**Other notes**
Single Point (EFFV=)	EFFP=η_p or AEFFP=N_1,$\eta_{p1}$$N_2$,$\eta_{p2}$ ...	SPD and DISP are required
Single Curve (AEFFV=)	EFFP=η_p or AEFFP=N_1,$\eta_{p1}$$N_2$,$\eta_{p2}$ ...	SPD and DISP are required
Full Map (HLIST=) (ELISTV=) (SPEEDS=)	EFFP=η_p or AEFFP=N_1,$\eta_{p1}$$N_2$,$\eta_{p2}$ ... or ELIST = ... or PLIST = ...	The SPEEDS and ELISTV inputs are required with the HLIST= method. DISP must be specified.

3.5.19.8 Full Map Example: ELISTV= Method

This section contains an example of the "ELISTV= method:" a full map of volumetric and isentropic efficiencies as functions of speed and pressure ratio, along with a short description of the mathematical treatment of these maps and the resulting requirements for their structure.

For the full map example listed below, note that the COMPPD connector lists for HLIST, ELISTV, and ELIST are "lists of lists:" lists of array IDs that point to the arrays containing the actual data in 1D strings. In this example, 7 speeds are used, with 8 values (pressure ratio, volumetric efficiency, and isentropic efficiency) provided at each speed. Thus, the map is a full 7x8 grid. In the ELISTV= submethod, all of the arrays listed in HLIST and ELISTV (and optionally ELIST or PLIST) must have the same lengths (8, in this example).

In this example, there are six compressor speeds for which data is available, as pointed to in the SPEEDS array (number 100): 300, 700, 1000, 1500, 2000, 3000, and 4000 rpm. For the first speed (300 rpm), the first pressure ratio array (number 201, the first in HLIST array 200), the first volumetric efficiency array (number 301, the first in ELISTV array 300), and the first isentropic efficiency array (number 401, the first in ELIST array 400) are all used.

In fact, in this map the same set of pressure ratios (H) is used at every speed line. Thus, the same array (#201) is listed repeatedly in array #200. (This usage is equivalent to the HLIST1 option that is available for turbines, but not for compressors.)

To calculate the efficiency values at other speeds and pressure ratios than those provided, linear interpolations are made. Extrapolation is never used for efficiency (i.e., edge values will be used past the range provided). If PLIST had been supplied instead of ELIST, not only would the powers be extrapolated, but both the extrapolations and interpolations would be nonlinear with speed: powers are estimated using the function $QTMK = A*SPD^B$, where A and B are estimated local curve fit parameters, with B normally equal to about 2 or 3.

In the example below, G is named as a volumetric flow rate in units of m^3/sec (GU=1). GCF is set to 1/60 to convert minutes (in SPD, which is in rpm) into seconds, and the displacement DISP is set to cubic meters (such that no further conversions are needed in GCF).

Efficiencies are input using the ELIST method which matches the ELISTV and HLIST points.

Note that the efficiencies, whether volumetric or isentropic, are not allowed to exceed 1.0 nor to be less than or equal to zero: 0.01 has been used as a lower limit.

```
HEADER FLOW DATA, COMPTOO, ...
...
PA CONN, 412,401,402, DEV=COMPPD, SPD = 3000.0
      DISP = 20.e-6           $ 20cc (UID=SI, so input in m3)
      GU = 1                  $ m3/sec
      GCF = 1.0/60.0          $ convert RPM to RPS
      SPEEDS = 100
      ELISTV = 300, HLIST = 200, ELIST = 400
      ISTP = YES
      TCF = 60./(2.*pi)       $ Torque in N-m
...
```

```
HEADER ARRAY DATA, COMPTOO
C
C SPEEDS =
C
      100 = 300., 700., 1000., 1500., 2000., 3000., 4000.
C
C HLIST = (the same at all speeds, so just repeat the same array number)
C
      200 = 201,201,201,201,201,201,201
      201 = 1.0, 1.5, 2.0, 4.0, 6.0, 8.0, 10.0, 15.0
C
C ELISTV =
C
            300 = 301,302,303,304,305,306,307
 301 = 1.00,  1.00,  0.9579,0.3625,0.01,  0.01,  0.01,  0.01
 302 = 1.00,  1.00,  1.00,  0.7877,0.5163,0.2450,0.01,  0.01
 303 = 1.00,  1.00,  1.00,  0.8511,0.6526,0.4542,0.2557,0.01
 304 = 1.00,  1.00,  1.00,  0.8593,0.7176,0.5758,0.4341,0.0797
 305 = 0.9951,0.9667,0.9384,0.8250,0.7116,0.5982,0.4848,0.2013
 306 = 0.8413,0.8201,0.7988,0.7137,0.6287,0.5437,0.4586,0.2460
 307 = 0.6875,0.6698,0.6521,0.5812,0.5103,0.4395,0.3686,0.1914


C
C ELIST =
C
            400 = 401,402,403,404,405,406,407
 401 = 0.01,  0.01,  0.2196,1.00,  1.00,  0.8878,0.6524,0.01
 402 = 0.01,  0.3744,0.6073,0.8629,0.8481,0.7658,0.6564,0.3356
 403 = 0.1986,0.5146,0.6595,0.8112,0.7918,0.7296,0.6503,0.4221
 404 = 0.3643,0.5643,0.6555,0.7487,0.7331,0.6903,0.6366,0.4834
 405 = 0.3637,0.5335,0.6118,0.6966,0.6898,0.6602,0.6214,0.5085
 406 = 0.1962,0.3914,0.4847,0.6027,0.6188,0.6093,0.5896,0.5225
 407 = 0.01,  0.2092,0.3377,0.5141,0.5554,0.5629,0.5570,0.5184
```

3.6 Ties (Convection)

Heat transfer calculations in FLUINT may be effected in several ways. First, the user may calculate heat transfer rates and manipulate lump QL (and perhaps nodal Q) values as needed within the user logic blocks or expressions. This method is appropriate in constant heat flux environments where the energy exchange rates are independent of fluid flow, but can be difficult to implement in other cases because of the potential for instabilities. Alternatively, the user may opt to let FLUINT calculate heat transfer automatically using *ties*. The remainder of this section describes how ties are used.

Ties are special conductor-like elements used to link fluid submodels with thermal submodels. Specifically, a tie always extends between a FLUINT lump and a SINDA node to permit energy interchange. In general, the lump represents the fluid within a container or pipe segment, and the node represents the inner wall of the container. With one minor exception,* any number of ties may exist between any lump and any node.

Ties are unique because they are the only network element that can connect fluid submodels to thermal submodels, and they can only be used for that purpose. Although they are very similar to intermodel thermal conductors, ties are strictly fluid network elements and always “belong” to the fluid submodel in which they were input. In order to tie lumps in two *different* fluid submodels† together (to model perhaps a thin-walled heat exchanger), there must exist an intermediate thermal node that represents perhaps the separating wall. Also, note that a constant wall temperature condition can be simulated by tying a lump to a boundary node; a constant heat flux condition is modeled with the QL term of the lump, and no ties are needed.

Ties can be dynamically moved from one location to another using the PUTTIE routine.‡ At every time step or steady-state iteration, the user has the opportunity of moving a tie to any other node and/or to any other lump within the same fluid submodel. One use for such “movable ties” is illustrated in Sample Problem F. Some types of ties (namely HTNS, HTUS, and CAPPMP HTM ties described below) move automatically.

When used with twinned tanks to model phasic nonequilibrium (Section 3.25), ties themselves can also be twinned. Normally, the secondary twin is ignored unless the lump itself is twinned and currently active.** If ties attach to twinned tanks and use twinned paths to calculate convection, the tie itself *must* be twinned. When a twinned tie attaches to a twinned tank, the primary tie attaches to the primary (liquid) tank, and the secondary tie attaches to the secondary (vapor) tank.

---

* CAPPMP junctions cannot support more than the one tie that is optionally generated by that macrocommand.

† To tie two lumps in the *same* submodel together, use fties (Section 3.7).

‡ Use of PUTTIE is discouraged because it cannot be supported by Sinaps, FloCAD, and restart operations. Use multiple ties as an alternative, turning them on and off using zero/unity DUPN and DUPL or UAM values.

** As will be noted below, ITAC=1 HTU and HTP ties are an exception to this statement, and will continue to work in a twinned mode even if the attached lump is homogeneous (meaning either untwinned or twinned but currently operating in the mixed mode). This allows user (or FloCAD Compartment) descriptions of the phase-specific heat transfer routes to be developed and preserved for each phase.

Guidance: The lump output variable QDOT for each lump reports the total heat rate on that lump, which is the sum of QL plus the effects of any ties that are currently active: QDOT = QL + ΣQTIE + ΣQF + QTM + QCHEM, where QTIE is the heat rate through a tie. This section deals only with the QTIE term. Fties (QF), chemical (QCHEM), and turbomachine (QTM) heat loads and powers are described elsewhere.

Guidance: If the thermal submodel containing the node is not currently active (i.e., the submodel is not listed in the current BUILD statement), then tie will be adiabatic: the node and lump will not exchange heat energy.

Restriction: If the thermal submodel containing the node is never active (i.e., the thermal submodel is *never* listed in *any* BUILD statement and neither is the BUILD ALL used), then the code will abort with preprocessing error. If for some reason the tie is purposely meant to remain off during the whole run, add a "fake" BUILD statement containing the required thermal submodels at the end of OPERATIONS. If this BUILD statement is placed after all calls to solution routines, then the thermal submodel will be pre-processed (avoiding the above error message) yet will never be used for calculations.

3.6.1 Basic Tie Parameters

All ties have a UA value analogous to the thermal conductor G value. This "conductance" is either calculated by the program or supplied by the user, depending on the option used. UA has the same units as does the G of a thermal conductor, and operates in an analogous fashion for most ties (HTUS ties are an exception, as noted below):

$$QTIE_{tie} = UAM_{tie} \cdot UA_{tie} \cdot (T_{node} - TEF_{tie})$$

where:

$QTIE_{tie}$. . . Heat rate through tie, positive *into* lump and *out of* node
UAM_{tie} . . Overall scaling factor (multiplier, defaults to unity)*
UA_{tie} . . . . Heat transfer conductance
T_{node}. . . . . Node temperature
TEF_{tie}. . . . Effective fluid temperature (usually the same as the lump temperature TL)

When a tie is made to a lump, the QTIE value is subtracted from the nodal source term (Q) for all ties on that node. (Actually, during steady-states the QTIE is summed but ignored by the node, which treats the adjacent lump as if it were another node.) Similarly, the QTIE value is summed into the lump source term QDOT for all ties on that lump, augmenting any user inputs for QL and any

* UAM is calculated automatically for FloCAD Compartment ties based on each bay's orientation and design margin factors.

turbomachinery or rotational power (QTM), chemical power (QCHEM), and ftie energy transfer (QF). Thus, energy appears in one network, exactly canceling the energy that disappears in the other network.*

Analogous to path duplication factors (DUPI and DUPJ, as described in Section 3.12), tie duplication factors DUPL and DUPN are available. DUPN is the factor applied to the node end of the tie, and DUPL is the factor applied to the lump end. The node "sees" DUPN such ties, and the lump "sees" DUPL such ties. Thus, the value added to the lump QDOT is actually DUPL•QTIE, while the value subtracted from the nodal Q is actually DUPN•QTIE.

Tie duplication factors function similar to path duplication factors, and in fact are intended to allow the user to deal with levels of duplication created by path factors that are not reflected in the thermal model. Tie duplication factors may be used to mimic the treatment of SINDA one-way conductors: zero duplication factors on either end cause the equivalent behavior of a one-way conductor. *Zero duplication factors on both ends effectively eliminate the tie, and equal values other than unity can be used as a multiplying factor to degrade heat transfer coefficients or to parametrically test the sensitivity of convection correlations, or to use as a correlation factor using perhaps the Solver (Section 5) to match available test data.* (UAM is a preferred method for performing overall scaling of heat transfer tie conductance.) Of course, duplication factors allow more general usage than do one-way conductors.

The user is cautioned that energy and mass can be created and lost via unequal path and tie duplication factors, and that although complicated nested levels of duplication are purposely valid, the results can be somewhat confusing and the corresponding models difficult to debug. Use of the mapping routines QMAP, FLOMAP, NODMAP, and LMPMAP is recommended to help clarify the results. These routines show how energy and mass flow through the network.

In summary, all ties have at least five descriptive parameters:

UA . . . . . the heat transfer conductance
UAM . . . . the heat transfer conductance multiplier (scaling factor)
QTIE . . . . the heat rate through the tie
DUPL. . . . the lump-side duplication factor
DUPN . . . the node-side duplication factor

* Actually, the heat rate may not be exactly equal to the UA times the temperature difference, especially in STEADY or when junctions and/or arithmetic nodes are tied. This is because the heat rates on those elements affect their temperatures (or states) immediately, and because the heat rates are held constant over each solution interval to conserve energy. Another exception is HTUS ties, which differ from the above equation because of augmentation of heat transfer due to upstream effects, as described below.

3.6.2 Tie Types and Classifications

Ties are subdivided into seven types: *HTN*, *HTNC*, *HTNS*, *HTP*, *HTU*, *HTUS*, and *HTM*. The first four types represent convective heat transfer options. The fifth and sixth types (HTU and HTUS) represent user-defined heat transfer processes. The last type is generated by certain models and macrocommands.

HTN . . . . . Convection heat transfer, forced convection
HTNC . . . . Convection heat transfer, centered lump
HTNS . . . . Convection heat transfer, segment-oriented version of HTN
HTP. . . . . Convection heat transfer, quasi-stagnant and pool boiling
HTU . . . . . User-described heat transfer
HTUS . . . . User-described heat transfer, segment-oriented version of HTU
HTM . . . . . Macro-specific heat transfer (CAPPMP)

HTM ties, which are currently only generated by CAPPMP macros, operate according to rules specific to that macro, which is documented in later sections.

Ties may be divided into subtypes according to whether (1) the tie conductance (UA) value is provided by the user or calculated by the program, and (2) whether the lump is intended to completely represent the bulk fluid or merely the downstream end of an isothermal-wall heat exchanger segment.

The first distinction is clearly made. With HTN, HTNC, HTNS, and HTP ties, the UA value is calculated automatically by the program according to convective heat transfer in a duct (see Appendix B and Section 7.11.3), and the user must name one or more *adjacent paths* (tubes or STUBE connectors*) to define the duct. With HTU and HTUS ties, the user provides the UA coefficient.

The second distinction is more subtle. HTN, HTNC, HTP, and HTU ties calculate heat transfer assuming that the lump represents the bulk fluid properties, and that the node represents the average wall temperature. This *lumped parameter* formulation implicitly assumes downstream-weighted heat transfer, since the lump state changes according to the heat transfer rate, and only that state is used in calculating heat transfer. Adjacent paths are used only to provide flow rate and heat transfer area data; whether they are upstream or downstream of the lump is irrelevant. While these methods are robust and are recommended for flows with phase change, they tend to underpredict heat transfer rates slightly for flows without phase change, especially when the resolution is too coarse. Also, they tend to conflict with the way many engineers treat a single-phase heat transfer problem: as a constant wall temperature over a segment that has distinct inlet and outlet states.

HTNS and HTUS ties, which will be referred to as *segment* ties, treat the heat transfer problem differently than do lumped parameter ties. Instead of being lump-oriented, they are path-oriented. The node is assumed to represent the average wall temperature over the length of the path, the lumps on either end of that path are assumed to represent the upstream and downstream states of that segment. Unless the user specifies otherwise, *these ties will automatically jump to the lump that is currently downstream of the specified path when flows reverse.*

* REDUCER and EXPANDER connectors can also be named, but do not contribute any heat transfer area to the tie.

Guidance: ***While HTNS ties are preferred in the limit of steady single-phase or flows without phase change, HTNC ties are recommended for unsteady flows with phase change or where flow conditions are not known a priori.***

Sample Problem H helps illustrate the difference between these two classes of heat transfer ties.

The remainder of this section describes each type of tie in more detail.

3.6.3 Lump-oriented (Lumped Parameter, Finite Difference) Ties

This section describes HTU, HTN, HTNC, and HTP ties. All of these ties operate under the assumption that the lump represents the average fluid state and the node represents the average wall temperature independent of flow direction or upstream conditions. For HTU ties, the user provides a UA value either directly or indirectly. For HTN, HTNC, and HTP ties, the program calculates the UA value, and the user identifies the path(s) (always tubes and STUBE connectors*) which define(s) the size, shape and flow rate of the heat transfer section represented by the node and lump.

3.6.3.1 HTU Ties

When using an HTU tie, the UA conductance must be provided by the user, perhaps as calculated in the FLOGIC 0 block or as input using an expression. No path or flow rate is associated with an HTU tie. Thus, HTU ties may be used to represent almost any heat transfer phenomena including natural convection, developing velocity profiles, pool boiling, grooved evaporation or condensation, and jet impingement (see, for example, Section 7.11.4).

While such phenomena are the purpose of HTU ties, they may also be used to override the standard convection correlation set used in the routine STDHTC, as documented in Section 7.11.3. Instead of using HTU ties for alternate convection correlations, however, it is recommended that the user instead use HTN, HTNC, and HTP ties (described below) and use the customization factors (e.g, CDB, XNUL, XNTM, CSF, etc.). If even those steps are insufficient, then the user can replace any convection correlations by inputting a routine with the same name and same argument list in SUBROUTINES (Section 7.11.3).

The reason for this recommendation is the same as the reason for a caution in the use of HTU ties. *Without the use of the UB and UEDT parameters described below, the program must assume the UA value of an HTU tie stays constant over each solution interval, and this may be a source of oscillations when used in combination with time-independent elements* if the user updates the conductance purely based on the state of the network before each solution step. With other ties, the UA product is calculated simultaneously with the lump state and nodal temperature by iterative calls to the convection correlations. Such “implicitness” improves stability but is not possible with HTU ties because the user is given limited opportunities to update the UA product (namely in FLOGIC 0 and 2 blocks).

* REDUCER and EXPANDER connectors can also be named, but do not contribute any heat transfer area to the tie.

For example, assume a user updates a UA value in FLOGIC 0 (or via an expression that is updated internally after FLOGIC 0) for a tie on a junction based on the state of the junction in a single-constituent (pure substance) case. If the quality of the junction is 0.0, a low value of UA is used for heat transfer in a liquid. If the quality rises above 0.0, a much larger value of UA is used to represent two-phase heat transfer for pure substances. Because junctions can change instantaneously (as can tanks in STEADY), the junction can easily oscillate between liquid and two-phase if the wall node is colder than the two-phase fluid entering the junction. A large two-phase coefficient cools the junction to single-phase; a small single-phase coefficient allows the junction to warm up into the two-phase region again. The problem is the same when the wall node is warmer than the two-phase flow entering the junction.

To counteract such oscillations, the user might attempt to anticipate such effects as much as possible by solving for an appropriate value of UA simultaneously with the junction state. This would become cumbersome for strings of junctions because of the dependency on upstream conditions. An easier though less physically meaningful method would be to add hysteresis or damping into the UA product update, which is not always as easy as it sounds, nor is it always successful.

One alternate to the “constant UA” problem is to describe how the UA varies, at least as a function of temperature changes. This is done using the AHT, UB, and UEDT parameters as follows:

$$UA_{tie} = AHT_{tie} \cdot UB_{tie} \cdot (T_{node} - TEF_{tie})^{UEDT_{tie}}$$

where:

AHT . . . . . heat transfer area (m^2 or ft^2)
UB. . . . . the base (temperature-independent) portion of the conductance U
UEDT . . . . . exponent on temperature difference

By default, these parameters are all zero and therefore ignored in favor of UA. Otherwise, when they are active the tie equation becomes:

$$QTIE_{tie} = UAM_{tie} \cdot AHT_{tie} \cdot UB_{tie} \cdot (T_{node} - TEF_{tie})^{1+UEDT_{tie}}$$

For pool boiling, UEDT is 2.0: the heat transfer coefficient is proportional to ΔT^2 since the heat flux is proportional to ΔT^3. (This example is for illustration purposes only: use HTP ties, Section 3.6.3.3, for pool boiling.) For natural convection (Section 7.11.4.8), UEDT is 1/4 for laminar flow and about 1/3 for turbulent flow.

In summary, *when UB, AHT, and UEDT are all nonzero, the program is able to solve more implicitly and therefore achieve better convergence and larger time steps.* UEDT can still be zero, in which case UB*AHT is mathematically identical to UA, but may be a more convenient or explicit choice.

Finally, note that HTU ties are somewhat unique (along with HTP ties) in their ability to accept inputs specific to each phase. Use twinned ties with ITAC=1 to enable this usage.

The format for inputting the HTU tie in FLOW DATA is provided below.

HTU Subblock Format:

```
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]
```

where:

- `tid` . . . . . tie id
- `lid` . . . . . lump id with which heat will be transferred
- `nid` . . . . . node id with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam
- `UA` . . . . . conductance of the tie. The conductance must be positive or zero.
- `DUPN` . . . . . node duplication factor (number of this tie seen by nid)
- `DUPL` . . . . . lump duplication factor (number of this tie seen by lid)
- `UAM` . . . . . conductance multiplier (scaling factor)
- `AHT` . . . . . heat transfer area (m^2 or ft^2). If AHT is zero, UA will be used instead.
- `UB` . . . . . the base (temperature-independent) portion of the conductance U
- `UEDT` . . . . . exponent on temperature difference
- `MODW` . . . . . Wall temperature method (see Section 3.6.5):
  - 0 . . . . . TL=TEF and DTEF=0: no high speed corrections, but superheated/subcooled wall state corrections still apply
  - 1 . . . . . User-specified velocity UVEL
    -1 disables superheated/subcooled wall state corrections
  - 2 . . . . . User-specified path identifier IPUV
    -2 disables superheated/subcooled wall state corrections
  - 3 . . . . . User-specified offset temperature DTEF
- `UVEL` . . . . . Velocity for calculating TEF. (see Section 3.6.5)
- `IPUV` . . . . . Path ID for velocity basis of UVEL (see Section 3.6.5)
- `DTEF` . . . . . TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)
- `TTWIN` . . . . . id of twinned (vapor or secondary) tie
- `ITAC` . . . . . twinned tie apportioning rule
  - 1 . . . . . user controls both UAs and FQs (or UB, etc.), and twinned ties will remain twinned even with homogeneous tanks
  - 2 . . . . . AHT of primary tie is apportioned (using FQ factors) according to tank volume ratio. Only the primary tie will remain if the lump is a plenum, junction, single-phase tank, or homogeneous (mixed mode) twinned tank.
- `DEPTH` . . . . . Distance below liquid surface for ITAC=1 ties (used in FloCAD Compartments)
- `XOL` . . . . . lower (liquid) limit for volume fraction apportionment (ITAC=2)
- `XOH` . . . . . upper (vapor/gas) limit for volume fraction apportionment (ITAC=2)

defaults:

DUPN 1.0
DUPL 1.0
UA. 0.0
UAM 1.0
AHT . . . . . 0.0 (UA is therefore used instead)
UB. 0.0
UEDT 0.0
MODW . . . . . 0 (No velocity corrections, but may still have local wall state in two-phase)
IPUV 0
UVEL 0.0
DTEF 0.0
TTWIN. . . . . no twin (single tie created)
ITAC . . . . . 2 (AHT of primary tie apportioned according to tank volume ratio)
DEPTH. 0.0
XOL 0.0
XOH 1.0

Guidance: This option allows the user to define the overall heat transfer between the lump and the node accounting for almost any phenomena. Note that no path is required. The UA value is held constant over each solution interval.

Guidance: References to processor variables within expressions may refer to "#this" as the tie identifier, "#node" as the identifier of the node, and "#lump" as the identifier of the lump. Note that "#path" is illegal in this block since there is no path associated with this type of tie.

Restrictions: *lid must be separately defined in **this** fluid submodel. nid must be defined in a thermal submodel.* The QDOT for lid will be automatically augmented during execution, and QTIE will be summed into the Q for nid. If UA=0.0, then the tie heat rate QTIE will be zero.

Guidance: If lid is an active twinned tank,* then the tie may be *optionally* twinned as well.

If tid is *not* twinned:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If tid is twinned, it will connect to both tanks if lid is the primary tank.

* See also Section 3.25.

When the tanks are separated the user can either control both UAs individually (ITAC=1), or use the default (ITAC=2) “void fraction” apportioning method subject to the upper and lower limits XOH and XOL, respectively. Actually, the program uses the volume ratio of the tanks irrespective of their contents, neglecting for apportionment purposes bubbles in the liquid tank and droplets in the vapor tank. The tie parameter FQ contains the apportionment factor for each tie if ITAC=2. If ITAC=1, the user should provide and adjust FQ as an input.

Note that ties defined in a FloCAD Compartment are type ITAC=1 by default, in which case the Compartment controls the FQ of the tie automatically.

For twinned ITAC=1 ties, the UA, AHT, UB, and UEDT keywords set the value of the primary (liquid) tie and, by default, that of the secondary (vapor/gas) tie as well. To specify these values independently (including expressions) for the secondary tie, use UA_V, AHT_V, UB_V, and UEDT_V. Note that these input keywords are not SINDA/FLUINT variables, meaning they do not appear in postprocessing (use UA, AHT, UB, and UEDT instead) and cannot be referenced in Fortran logic blocks.*

Twinned ITAC=1 ties do not collapse to homogeneous ties, even when the attached lump is not twinned and even when it is single-phase. Each side of the tie permanently addresses a particular phase, whether that phase is present or not. For example, when an attached lump contains 100% vapor or gas (XL=AL=1.0), the primary tie is still active, but normally its FQ will be zero (as set by the user or by the FloCAD Compartment), while the FQ of the secondary tie is normally be unity in that case. This treatment allows expressions (and NEL logic, per the footnote) to be developed and maintained for each phase separately.

When the tanks are homogeneous (combined), then the secondary twin of the tie is ignored and the primary tie connects to the primary tank.

Examples:

```
T HTU,10,10,RIBS.2014,UA=2.0
T HTU,22,222,SLAW.2
      UA = 0.0
T HTU,1011,10,11,TATO.330, UB = htc, AHT = hite*wid/resol
```

* They can, however, be adjusted in Network Element Logic (NEL) in FloCAD. For example, use “UB#this” in a tie’s NEL to refer to the UB of the primary tie, and “UB#twin” to refer to the UB of the secondary tie.

3.6.3.2 HTN and HTNC (Forced Convection) Ties

When these ties are used, the UA conductance for the tie is not specified directly by the user. It is instead calculated by the program according to the user's instructions. These ties are based on forced convection (including boiling and condensation) in internal ducts assuming constant wall and fluid temperatures over each solution interval. Appendix B describes the set of heat transfer correlations, which are all access via the central routine STDHTC. Section 3.26.4.2 describes the special circumstance of heat transfer within rotating passages with shear (RVR<1).

To enable FLUINT to calculate a UA based on forced convection, users must identify more than a container node and a fluid lump—they must also name at least one path.* Energy may only be added or extracted at fluid lumps. However, lumps have no specific geometry (except perhaps a volume), and cannot be characterized by a flow regime. Tubes and STUBE connectors, however, have specific geometry (areas, diameters, lengths) and flow rates, but have no associated mass or thermodynamic properties. Convection heat transfer calculations depend on specific properties,† geometries, flow rates, and wall temperatures. Thus, these predictions require the attributes of lumps, paths, and thermal nodes to be combined in one calculation. In order to describe the methods behind convection ties, some preliminary discussions of duct modeling are necessary.

Diabatic ducts, a necessary component in any heat exchanger model, may be modeled in FLUINT as a series of discretized sections. There are two basic options for describing each section.

First, the user may "lump" all the associated mass and volume of a heat exchanger section into a tank (or other lump), and connect this tank to the up- or downstream lump with a single path that has the appropriate geometry of the section. This is the basis of the HTN tie. With only one path used, energy is assumed to enter or leave the section at one end or the other (upstream or downstream—see Figure 3-14). Flow reversals do not change the end at which heat is exchanged (i.e., transfer at the downstream end of the section becomes transfer at the upstream end). The governing equation for this type of heat transfer is then:

$$UA = F * AP * HTC$$

where:

F . . . . . Fraction of active wetted area (or fouling factor)
AP . . . . . Heat transfer area (wetted area of path: 4.0*TLEN*AF/DH)
HTC . . . . . Heat transfer coefficient based on thermodynamic state of lump, temperature of wall, and flow rate and geometry of path

* Throughout this section, reference to *one path* or a *single path* can also be interpreted to mean reference to *the primary twin of a pair of twinned tubes or STUBE connectors.* Only the primary twin need be named, the involvement of the secondary twin, which shares the same passage, is implied. In other words, despite the presence of two paths, only one passage of the appropriate diameter, length, and heat transfer area exists. Thus, an HTN tie can refer to a single path or a single pair of twins. An HTNC tie can refer to two single paths, one single path and one pair of twins, or two pairs of twins.

† To model condensation in a mixture containing noncondensible gases, the diffusion factor must be supplied for all gases, as described in Section 3.21.4. A processor abort will result if necessary information is missing.

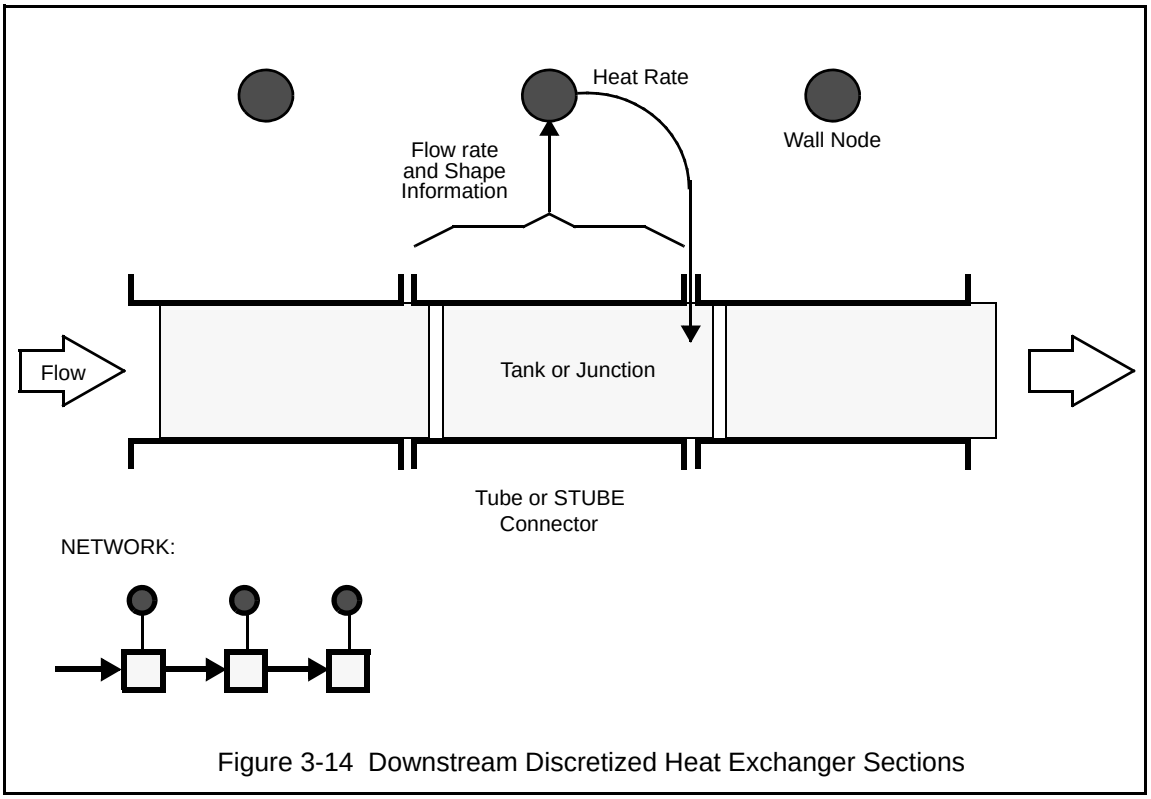


Figure 3-14 Downstream Discretized Heat Exchanger Sections

(The parameters AHT and UB will contain the values of F*AP and HTC, respectively. UEDT will be zero. These parameters are inputs for HTU ties, but are available as outputs for HTN and HTNC ties.)

Alternatively, the user may “center” the lump between two paths to represent the section. This is the basis of the HTNC (“centered”) tie. In other words, the energy entering or leaving the lump is the sum of the heat transfer calculated with two paths representing the container wall. One path may be upstream of the lump, and one may be downstream. For a string of such sections, the paths between the lumps may be shared (divided in two) for heat transfer purposes, as represented in Figure 3-15.

Actually, as noted in the following equation, the lump does not have to reside in the exact center of the path sections, and does not have to share a path at all. The HTN tie is simply a special case of the HTNC tie. However, if the lump is centered (F1=F2=0.5 for equal sized up and downstream paths), flow reversals will not alter the basic assumption of center-added heat. The equation for HTNC ties is:

$$UA = F1 * AP1 * HTC1 + F2 * AP2 * HTC2$$

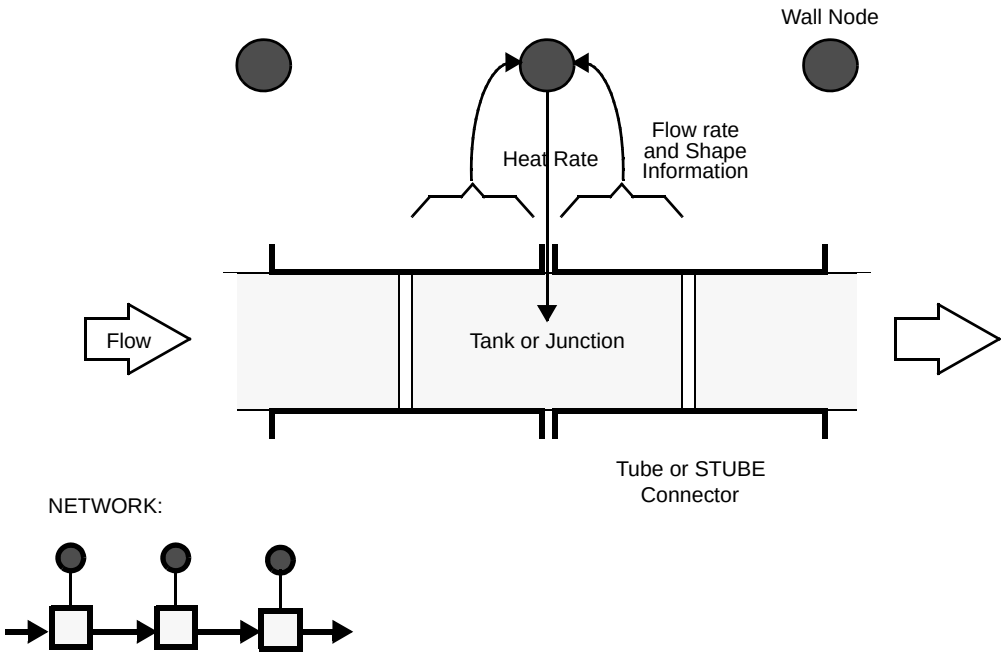


Figure 3-15 Center Discretized Heat Exchanger Sections

where:

F1 . . . . . Fraction of path 1 whose volume is contained in lump (fraction of active wetted area or fouling factor for path 1)

AP1 . . . . . Heat transfer area of path 1 (wetted area of path: $4.0*TLEN_1*AF_1/DH_1$)

HTC1 . . . . . Heat transfer coefficient based on thermodynamic state of lump and flow rate and geometry of path 1

F2 . . . . . Fraction of path 2 whose volume is contained in lump (fraction of active wetted area or fouling factor for path 2)

AP2 . . . . . Heat transfer area of path 2 (wetted area of path: $4.0*TLEN_2*AF_2/DH_2$)

HTC2 . . . . . Heat transfer coefficient based on thermodynamic state of lump and flow rate and geometry of path 2

(The parameters AHT and UB will contain the values F1*AP1+F2*AP2 and HTC_{avg}, respectively. UEDT will be zero. These parameters are inputs for HTU ties, but are available as outputs for HTN and HTNC ties.)

The area fractions for both HTN ties and HTNC ties are really just multipliers, and may be used to augment heat transfer (due to thinner than fully-developed boundary layers or impingement) or to degrade it (due to fouling). However, UAM and other factors (e.g., XNLM, XNTM) are better suited for such uses. The primary use of area fractions is for ducts whose heat transfer wetted area is not equal to the frictional wetted area: some parts are adiabatic. For example, a square duct with one side adiabatic would have an area fraction of 0.75. A duct with heat added over one half of its

length (or in which the other half of the path representing the duct were accounted for by another tie) would have an area fraction of 0.5. Tie duplication factors DUPN and DUPL are perhaps better suited for fouling or other multiplying factors since these can be dynamically altered within user logic blocks, whereas area multipliers can only be specified once within the FLOW DATA block.

As guidance, up- or downstream-lumped pipe and heat exchanger sections are slightly more efficient than center-lumped sections, especially for single-phase flow. The reason is that heat transfer will not change significantly from one method to the other, although pressure drops may differ slightly with two-phase flow (causing an indirect effect to the heat transfer calculations). Using a center-lumped method may require a smaller time step (because of the half-sized paths on the ends) and will require an additional lump/path pair to produce the same resolution of up or downstream-lumped methods. However, a center-lumped method is less sensitive to flow reversals and *should be the method-of-choice for two-phase flow* because of large property gradients along the line. **When in doubt, use the center-lump method.**

A heat exchanger section may also be modeled using a junction in place of a tank if the mass and energy storage capabilities of the section are neglected. In fact, this approach is strongly recommended for two-phase problems if transient liquid inventory changes within the section are not part of the problem. Using junctions instead of two-phase tanks will significantly increase solution speed. Both plena and boundary nodes may be named in ties used to represent boundary conditions for either thermal or fluid submodels.

Finally, it should be noted that there are several powerful preprocessor commands available in FLUINT that allow fast generation of complete heat exchanger models (as well as segmented line models without such heat transfer calculations). These commands (LINE and HX) are discussed in the a later section (3.9.2).

Inherent Downstream-weighted Heat Transfer—The above discussion on upstream, downstream, and centered discretizing relates to where the lump is located with respect to paths, which dictate flow rates and heat exchange area. However, this assignment is unrelated to where the lump is located with respect to the wall node(s). In fact, inspection of the equation

$$QDOT = QL + \Sigma QTIE = QL + \Sigma(UAM*UA*(TEF - TL)) \qquad \text{(fties etc. neglected)}$$

reveals an inherent assumption of downstream-weighted heat transfer: under steady conditions, the temperature (or quality) of the lump is a function of the heat transfer into it, which is in turn a function of its temperature (or quality). In other words, the heat transfer rate is calculated on the temperature at which fluid exits the perfectly mixed lump. *When plotting results versus lump axial location, the lump should be considered to exist at the downstream end of each section when using lumped parameter ties.*

This lumped parameter formulation is stable, and is entirely consistent with the fundamental assumptions of a lumped-parameter or finite difference network. However, when lumps are coarsely discretized (i.e., too few lumps are used to adequately represent the flow-wise temperature gradient), the heat transfer rate will be underestimated, especially if the node is perceived as representing an average wall temperature for the section.

Strictly, the correct solution is to not abuse the network assumptions, and use an adequate number of elements to capture the desired number of states. However, the user may alternatively choose to interpret the lump states as representing an inlet and outlet condition, with the node representing the average wall temperature from inlet to outlet. For single-phase flow, where the heat transfer coefficients and flow rates are uniform axially, the types of ties described in the next section (3.6.4) may be more appropriate.

FLOW DATA Formats—The input formats for both the HTN and HTNC ties are specified below.

HTN Subblock Format:

```
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]
```

where:

- `tid` ..... tie id
- `lid` ..... lump id with which heat will be transferred
- `nid` ..... node id with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam
- `pid` ..... path id of the path to use for flow rate and geometry (either tube or STUBE connector). Alternatively, the path id of the primary path of a pair of twins (the involvement of the secondary path is implied). For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area: AHT and therefore UA will be zero.
- `f` ..... fraction of pid wetted area attributed to tie
- `DUPN` ..... node duplication factor (number of this tie seen by nid)
- `DUPL` ..... lump duplication factor (number of this tie seen by lid)
- `TTWIN` ..... id of twinned (vapor or secondary) tie
- `UAM` ..... conductance multiplier (scaling factor)
- `MODW` ..... Wall temperature method (see Section 3.6.5):
  - 0 ..... TL=TEF and DTEF=0: no high speed corrections, but superheated/subcooled wall state corrections still apply
  - 1 ..... User-specified velocity UVEL
    -1 disables superheated/subcooled wall state corrections
  - 2 ..... User-specified path identifier IPUV
    -2 disables superheated/subcooled wall state corrections
  - 3 ..... User-specified offset temperature DTEF

UVEL.....Velocity for calculating TEF. (see Section 3.6.5)
IPUV.....Path ID for velocity basis of UVEL (see Section 3.6.5)
DTEF.....TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)
MODR.....Reference state method (see Section 3.6.5):
0.....Use TL (static temperature).
1.....Use a simple average of TL and T_{wall} as the reference state
2.....Use T_{wall}
3.....Use Eckert's reference enthalpy.
4.....Use van Driest's reference enthalpy.
5.....Use the Schacht-Quentmeyer integrated property method.
XNUL.....Laminar Nusselt number (see Section 3.6.6)
XNLM.....Laminar Nusselt number multiplier (see Section 3.6.6)
XNLA.....Laminar Nusselt number bias (see Section 3.6.6)
XNTM.....Turbulent Nusselt number multiplier (see Section 3.6.6)
XNTA.....Turbulent Nusselt number bias (see Section 3.6.6)
CDB.....Dittus-Boelter scaling parameter (see Section 3.6.6)
UER.....Reynolds number exponent (see Section 3.6.6)
UEC.....Prandtl number exponent for cooling (see Section 3.6.6)
UEH.....Prandtl number exponent for heating (see Section 3.6.6)
UEV.....Correction factor exponent for viscosity, applies to both laminar and turbulent flow (see Section 3.6.6)
UED.....Turbulent correction factor exponent for density (see Section 3.6.6)
UEK.....Turbulent correction factor exponent for thermal conductivity (see Section 3.6.6)
UECP.....Turbulent correction factor exponent for specific heat (see Section 3.6.6)
UEP.....Turbulent correction factor exponent for pressure (see Section 3.6.6)
UET.....Turbulent correction factor exponent for temperature (see Section 3.6.6)
RLAM.....Lower limit in Reynolds number for transitioning flow: the flow must be laminar if the Reynolds number is below this value.
RTURB .....Upper limit in Reynolds number for transitioning flow: the flow must be turbulent if the Reynolds number is above this value. RTURB should be greater than or equal to RLAM.
CHFF.....Critical heat flux factor (see Section 3.6.7)
HFFL.....Scaling parameter on Leidenfrost flux/temperature (lower limit to film boiling) estimation (see Section 3.6.7)
XNB.....Upper flow quality limit on nucleate boiling. Past this quality, the film coefficient will transition to single-phase vapor using interpolation.

defaults:

f.....1.0
DUPN.....1.0
DUPL.....1.0

TTWIN..... no twin (single tie created)
UAM 1.0
MODW..... 2 (path IPUV determined by program)
IPUV..... 0
UVEL..... 0.0
DTEF..... 0.0
MODR..... 3 (Eckert's reference enthalpy)
XNUL..... 3.66
XNLM..... 1.0
XNLA..... 0.0
XNTM..... 1.0
XNTA..... 0.0
CDB 0.023
UER 0.8
UEC 0.3
UEH 0.4
UEV 0.0
UED 0.0
UEK 0.0
UECP..... 0.0
UEP 0.0
UET 0.0
RLAM..... 1958.785
RTURB..... 6421.960
CHFF..... π/24
HFFL..... 0.09
XNB 0.7

Guidance: This option specifies the automatic calculation of convective heat exchange between a fluid lump and a thermal node that represents the container wall. A standard set of forced convection correlations are used, including correlations for boiling and condensing (see Appendix B). The path named pid will be used to provide geometry (diameter, perimeter, flow area, length, and heat transfer area) and flow rate information to the correlations. If a twinned pair of paths is used, only the ID of the primary path need be input: the involvement of the secondary path is implied. If flows with phase change are being analyzed, the use of the default IPDC=6 is recommended for the path. Heat transfer to a constant wall temperature may be simulated if nid is a boundary node. Alternatively, a fluid-dominated thermal boundary condition may be simulated if lid is a plenum. This option may be used to specify one heat exchanger segment where the mass and volume of the segment are contained in lid, and the

length and cross-section of the same segment are described by pid. lid may be up or downstream of pid, depending on modeling preferences. The area fraction is applied to the final UA value, and hence may be used to model nonuniform temperatures, fouling, etc.

Guidance: References to processor variables within expressions may refer to "#this" as the tie identifier, "#node" as the identifier of the node, and "#path" as the identifier of the path, and "#lump" as the identifier of the lump.

Restrictions: *lid and pid must be separately defined in* ***this*** *fluid submodel. nid must be defined in a thermal submodel.* The QDOT for lid will be calculated automatically during execution, and will be summed into the Q for nid. If f=0.0, the tie will be adiabatic.

Restrictions: *pid cannot reference a secondary twin, nor to a path other than a tube or STUBE.* For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area: AHT and therefore UA will be zero.

Guidance: If lid is an active twinned tank,* then the tie may be twinned as well. If either lid or the paths are *not* twinned, the twinning of the tie is optional *and the secondary tie will be ignored*. But if lid and pid *is* twinned, the tie must also be twinned.

If the path is not twinned, a singlet tie may be used and:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If lid and pid are both twinned, then the tie must be twinned and will connect to both tanks.

When the tanks are separated the user cannot control apportionment, since this is handled automatically as a function of the underlying flow and heat transfer regime. The code uses ITAC=3 to denote this automatic control. The tie parameter FQ contains the resulting apportionment factor for each tie.

When the tanks are homogeneous (combined), then the secondary twin of the tie is ignored and the primary tie connects to the primary tank.

Examples:

```
T HTN,10,10,RIBS.2014,33,  DUPL=2.0, DUPN=2.0
T HTN,22,222,SLAW.2,2203,0.5
      DUPL  = (T#node > TL#lump)? 1.0 : 0.0          $ diode
      DUPN  = DUPL#this
```

* See also Section 3.25.

HTNC Subblock Format:

```
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]
```

where:

`tid` ..... tie id

`lid` ..... lump id with which heat will be transferred

`nid` ..... node id with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam

`pid` ..... path id of first path to use for flow rate and geometry (either tube or STUBE connector). Alternatively, the path id of the primary path of a first pair of twins (the involvement of the secondary path is implied). For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area.

`f1`..... fraction of pid wetted area attributed to tie

`p2id` ..... path id of second path to use for flow rate and geometry (either tube or STUBE connector). Alternatively, the path id of the primary path of a second pair of twins (the involvement of the secondary path is implied). For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area.

`f2`..... fraction of p2id wetted area attributed to tie

`DUPN` ..... node duplication factor (number of this tie seen by nid)

`DUPL` ..... lump duplication factor (number of this tie seen by lid)

`TTWIN`..... id of twinned (vapor or secondary) tie

`UAM` ..... conductance multiplier (scaling factor)

`MODW` ..... Wall temperature method (see Section 3.6.5):

- 0 ..... TL=TEF and DTEF=0: no high speed corrections, but superheated/subcooled wall state corrections still apply
- 1 ..... User-specified velocity UVEL
  -1 disables superheated/subcooled wall state corrections
- 2 ..... User-specified path identifier IPUV
  -2 disables superheated/subcooled wall state corrections
- 3 ..... User-specified offset temperature DTEF

`UVEL` ..... Velocity for calculating TEF. (see Section 3.6.5)

`IPUV` ..... Path ID for velocity basis of UVEL (see Section 3.6.5)

`DTEF` ..... TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)

MODR. . . . . Reference state method (see Section 3.6.5):
- 0. . . . . Use TL (static temperature).
- 1. . . . . Use a simple average of TL and T_{wall} as the reference state
- 2. . . . . Use T_{wall}
- 3. . . . . Use Eckert's reference enthalpy.
- 4. . . . . Use van Driest's reference enthalpy.
- 5. . . . . Use the Schacht-Quentmeyer integrated property method.

XNUL. . . . . Laminar Nusselt number (see Section 3.6.6)

XNLM. . . . . Laminar Nusselt number multiplier (see Section 3.6.6)

XNLA. . . . . Laminar Nusselt number bias (see Section 3.6.6)

XNTM. . . . . Turbulent Nusselt number multiplier (see Section 3.6.6)

XNTA. . . . . Turbulent Nusselt number bias (see Section 3.6.6)

CDB . . . . . Dittus-Boelter scaling parameter (see Section 3.6.6)

UER . . . . . Reynolds number exponent (see Section 3.6.6)

UEC . . . . . Prandtl number exponent for cooling (see Section 3.6.6)

UEH . . . . . Prandtl number exponent for heating (see Section 3.6.6)

UEV . . . . . Correction factor exponent for viscosity, applies to both laminar and turbulent flow (see Section 3.6.6)

UED . . . . . Turbulent correction factor exponent for density (see Section 3.6.6)

UEK . . . . . Turbulent correction factor exponent for thermal conductivity (see Section 3.6.6)

UECP. . . . . Turbulent correction factor exponent for specific heat (see Section 3.6.6)

UEP . . . . . Turbulent correction factor exponent for pressure (see Section 3.6.6)

UET . . . . . Turbulent correction factor exponent for temperature (see Section 3.6.6)

RLAM. . . . . Lower limit in Reynolds number for transitioning flow: the flow must be laminar if the Reynolds number is below this value.

RTURB . . . . . Upper limit in Reynolds number for transitioning flow: the flow must be turbulent if the Reynolds number is above this value. RTURB should be greater than or equal to RLAM.

CHFF. . . . . Critical heat flux factor (see Section 3.6.7)

HFFL. . . . . Scaling parameter on Leidenfrost flux/temperature (lower limit to film boiling) estimation (see Section 3.6.7)

XNB . . . . . Upper flow quality limit on nucleate boiling. Past this quality, the film coefficient will transition to single-phase vapor using interpolation.

defaults:

f2..... 0.5
DUPN..... 1.0
DUPL..... 1.0
TTWIN..... no twin (single tie created)
UAM 1.0
MODW..... 2 (path IPUV determined by program)
IPUV..... 0
UVEL..... 0.0
DTEF..... 0.0
MODR..... 3 (Eckert's reference enthalpy)
XNUL..... 3.66
XNLM..... 1.0
XNLA..... 0.0
XNTM..... 1.0
XNTA..... 0.0
CDB 0.023
UER 0.8
UEC 0.3
UEH 0.4
UEV 0.0
UED 0.0
UEK 0.0
UECP..... 0.0
UEP 0.0
UET 0.0
RLAM..... 1958.785
RTURB..... 6421.960
CHFF..... π/24
HFFL..... 0.09
XNB 0.7

Guidance: This option specifies the automatic calculation of heat exchange between a fluid lump and a thermal node that represents the container wall. A standard set of forced convection correlations are used, including correlations for boiling and condensing. The paths named p1id and p2id will be used to provide geometry (diameter, perimeter, flow area, length, and heat transfer area) and flow rate information to the correlations. If a twinned pair of paths is used, only the ID of the primary path need be input: the involvement of the secondary path is implied. Both, either, or neither of p1id and p2id may represent twins; any combination is allowed. If two-phase multiple-constituent flows are being analyzed, the use of the default IPDC=6 is recommended for the paths. Heat transfer to a constant wall temperature may be simulated if nid is a boundary node. Alternatively, a fluid-dominated thermal boundary condition may be sim-

ulated if lid is a plenum. This option may be used to specify one heat exchanger segment where the mass and volume of the segment are contained in lid. The segment is assumed to span f1 of p1id, and f2 of p2id. Thus, lid is assumed to be "centered" between p1id and p2id if f1=f2=0.5, and this option is the same as HTN if f1=1.0 and f2=0.0. Note that f1 and f2 need not sum to one. The area fractions are applied to the final UA value, and hence may be used to model nonuniform temperatures, fouling, etc.

Guidance: References to processor variables within expressions may refer to "#this" as the tie identifier, "#node" as the identifier of the node, and "#path" as the identifier of the first path, "#path2" as the identifier of the second path, and "#lump" as the identifier of the lump.

Restrictions: *lid, p1id, and p2id must be separately defined in* ***this*** *fluid submodel. nid must be defined in a thermal submodel.* The QDOT for lid will be augmented automatically during execution, and QTIE will be summed into the Q for nid. If f1=f2=0.0, then the tie heat rate QTIE will be zero.

Restrictions: *p1id and p2id cannot reference a secondary twin, or a path other than a tube or STUBE.* For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area: AHT and therefore UA will be zero.

Guidance: If lid is an active twinned tank,* then the tie may be twinned as well. The paths pid and pid2 must then either be both twinned or both untwinned (singlets). If either lid or the paths are *not* twinned, the twinning of the tie is optional *and the secondary tie will be ignored*. But if lid and the paths *are* twinned, the tie must also be twinned.

If the paths are not twinned, a singlet tie may be used and:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If lid and the paths are twinned, then the tie must be twinned and will connect to both tanks.

When the tanks are separated the user cannot control apportionment, since this is handled automatically as a function of the underlying flow and heat transfer regime. The code uses ITAC=3 to denote this automatic control. The tie parameter FQ contains the resulting apportionment factor for each tie.

When the tanks are homogeneous (combined), then the secondary twin of the tie is ignored and the primary tie connects to the primary tank.

* See also Section 3.25.

Examples:

```
T HTNC,10,10,RIBS.2014, 33,0.5,34,1.0
T HTNC,22,222,SLAW.2
            2203,1.0
            2204,1.0
            DUPN  = 0.0 $ Node is unaware of tie
T HTNC,144,101,BEANS.24, 633,0.5, 366
```

3.6.3.3 HTP ("Pool Boiling" and Quasi-stagnant Convection) Ties

If liquid is stagnant within a pipeline and it begins to boil, it won't be stagnant for long as low-density vapor will quickly displace high-density liquid: forced convection ties (Section 3.6.3.2: HTN, HTNC) using Chen's nucleate boiling correlations are still appropriate in such a situation.

On the other hand, if a tank or vessel is being modeled rather than a duct, such that flow is nearly stagnant (excluding perhaps venting, draining, and filling), then HTP "pool boiling" ties offer a better choice. The terms *tank* and *vessel* can be used loosely: an HTP tie may also be appropriate for modeling more complex volumes (e.g., the chilldown of a pump volute to cryogenic temperatures before the pump can operate).

The primary purpose of an HTP tie is to automate the calculations surrounding pool boiling, including estimation of post critical heat flux (CHF) operation, and suppression or reduction of boiling deep beneath the surface of the "pool" (specifically, the subcooling corresponding to hydrostatic pressure head). These calculations could also be performed using an HTU tie (Section 3.6.3.1) with calls to the routine POOLBOIL (Section 7.11.4.1), but HTP ties relieve the user of excessive programming responsibility.

For completeness, an HTP tie also covers condensation and single-phase (zero or reduced phase change) regimes. However, since the program lacks detailed descriptions of the geometry, those heat transfer regimes are highly simplistic. For condensation, a user-input liquid film thickness is often the basis for calculation of the film coefficient. For single-phase regimes, natural convection in a "sphere" or mixed natural/forced convection in a filling or emptying "cylinder" is employed. The user-callable routines NCSPHERE (Section 7.11.4.8) and NCCYLIN (Section 7.11.4.9) are applied internally for those calculations. Of course, it is possible to use an HTP tie exclusively to automate such single-phase simulations, even if boiling never occurs (e.g., if the working fluid is a gas or gas-only mixture). In other words, an HTP tie may be used simply to automate the invocation of the NCCYLIN routine on the inside wall of a vessel.

To use an HTP tie, the heat transfer area (AHT) must be specified. The program will then calculate UA, UB, and UEDT. In the pool boiling regime, the local depth of the node (beneath the liquid/vapor interface represented by the lump's thermodynamic state) may be accounted for using the HTP tie's ACCM (local acceleration) and DEPTH parameters. This hydrostatic pressure (ρgh term, where ρ is the density of liquid, g is ACCM, and h is DEPTH) might be enough to suppress boiling altogether, but will often just cause an increase in the local saturation temperature.*

* Refer to the automatic calculation of DTEF in Section 3.6.7.1.

If the wall temperature is hot enough such that the program anticipates that the critical heat flux has been exceeded, then transition and film boiling will be estimated (under control of the parameters CHFF, HFFL, and XNB). If instead the wall is colder than saturation and condensation occurs, a user-supplied liquid thickness (THKL) is used to estimate heat transfer (if the coefficient itself is not directly supplied as the parameter HCON). If no phase change is occurring (perhaps because all liquid has been depleted), then natural convection is assumed (either in a sphere or in a filling or emptying vertical cylinder).

Note that HTP ties are somewhat unique (along with HTU ties) in their ability to accept inputs specific to each phase. Use twinned ties with ITAC=1 to enable this usage.

An output tabulation routine specific to HTP ties, TIEHTPTAB (Section 7.11.8), is available to check inputs and current status.

Note that wall reference states and high speed heat transfer options (MODR, MODW, DTEF, IPUV, UVEL, and DTEF) are available. However, these options will rarely apply to HTP ties, which are normally considered to represent stagnant or nearly-stagnant conditions.

Boiling Mode—The "boiling mode" of an HTP tie is largely based on internal calls to the POOLBOIL routine, which is also available for user calls and is therefore documented separately in Section 7.11.4.1. However, an HTP tie also attempts to continue simulations past the CHF limit into the transition and film boiling regimes. It also takes into account local liquid "depth" in a gravity or rotational field, such that the local saturation temperature at the node may be slightly greater than that of the lump.

If a volatile liquid is present, and if the wall is hotter than saturation but not so hot that the CHF (critical heat flux) limit is exceeded, then the pool boiling is simulated using Rohsenow's correlation.* In this regime, the heat transfer coefficient (H_{pb}) is strongly dependent on the degree of wall superheat, and therefore UEDT=2.0:

$$H_{pb} = \mu_l \left(\frac{\Delta T}{h_{fg}}\right)^2 \left(\frac{g(\rho_l - \rho_v)}{\sigma}\right)^{\frac{1}{2}} \left[\frac{C_{p,l}}{Pr_l^S \cdot C_{sf}}\right]^3$$

where S and C_{sf} are empirical corrections that depend on the fluid and the surface (see SPR and CSF parameters below). The body force "g" in the above is independent of either ACCM or the acceleration level set in NCSET, as described later.

* W.M. Rohsenow, "A Method of Correlating Heat-Transfer Data for Surface Boiling Liquids," *Transactions of ASME,* vol. 74, pp. 969-975, 1952.

The critical heat flux and Leidenfrost heat flux (QCHF, QMIN respectively) are calculated using Zuber's estimates (Section 3.6.7):

$$QCHF = CHFF \cdot \rho_v h_{fg} \left[\frac{\sigma g(\rho_l - \rho_v)}{\rho_v^2} \right]^{\frac{1}{4}} \left(\frac{\rho_l + \rho_v}{\rho_l} \right)^{\frac{1}{2}}$$

CHFF defaults to $\pi/24$. TCHF, the temperature corresponding to QCHF, is calculated using the above equation for H_{pb} such that $QCHF = H_{pb}*(TCHF - T_{sat}) = UB*(TCHF - T_{sat})^3$.

For QMIN, the level to which the heat flux must be reduced before liquid can be re-established on the wall, the estimation used is:

$$QMIN = HFFL \cdot \rho_v h_{fg} \left[\frac{\sigma g(\rho_l - \rho_v)}{(\rho_l + \rho_v)^2} \right]^{\frac{1}{4}}$$

HFFL defaults to 0.09. The body force "g" in both the QMIN and QCHF calculation is set by the HTP tie's ACCM value.

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., well above the critical point), is therefore usually overridden by the upper limit of the departure from film boiling (T_{dfb}) as described in Section 3.6.7:

$$T_{dfb} = 0.97 \cdot T_{crit} \left(0.932 + 0.077 \left(\frac{T_{sat}}{T_{crit}} \right)^9 \right) + 0.03 \cdot T_{sat}$$

In other words, $TMIN = \min(T_{leid}, T_{dfb})$.

If the wall becomes hotter than TCHF, transition boiling will begin. Transition boiling is basically a nonlinear interpolation between nucleate boiling and film boiling, based on the scaling suggested by Ramilison and Lienhard (the same scaling as is used by HTN and HTNC ties).

However, the film boiling heat transfer coefficient is *not* similar to that used by HTN and HTNC ties since details of the geometry and flow field are lacking. Rather, the film boiling "correlation" for HTP ties is simply a calculation based on TMIN and QMIN. Given these two values, the film boiling heat transfer coefficient, H_{fb}, is chosen such that numerical continuity exists at that point: H_{fb} = QMIN/(TMIN-T_{sat}).

Beyond TMIN, H_{fb} scales with $(T_{wall} - T_{sat})^{-1/4}$ (i.e., UEDT=-0.25), and a correction for radiation is also made for extremely high wall temperatures.

These post-CHF calculations should be treated only as rough guesses, and are present for completeness and numerical continuity.

As with HTN and HTNC ties, if the lump's quality exceeds XNB, even if the flux is beneath QCHF, an interpolation with 100% vapor heat transfer begins. (With natural convection, the tie's UEDT is on the order of 0.25 to 0.35.) Similarly, adding nonvolatile liquids to the mixture can cause the heat transfer to be degraded, via interpolation based on the liquid fraction, toward a single-phase (nonboiling) regime. For all of these reasons, the calculated UEDT may be anywhere between -0.25 and 2.

Condensing Mode—If the wall is colder than saturation, condensation will occur. The condensing film coefficient (H_c) for an HTP tie may be given directly as H_c= HCON (if HCON is positive).

If HCON is negative (and HCON = -1.0 by default), then the condensation coefficient is based on a user-supplied value of THKL, the thickness of the liquid layer against the wall. The magnitude of HCON is then used as a scaling factor for this calculation: $H_c = |HCON|*K_{liq}/THKL$ where K_{liq} is the thermal conductivity of the liquid in the lump (calculated automatically by the program). The returned UEDT is 0.0 for condensation of a pure substance.

If condensation is an important mode, then the user should consider updating HCON and/or THKL with estimates from other methods or correlations. Otherwise, these default values and methods are highly simplified.

THKL defaults to 1mm (3.28084e-3 feet), and is intended for the user to adjust or update in logic or expressions. Set THKL=DEPTH#this, for example, if the node is located at the bottom of a tank or vessel and DEPTH is being adjusted or calculated separately.

If noncondensible gases are present in the mixture, the resulting condensation coefficient (H_c) may be reduced according to diffusion blockage effects using internal calls to HTUDIF.

Single-Phase Mode—If all volatile liquid is gone, or if all condensible vapor is gone (or if a twinned tie is above or below the liquid level, as explained in the next subsection), then neither pool boiling nor condensation is appropriate. The user then has the choice if defining the Nusselt number, letting the program calculate heat transfer for the HTP tie based on natural convection within a sphere, or letting the program calculate heat transfer based on a cylinder that is being filled or emptied from the top or bottom. In the last case, the user also defines a flow path that describes the flow in or out of the cylinder: a "port path."

In the single-phase mode, heat transfer coefficient (H_{sp}) may be specified directly by providing a positive value for the Nusselt number XNUL, and by providing a value for DCH, the characteristic dimension upon which the Nusselt number is based. In this mode (positive XNUL), H_{sp} = XNUL*K/DCH where K is the thermal conductivity of the fluid. If the scaling and shifting factors XNLM and XNLA are used, then H_{sp} = (XNLM*XNUL+XNLA)*K/DCH

On the other hand, if XNUL is negative, a more complex calculation is made based on natural convection (and perhaps mixed convection) and |XNUL| is interpreted as a multiplying factor on this calculation. The remainder of this section documents the calculation made when XNUL is negative, noting that XNUL = -1.0 by default.

If no inlet/outlet port path (*iport*) has been defined, then H_{sp} is calculated based on natural convection within a sphere using DCH as the sphere diameter in an internal call to NCSPHERE. As noted in the documentation of the routine NCSPHERE in Section 7, the vessel need not be a real sphere, and DCH may be calculated or estimated for more complex geometries.

If instead a port path (*iport*) *has* been defined, then that port path's flow rate (after multiplying by *fport*), flow area AF, and diameter (calculated as if circular for non-duct paths), are applied assuming mixed natural and forced convection within a vertical cylinder using an internal call to NCCYLIN. A positive sign on *iport* signals that the path is located on top of the vessel, and a negative sign signals that it is located instead on the bottom. In this mode, DCH is interpreted as the length of a real or effective cylinder. NCCYLIN assumes a positive flow rate is entering the vessel. If instead the path |*iport*| is defined as positive for flow *out* of the tank or vessel, use *fport*=-1.0 to rectify the flow rate.

If the scaling and shifting factors XNLM and XNLA are used with negative XNUL, then the resulting Nusselt number from the above NCSPHERE or NCCYLIN calculations (Nu_{sp}) is adjusted as $Nu_{sp,eff}$ = XNLM*|XNUL|*Nu_{sp}+XNLA.

If a port path has been defined, yet its flow rate has become zero, Nu_{sp} = 1.0 is produced by NCCYLIN. This is likely to be an underestimate of the heat transfer rate. If port path is used, and the single-phase mode persists for a long time with no flow, then a parallel HTP tie that lacks a port path (such that NCSPHERE is invoked instead of NCCYLIN) is advised.*

During these internal calls to NCSPHERE or NCCYLIN, the HTP tie's ACCM value is *not* used as the gravity term in case it represented a spin rate and was intended only to affect boiling calculations. The user must therefore independently adjust this natural convection body force acceleration value using NCSET calls (see parameter "GEEF") if necessary.

The returned UEDT for NCSPHERE is usually within the range of 0.25 to 0.3333, and for NCCYLIN, UEDT can be up to 0.352. Therefore, a UEDT value in the range of 0.25 to 0.352 is an indication that single-phase heat transfer is dominant.

* Set UAM=0.0, or set DUPN=DUPL=0.0, in order to turn off whichever tie is currently redundant.

In the special case of a nonvolatile liquid forming a two-phase mixture with a noncondensible gas, the above single-phase calculation is made, interpolating between liquid and vapor coefficients based on the void fraction of the lump. Such calculations are present only to provide numerical continuity, and should not be relied upon for predictions since the underlying natural convection correlations were never intended for two-phase mixtures.

If, during a run, the definition of *iport* or *fport* needs to be changed for an HTP tie, the CHG_HTPPORT utility routine (Section 7.11.1.1) can be called.

Using with Twinned Ties and Tanks—Unlike HTN or HTNC ties, if an HTP tie is twinned and attached to a twinned tank (Section 3.25), the UA of the each tie may be distinct. In other words, a single UA is not calculated and apportioned as it is with HTN and HTNC ties, but rather the UA of the primary tie and secondary tie are calculated independently. It is therefore possible for boiling to be occurring in the primary tie (to the primary, liquid tank), while condensation is occurring in the secondary tie (to the secondary, vapor tank).

By default (ITAC=3), the program will use the input values of ACCM and DEPTH to decide how to calculate and apportion the twinned tank heat transfer.

If ACCM=0.0 (zero gravity), for example, liquid is assumed to be against the wall and all heat will be exchanged with the liquid tank (FQ_{vap}=0.0; the UA for the vapor twin will be updated but disabled). If boiling is detected, the heat will be applied to the liquid tank (FQ_{liq}=1.0). However, if condensation or nonvolatile heating is detected, FQ_{liq} will be set to 2.0 to reflect the fact that heat transfer is interpreted as applying to the middle of the liquid layer, with the liquid located between the wall and the liquid/vapor interface (where H_c is defined, for example).

For nonzero ACCM (noting 1g is the default), if DEPTH=0.0 then void fraction apportioning (as if ITAC=2) is applied to AHT (while the UB and UEDT for each twin may still be independent). Otherwise, if DEPTH>0 then all heat is exchanged with the liquid (FQ_{liq}=1.0, FQ_{vap}=0.0) and if DEPTH<0, all heat is exchanged with the vapor or gas phase (FQ_{liq}=0.0, FQ_{vap}=1.0).

As noted above, the UA for each tie is updated as if it were active, even if that heat transfer pathway is disabled or ignored (FQ=0.0) according to the estimated location of the node relative to the liquid/vapor interface.

HTP ties are important components of FloCAD Compartments, which by default use twinned ties with ITAC=1. For Compartment ties, parameters such as UAM, DCH, and DEPTH are updated automatically, but the user may still select values such as SPR, XNB, or XNUL.

Even without use of FloCAD Compartments, ITAC=1 twinned HTP ties may be used to address each phase separately, with unique DCH, FQ, XNUL, etc. values. In this mode, HTP ties remained twinned even if the attached lump is a single-phase tank, junction, or plenum.

Use TI2TAB (Section 7.11.8) to print a summary of twinned ties.

FLOW DATA Formats—The input formats for HTP ties are specified below.

HTP Subblock Format:

```
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]
```

where:

`tid` ..... tie ID

`lid` ..... lump ID with which heat will be transferred. This may be the primary (liquid) tank in a pair of twinned tanks.

`nid` ..... node ID with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam

`iport` ..... optional path ID of the vessel inlet or outlet flow path to use for flow rate and flow area (any path with a nonzero AF is valid) for single-phase calculations based on an internal call to NCCYLIN. If iport is the primary path in a pair of twinned paths, the involvement of the secondary path is implied. The sign of iport is used as a signal to indicate the location of the port: positive for the top, and negative for the bottom of the vessel.
If iport is missing or zero, natural convection within a closed sphere (NCSPHERE methods) will be used for estimating single-phase heat transfer.
During a run, iport can be changed via a call to CHG_HTPPORT (Section 7.11.1.1).

`fport` ..... optional multiplier on the flow rate (*not* wetted surface area) of iport. Path iport is assumed to flow into the vessel in question, so fport=-1.0 can be used as a multiplier to rectify the flow rate of iport if it instead flows out of the vessel in the positive FR direction. In other words,
FRin = fport*FR(|iport|)
During a run, fport can be changed via a call to CHG_HTPPORT (Section 7.11.1.1).

`AHT` ..... heat transfer area (m^2 or ft^2)

`DUPN` ..... node duplication factor (number of this tie seen by nid)

`DUPL` ..... lump duplication factor (number of this tie seen by lid)

TTWIN . . . . . ID of optional twinned (vapor or secondary) tie

ITAC. . . . . twinned tie apportioning rule

1. . . . . .user controls both FQs, DCHs, XNULs, etc., and twinned ties will remain twinned even with homogeneous tanks

2. . . . . .AHT of primary tie apportioned (using FQ factors) according to tank volume ratio. Only the primary tie's XNUL, DCH, etc. value will be used for both phases. Only the primary tie will remain if the lump is a plenum, junction, single-phase tank, or homogeneous (mixed mode) twinned tank.

3. . . . . .AHT apportioned according to estimate of liquid location based on ACCM, DEPTH, and presence/absence of nucleate boiling. Only the primary tie's XNUL, DCH, etc. value will be used for both phases. Only the primary tie will remain if the lump is a plenum, junction, single-phase tank, or homogeneous (mixed mode) twinned tank.

XOL. . . . . .lower (liquid) limit for volume fraction apportionment (ITAC=2)

XOH. . . . . .upper (vapor/gas) limit for volume fraction apportionment (ITAC=2)

UAM. . . . . .conductance multiplier (scaling factor)

MODW. . . . . .Wall temperature method (see Section 3.6.5):

0. . . . . .TL=TEF and DTEF=0: no high speed corrections, but super-heated/subcooled wall state corrections still apply

1. . . . . .User-specified velocity UVEL

-1 disables superheated/subcooled wall state corrections

2. . . . . .User-specified path identifier IPUV

-2 disables superheated/subcooled wall state corrections

3. . . . . .User-specified offset temperature DTEF

UVEL. . . . . .Velocity for calculating TEF (see Section 3.6.5)

IPUV. . . . . .Path ID for velocity basis of UVEL (see Section 3.6.5)

DTEF. . . . . .TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)

MODR. . . . . .Reference state method (see Section 3.6.5):

0. . . . . .Use TL (static temperature)

1. . . . . .Use a simple average of TL and T_{wall} as the reference state

2. . . . . .Use T_{wall}

3 or 4 . . . . .Not allowed with this type of tie

5. . . . . .Use the Schacht-Quentmeyer integrated property method.

XNUL. . . . . .Nusselt number for single-phase heat transfer. If positive, the heat transfer in the single-phase regime is calculated based on DCH and the thermal conductivity of the fluid. If negative, |XNUL| is used as a scaling factor on the predictions made using NCSPHERE or NCCYLIN.

XNLM. . . . . .Nusselt number multiplier (see page 3-218, Single-phase Mode)

XNLA. . . . . .Nusselt number bias (see page 3-218, Single-phase Mode)

DCH. . . . . .Characteristic dimension (in meters or feet) for natural convection and other single-phase (nonboiling, noncondensing) flows. If no iport (port path ID) has been input, the DCH is treated as the diameter of an effective sphere in an internal call to NCSPHERE. Otherwise, DCH is treated as the length of an effective cylinder in an internal call to NCCYLIN.

CSF . . . . . Surface roughness factor C_{sf} for pool boiling. See C_{sf} in Table 15.1 of the Handbook of Heat Transfer, 3rd Ed., McGraw Hill, for example. (Caution: values much lower than the default of 0.01 can result in very large heat transfer coefficients.)

SPR . . . . . Prandtl number exponent S for pool boiling. Override the default with 1.0 when using water as the working fluid.

CHFF . . . . . Critical heat flux factor (see page 3-216, Boiling Mode)

HFFL . . . . . Scaling parameter on Leidenfrost flux/temperature (lower limit to film boiling) estimation (see page 3-216, Boiling Mode)

XNB . . . . . Upper flow quality limit on nucleate boiling. Past this quality, the film coefficient will transition to single-phase vapor using interpolation.

THKL . . . . . Thickness (meters or ft) of liquid layer when condensing. Used when HCON is negative. See page 3-218, Condensing Mode.

HCON . . . . . Condensing heat transfer coefficient (W/m^2-K or BTU/hr-ft^2-F). If negative, used as a scaling factor based on |HCON|*K_{liq}/THKL where K_{liq} is the thermal conductivity of the liquid. For mixtures including noncondensible gases, the input or calculated value will be corrected internally for diffusion effects using HTUDIF.

ACCM . . . . . Local acceleration. Defaults to 1g even if the ACCEL vector is zero. For rotating systems, an $r\omega^2$ term may be appropriate. Specifying zero implies zero-gravity, and that liquid can wet the wall at all locations. Otherwise, the primary purpose of this factor is to set the hydrostatic pressure term ρgh (DL_{liq}*ACCM*DEPTH) as needed to suppress or reduce boiling if the wall node is significantly below the surface depth (noting that the PL and TL of the lump are taken to represent the liquid/vapor interface).

DEPTH. . . . . Depth of the wall node beneath the liquid/vapor surface (as represented by the lump state). Used in conjunction with ACCM as noted above to calculate the saturation temperature of the liquid against the wall. When ACCM is nonzero, DEPTH can also affect apportioning to twinned tanks. Negative values are valid, and are interpreted as meaning “this node is above the surface.” A value of zero means that this node is exposed to both phases.

defaults:

- iport . . . . . 0 (no port: enclosed sphere)
- fport 1.0
- DUPN. 1.0
- DUPL. 1.0
- TTWIN . . . . . no twin (single tie created)
- ITAC. . . . . 3 (program-calculated FQ apportioning based on ACCM, DEPTH)
- XOL. 0.0
- XOH. 1.0
- UAM. 1.0
- MODW. . . . . 0 (no high speed correction, but subcooled boiling and superheated condensation are still possible). Nonzero values should be used with caution.
- IPUV. 0
- UVEL. 0.0
- DTEF. 0.0
- MODR. . . . . 0 (bulk fluid static temperature used)
- XNUL. . . . . -1.0
- XNLM. 1.0
- XNLA. 0.0
- DCH. . . . . 1.0 m or 3.28084 ft
- CSF. 0.01
- SPR. . . . . 1.7 (Not appropriate if water is the working fluid. Use 1.0 instead.)
- CHFF . . . . . $\pi/24$
- HFFL. 0.09
- XNB. 0.7
- THKL. . . . . 0.001 m or 3.28084e-3 ft
- HCON. . . . . -1.0
- ACCM. . . . . 9.806 m/s^2 or $32.174*3600^2$ ft/hr^2
- DEPTH 0.0

Guidance: References to processor variables within expressions may refer to “#this” as the tie identifier, “#node” as the identifier of the node, and “#path” as the identifier of the port path, and “#lump” as the identifier of the lump.

Restrictions: *lid and iport must be separately defined in **this** fluid submodel. nid must be defined in a thermal submodel.* The QDOT for lid will be calculated automatically during execution, and will be summed into the Q for nid.

Restrictions: *iport cannot reference a device such as an MFRSET or VFRSET unless a nonzero AF has been supplied for that path.*

Guidance: If lid is an active twinned tank,* then the tie may be twinned as well. If lid is *not* twinned, the twinning of the tie is optional *and the secondary tie will be ignored*. But if lid *is* twinned, the use of a twinned HTP tie is strongly recommended.

If lid is twinned, yet the HTP tie is not twinned, then:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If the tie is twinned (and the user is controlling the FQ of each tie):

For twinned ITAC=1 ties, the AHT, DCH, XNUL, XNLM, and XNLA keywords set the value of the primary (liquid) tie and, by default, that of the secondary (vapor/gas) tie as well. To specify these values independently (including expressions) for the secondary tie, use AHT_V, DCH_V, XNUL_V, XNLM_V, and XNLA_V. Note that these input keywords are not SINDA/FLUINT variables, meaning they do not appear in post-processing (use AHT, DCH, XNUL, XNLM, and XNLA instead) and cannot be referenced in Fortran logic blocks.†

Twinned ITAC=1 ties do not collapse to homogeneous ties, even when the attached lump is not twinned and even when it is single-phase. Each side of the tie permanently addresses a particular phase, whether that phase is present or not. For example, when an attached lump contains 100% vapor or gas (XL=AL=1.0), the primary tie is still active, but normally its FQ will be zero (as set by the user or by the FloCAD Compartment), while the FQ of the secondary tie is normally be unity in that case. This treatment allows expressions (and NEL logic, per the footnote) to be developed and maintained for each phase separately.

FloCAD Compartment ties:

HTP ties in a Compartment are type ITAC=1 by default, in which case the Compartment controls the FQ of the tie automatically. For HTP ties, the Compartment will also automatically update the DCH during a run.

Examples:

```
c closed if in single phase (no port path defined):
T HTP,10,10,HOTPOT.2014, AHT = SURFA


c path 23 is on bottom, flowing out by default (positive FR direction):
T HTP,22,222,VESSEL.2,-23,-1.0
      DCH = CYLINDER_LENGTH
      AHT = AREA_22
```

* See also Section 3.25 and page 3-220.

† They can, however, be adjusted in Network Element Logic (NEL) in FloCAD. For example, use “XNUL#this” in a tie’s NEL to refer to the XNUL of the primary tie, and “XNUL#twin” to refer to the UB of the secondary tie.

```
c water (S=1.0), and assume 1/2 of CHF. Node is at bottom.
T HTP,11,11,BOTTOM.2, SPR = 1.0, CHFF = 0.5*pi/24
      THKL = DEPTH#this
      DEPTH = VESSEL_HEIGHT*(1.0-AL#lump)
      AHT = 0.25*pi*VESSEL_DIAMETER^2


c 1/2g, and depth of node is adjusted based on how full the lump is
T HTP,101,202,SIDE.33, 104, AHT = 1.42e-4
      ACCM = 0.5*gravsi
      DEPTH = LENGTH*(1.0-AL#lump)


c twinned tank and tie example. Might need to calc DEPTH in FLOGIC 0
T HTP,1001,1001,DOUB.1001, 5000, -1.0
      TTWIN = 1002
      DEPTH = 0.0
      AHT = 0.1*Atotal


c twinned tank and tie example, using permanently twinned ITAC=1 ties,
c and addressing each phase independently. Update FQs in logic.
T HTP,1001,1001,DOUB.1001, 5000, -1.0
      TTWIN = 1002, ITAC=1
      DEPTH = 0.0
      AHT = 0.1*Atotal
      XNUL = 10.0 $ liquid-side Nusselt number
      XNUL_V = 2.0 $ keyword only, "XNUL_V" is not a parameter
```

3.6.4 Path-oriented (Segment) Ties

As noted in on page 3-206 (Inherent Downstream-weighted Heat Transfer), coarsely discretized single-phase lines (and steady flows without phase change) tend to underestimate heat transfer rates. This section describes ties that model single-phase heat transfer using segments rather than lumped parameters.

The previously described lumped parameter ties (HTU, HTN, HTNC, HTP) link a node and a lump, and may optionally point to one or more paths, which are used to define size (heat transfer area), shape, and flow rates. The node represents the wall; the lump represents the fluid. In those ties, the lump is viewed as coincident with the node, and each represents the average fluid/wall state. No distinction is made for up or downstream. For HTNC ties, the flow rates and sizes of two paths are used, but the lump is considered centered.

With the alternative *segment tie*, the link is defined between a node and a *path*. Instead of tying to a lump directly, the user ties to a path representing the segment. The node represents the average wall temperature over the entire length of the path, and the endpoint lumps represent the inlet and outlet states. Of course, since heat may only be added to a lump, the tie is really attached to the downstream lump, and the UA and QTIE values simply reflect the augmentation caused by the inclusion of an upstream state. *However, if the flow rate reverses, then by default the tie automatically jumps to the opposite end of the path.* (Alternately, the user may command the tie to stay put on the defined downstream lump by issuing a "STAY" command in the input subblock, as described below.) Because of the potential for flow reversals, the link to the endpoint lumps is somewhat indirect.

Two types of segment ties exist: HTUS and HTNS. With HTUS ties, the analogs of HTU ties, the user is expected to input the UA value. Unlike HTU ties, a path (of any type) is named.

With HTNS ties, the analog of the HTN tie, the program calculates the UA based on convection calculations for the path representing the segment (Figure 3-16).

No segment-oriented analog exists for HTNC and HTP ties.

For segment ties, $QTIE = (Q_{up} - Q_{down})/\ln(Q_{up}/Q_{down})$, where $Q_{up} = U_{up*}A*(T_{node} - T_{lump,up})$ and $Q_{down} = U_{down*}A*(T_{node} - T_{lump,down})$ by default, representing an logarithmic average of the inlet and outlet heat transfer rates. For flows without phase change, U is relatively constant and this solution is equivalent to a log-mean temperature difference (LMTD) solution.* If either lump is two-phase, or if flows reverse and the "STAY" option has been used, the ties revert to downstream-weighted (HTU and HTN) behavior: $QTIE = Q_{down}$. For single-phase flows, if $|Q_{down}| > |Q_{up}|$ then an arithmetic average is used. In the special case of HTUS ties in single-phase flows, the QTIE value will usually exceed $QTIE_{tie} = UA_{tie} \bullet (T_{node} - T_{lump})$ because of the effects of the upstream lump.

Unfortunately, since segment ties represent a different way of looking at the same problem, they are not always easily convertible to or from the lump-oriented type of tie without other model changes.

* Segment ties roughly correspond to one side of a heat exchanger in which the other side is infinite (C_r=0, C_{max}=∞). See Section 7.11.10 for other options for finite heat capacity heat exchanger modeling.

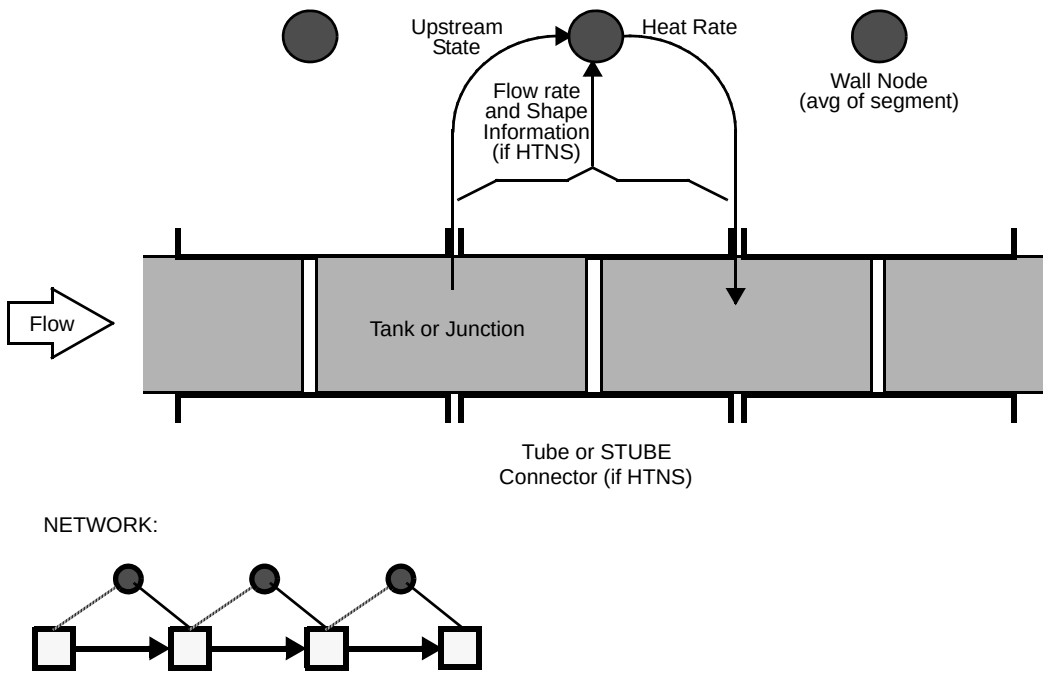


Figure 3-16 Heat Exchanger Sections Using Segment Ties

Lumped Parameter vs. Segment Ties—Improvements of segment ties over lumped parameter ties exist but aren't always significant unless very coarse spatial resolution is used (refer to Sample Problem H). Furthermore, the limiting assumption in such coarse models often becomes that of constant wall temperature. In other words, if the node temperatures vary axially, then the user is forced to discretize as needed to capture this gradient, which in turn drives the selection of the flow model resolution. In addition to wall temperature, the user must be cognizant of other important property variations, such as pressure (for compressible fluids), viscosity, and conductivity. Velocities, densities, Reynolds numbers, and heat transfer coefficients can all vary axially as a result, potentially placing another lower limit on acceptable resolution. Nevertheless, in preliminary designs where models are often very coarse and analyses are all steady-state, these ties can be used to collapse an entire single-phase heat exchanger model into one node, one tie, and two lumps.

While segment ties function adequately for two-phase (including slip flow) and compressible fluids, they can be problematic in models with phase change and tanks if the flow reverses while being cooled or condensed. For example, if by some perturbation a flow rate temporarily reverses in the middle of a two-phase model of a condenser that uses tanks, then the potential exists for two ties to jump to the same tank, virtually assuring its collapse to all liquid. HTNC ties avoid such problems not only by staying put even when flows reverse, but also by averaging velocities axially when calculating UA. The use of the "STAY" command helps alleviate some but not all of this difficulty.

In summary, ***while HTNS ties are preferred in the limit of single-phase or steady flows with no phase change, HTNC ties are recommended for unsteady flows where phase change is possible, or where flow conditions are not known a priori.***

Another potential problem related to reversing flows with these ties is the use of unequal (asymmetric) path duplication factors: DUPI ≠ DUPJ. Recalling that such duplication factors affect the amount of heat transfer area 'seen' by the lump, the value of UA may change by the ratio DUPI/DUPJ for HTNS ties if flows reverse and the "STAY" command has not been issued. A warning is produced at the start of the processor if it detects the potential for such a condition given the *initial* values of DUPI and DUPJ and the absence of the "STAY" command, but no further actions or checks are performed thereafter.

3.6.4.1 HTUS Ties

HTUS ties are analogous to HTU ties in that the user specifies the local UA coefficient instead of letting the program calculate one according to convection heat transfer correlations. However, unlike the HTU tie, a path must be named in the HTUS tie. Unlike convection ties, this path may be of any type, since it is used to define the segment over which the HTUS tie acts: its size and shape (if any) are not used for heat transfer coefficient calculations.

The HTUS tie is unique in that the QTIE value almost never equals UA times the temperature difference of the node and the downstream lump, since the actual QTIE is affected by the temperature of the upstream lump. In HTNS ties, which experience a similar augmentation, the calculated UA value reflects this effect. Since the user provides the local UA value in the HTUS tie, the augmentation appears hidden.

When two-phase flow exists with phase change, the HTUS tie behaves like a HTU tie.

The input (FLOW DATA) format for the HTUS tie is specified below.

HTUS Subblock Format:

```
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]
```

where:

`tid`.....tie id

`pid`.....path id of the path (any type) representing the segment. For twinned paths, pid should reference the primary path of the pair (the involvement of the secondary path is implied). The tie will attach to the lump downstream of this path, moving as needed if flow rates reverse unless the STAY option is in effect (see below).

`nid`.....node id with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam

`UA`.....conductance of the tie. The conductance must be positive or zero.

`DUPN`.....node duplication factor (number of this tie seen by nid)

`DUPL`.....lump duplication factor (number of this tie seen by lid)

`STAY`.....if present, indicates that the tie should remain on the defined downstream end of pid even if the flow rate reverses; otherwise the tie will automatically jump to the current downstream end.

`UAM`.....conductance multiplier (scaling factor)

`MODW`.....Wall temperature method (see Section 3.6.5):

- 0.....TL=TEF and DTEF=0: no high speed corrections, but superheated/subcooled wall state corrections still apply
- 1.....User-specified velocity UVEL
  -1 disables superheated/subcooled wall state corrections
- 2.....User-specified path identifier IPUV
  -2 disables superheated/subcooled wall state corrections
- 3.....User-specified offset temperature DTEF

`UVEL`.....Velocity for calculating TEF. (see Section 3.6.5)

`IPUV`.....Path ID for velocity basis of UVEL (see Section 3.6.5)

`DTEF`.....TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)

`AHT`.....heat transfer area (m^2 or ft^2). If AHT is zero, UA will be used instead.

`UB`.....the base (temperature-independent) portion of the conductance U

`UEDT`.....exponent on temperature difference

`TTWIN`.....id of twinned (vapor or secondary) tie

`ITAC`.....twinned tie apportioning rule

- 1.....user controls both UBs, FQs, etc. (but unlike HTU ties, the secondary the HTUS tie is only active when attached to a secondary twinned tank)
- 2.....AHT (or UA) of primary tie apportioned (using FQ factors) according to tank volume ratio

`DEPTH`.....Distance below liquid surface for ITAC=1 ties (used in FloCAD Compartments)

`XOL`.....lower (liquid) limit for volume fraction apportionment (ITAC=2)

`XOH`.....upper (vapor/gas) limit for volume fraction apportionment (ITAC=2)

defaults:

DUPN 1.0
DUPL 1.0
UA. 0.0
UAM 1.0
AHT . . . . . 0.0 (UA is therefore used instead)
UB. 0.0
UEDT 0.0
MODW . . . . . 0 (No velocity corrections, but may still have local wall state if two-phase)
IPUV 0
UVEL 0.0
DTEF 0.0
TTWIN. . . . . no twin (single tie created)
ITAC . . . . . 2 (AHT or UA of primary tie apportioned according to tank volume ratio)
DEPTH. 0.0
XOL 0.0
XOH 1.0

Guidance: This option allows the user to define the overall heat transfer between fluid heat exchanger segment and a thermal node that represents the average container wall temperature over the segment. The heat will be added or subtracted to the lump instantaneously downstream of pid, and this heat will move to the opposite lump if the flow rate in pid reverses and if 'STAY' has not been specified. The path pid can be of any type. The UA value is held constant over each solution interval. A logarithmic average of the up- and downstream heat transfer rates is used unless either lump becomes two-phase (in flows where phase change occurs) or the flow rate has reversed and the 'STAY' command has been issued, in which case the heat rate is calculated as if this were an HTU tie.

Note: The heat rate through the tie (QTIE) will normally exceed UA*DT for HTUS ties because of upstream gradients.

Guidance: References to processor variables within expressions may refer to "#this" as the tie identifier, "#node" as the identifier of the node, and "#path" as the identifier of the path. Note that "#lump" is illegal in this block since the tie is defined on the basis of a path, not a lump: the attached lump could conceivably change during execution.

Restrictions: *pid must be separately defined in **this** fluid submodel, and cannot be a secondary twin. nid must be defined in a thermal submodel.* The QDOT for the lump downstream of pid will be augmented automatically during execution, and the QTIE will be summed into the Q for nid. If UA=0.0, then the tie heat rate QTIE will be zero.

Guidance: If lid is an active twinned tank,* then the tie may be *optionally* twinned as well.

If tid is *not* twinned:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If tid is twinned, it will connect to both tanks if lid is the primary tank.

When the tanks are separated the user can either control both UAs individually (ITAC=1), or use the default (ITAC=2) "void fraction" apportioning method subject to the upper and lower limits XOH and XOL, respectively. Actually, the program uses the volume ratio of the tanks irrespective of their contents, neglecting for apportionment purposes bubbles in the liquid tank and droplets in the vapor tank. The tie parameter FQ contains the resulting apportionment factor for each tie.

When the tanks are homogeneous (combined), then the secondary twin of the tie is ignored and the primary tie connects to the primary tank.

Examples:

```
T HTUS,10,10,RIBS.2014, STAY, UA=2.0
T HTUS,22,2203,SLAW.2, UA = -FR10*Cp*ln(1.0-effectiv)
```

3.6.4.2 HTNS Ties

HTNS ties are analogous to HTN ties in that the user specifies one path or pair of twinned paths, letting the program calculate one according to convection heat transfer correlations.

When two-phase flow exists with phase change, the HTNS tie behaves like a HTN tie.

The input (FLOW DATA) format for the HTNS tie is specified below.

HTNS Subblock Format:

```
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]
```

* See also Section 3.25.

where:

- `tid` ..... tie id
- `pid` ..... path id of the path representing the segment, to be used for flow rate and geometry (either tube or STUBE connector). Alternatively, the path id of the primary path of a pair of twins (the involvement of the secondary path is implied). The tie will attach to the lump downstream of this path, moving as needed if flow rates reverse unless the STAY option is in effect (see below). For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area: AHT and therefore UA will be zero.
- `nid` ..... node id with which heat will be transferred. This must be input in the format tsmn.nodnam, where tsmn is the thermal submodel name containing the node named nodnam
- `f` ..... fraction of pid wetted area attributed to tie
- `DUPN` ..... node duplication factor (number of this tie seen by nid)
- `DUPL` ..... lump duplication factor (number of this tie seen by lid)
- `STAY` ..... if present, indicates that the tie should remain on the defined downstream end of pid even if the flow rate reverses; otherwise the tie will automatically jump to the current downstream end
- `TTWIN` ..... id of twinned (vapor or secondary) tie
- `UAM` ..... conductance multiplier (scaling factor)
- `MODW` ..... Wall temperature method (see Section 3.6.5):
  - 0 ..... TL=TEF and DTEF=0: no high speed corrections, but superheated/subcooled wall state corrections still apply
  - 1 ..... User-specified velocity UVEL
    -1 disables superheated/subcooled wall state corrections
  - 2 ..... User-specified path identifier IPUV
    -2 disables superheated/subcooled wall state corrections
  - 3 ..... User-specified offset temperature DTEF
- `UVEL` ..... Velocity for calculating TEF. (see Section 3.6.5)
- `IPUV` ..... Path ID for velocity basis of UVEL (see Section 3.6.5)
- `DTEF` ..... TEF offset temperature (TEF = TL + DTEF) (see Section 3.6.5)
- `MODR` ..... Reference state method (see Section 3.6.5):
  - 0 ..... Use TL (static temperature).
  - 1 ..... Use a simple average of TL and T_{wall} as the reference state
  - 2 ..... Use T_{wall}
  - 3 ..... Use Eckert's reference enthalpy.
  - 4 ..... Use van Driest's reference enthalpy.
  - 5 ..... Use the Schacht-Quentmeyer integrated property method.
- `XNUL` ..... Laminar Nusselt number (see Section 3.6.6)
- `XNLM` ..... Laminar Nusselt number multiplier (see Section 3.6.6)
- `XNLA` ..... Laminar Nusselt number bias (see Section 3.6.6)
- `XNTM` ..... Turbulent Nusselt number multiplier (see Section 3.6.6)
- `XNTA` ..... Turbulent Nusselt number bias (see Section 3.6.6)

CDB.....Dittus-Boelter scaling parameter (see Section 3.6.6)
UER.....Reynolds number exponent (see Section 3.6.6)
UEC.....Prandtl number exponent for cooling (see Section 3.6.6)
UEH.....Prandtl number exponent for heating (see Section 3.6.6)
UEV.....Correction factor exponent for viscosity, applies to both laminar and turbulent flow (see Section 3.6.6)
UED.....Turbulent correction factor exponent for density (see Section 3.6.6)
UEK.....Turbulent correction factor exponent for thermal conductivity (see Section 3.6.6)
UECP.....Turbulent correction factor exponent for specific heat (see Section 3.6.6)
UEP.....Turbulent correction factor exponent for pressure (see Section 3.6.6)
UET.....Turbulent correction factor exponent for temperature (see Section 3.6.6)
RLAM.....Lower limit in Reynolds number for transitioning flow: the flow must be laminar if the Reynolds number is below this value.
RTURB .....Upper limit in Reynolds number for transitioning flow: the flow must be turbulent if the Reynolds number is above this value. RTURB should be greater than or equal to RLAM.
CHFF.....Critical heat flux factor (see Section 3.6.7)
HFFL.....Scaling parameter on Leidenfrost flux/temperature (lower limit to film boiling) estimation (see Section 3.6.7)
XNB.....Upper flow quality limit on nucleate boiling. Past this quality, the film coefficient will transition to single-phase vapor using interpolation.

defaults:

f.....1.0
DUPN.....1.0
DUPL.....1.0
TTWIN .....no twin (single tie created)
UAM.....1.0
MODW.....2 (program determines IPUV internally)
IPUV.....0
UVEL.....0.0
DTEF.....0.0
MODR.....3 (Eckert’s reference enthalpy)
XNUL.....3.66
XNLM.....1.0
XNLA.....0.0
XNTM.....1.0
XNTA.....0.0

CDB 0.023
UER 0.8
UEC 0.3
UEH 0.4
UEV 0.0
UED 0.0
UEK 0.0
UECP 0.0
UEP 0.0
UET 0.0
RLAM 1958.785
RTURB. 6421.960
CHFF π/24
HFFL 0.09
XNB 0.7

Guidance: This option specifies the automatic calculation of convective heat exchange between a fluid heat exchanger segment and a thermal node that represents the average container wall temperature over the segment. The heat will be added or subtracted to the lump instantaneously downstream of pid, and this heat will move to the opposite lump if the flow rate in pid reverses and if 'STAY' has not been specified. A standard set of convection correlations are used, including correlations for boiling and condensing. The path named pid will be used to provide geometry (diameter, perimeter, flow area, length, and heat transfer area) and flow rate information to the correlations. If a twinned pair of paths is used, only the ID of the primary path need be input: the involvement of the secondary path is implied. If two-phase flows with phase change are being analyzed, the use of the default IPDC=6 is recommended for the path. The area fraction is applied to the final UA value, and hence may be used to model nonuniform temperatures, fouling, etc. A logarithmic average of the up- and downstream heat transfer rates is used unless either lump becomes two-phase (in flows where phase change is possible) or the flow rate has reversed and the 'STAY' command has been issued, in which case the heat rate is calculated as if this were an HTN tie.

Guidance: References to processor variables within expressions may refer to "#this" as the tie identifier, "#node" as the identifier of the node, and "#path" as the identifier of the path. Note that "#lump" is illegal in this block since the tie is defined on the basis of a path, not a lump: the attached lump could conceivably change during execution.

Restrictions: *pid must be separately defined in* ***this*** *fluid submodel. nid must be defined in a thermal submodel.* The QDOT for the lump downstream of pid will be augmented automatically during execution, and QTIE will be summed into the Q for nid. If f=0.0, the tie will be adiabatic.

Restrictions: *pid cannot reference a secondary twin or a path type other than a tube or STUBE.* For compatibility with duct macros, a REDUCER or EXPANDER connector can also be named, but if so it will contribute no heat transfer area: AHT and therefore UA will be zero.

Warning: If 'STAY' is not specified, then the tie can jump to either lump, and the DUPI and DUPJ options have the same magnifying effect on heat transfer area as with HTN and HTNC ties. An initial warning is produced if DUPI is not equal to DUPJ.

Guidance: If lid is an active twinned tank,* then the tie may be twinned as well. If either lid or pid are *not* twinned, the twinning of the tie is optional *and the secondary tie will be ignored*. But if lid and pid *are* twinned, the tie must also be twinned.

If the path is not twinned, a singlet tie may be used and:

If lid is the primary (liquid) twin, the tie will remain attached to it at all times.

If lid is the secondary (vapor) twin, the tie will attach to it when it is active (when the twinned tanks are in the split mode), but will otherwise move temporarily to the primary (liquid) twin when the tanks are in the combined mode.

If lid and pid are both twinned, then the tie must be twinned and will connect to both tanks.

When the tanks are separated the user cannot control apportionment, since this is handled automatically as a function of the underlying flow and heat transfer regime. The code uses ITAC=3 to denote this automatic control. The tie parameter FQ contains the resulting apportionment factor for each tie.

When the tanks are homogeneous (combined), then the secondary twin of the tie is ignored and the primary tie connects to the primary tank.

Examples:

```
T HTNS,10,33,RIBS.2014,  DUPL=2.0,STAY, DUPN=2.0
T HTNS,11,33,ROLLS.1133,0.75
T HTNS,22,222,SLAW.2, STAY
```

* See also Section 3.25.

3.6.5 Effective Fluid Temperature TEF

For LSTAT=NORM (default) lumps in high speed flows (Mach number above 0.1 or so), the adiabatic surface temperature (also called “adiabatic wall temperature”) should be used instead of the static temperature TL as the “zero point” in heat transfer calculations. This temperature is between the static and stagnation temperature, though closer to the stagnation temperature. In the limit of a calorically perfect gas:

$$T_{as} = TL + r*(T_{stag}-TL)$$

where T_{as} is the adiabatic wall temperature, TL is the fluid static temperature, and T_{stag} is the total or stagnation temperature. The *local recovery factor* “r” is calculated as $Pr^{1/2}$ for laminar flow and $Pr^{1/3}$ for turbulent flow, where Pr is the Prandtl number. (An enthalpy basis is actually used internally for real gases and other more complex working fluids.)

Ties interconnect a node and a lump. Unfortunately, given FLUINT network rules, a lump cannot be assigned a characteristic velocity (other than zero for stagnant lumps) for calculating the stagnation temperature, much less the designations “laminar” and “turbulent.” Such are characteristics of paths, not lumps. This distinction presents no inconveniences when HTN, HTNC, or HTNS ties are used, since those ties are already associated with one or more paths and FLUINT therefore has enough information to complete the calculations. Still, the adiabatic wall temperature is therefore a function of the tie, and not the lump.

TEF is the effective fluid temperature. *For low speed flows, TEF will often be the same as TL for the tied lump.* For high speed flows, TEF will be higher than TL by an amount DTEF (also a tie variable: $TEF_{tie} = TL_{lump} + DTEF_{tie}$). TEF will therefore represent T_{as} in such cases. It may also differ from the lump TL in other ways, and therefore is not always the same as T_{as}. For example, if a volatile species is present in a subcooled liquid state (XL=AL=0.0, $TL<T_{sat}$) and the wall is hotter than saturation, TEF may represent the saturation temperature T_{sat} as needed to model subcooled boiling (see Section 3.6.7). Similarly, if a condensible species is present in a superheated vapor state (XL=AL=1.0, $TL>T_{sat}$), TEF may represent the saturation temperature T_{sat} as needed to model superheated condensation.

For rotating paths with walls that move relative to each other (RVR<1), the TEF calculation will take into account high rotational speeds (tangent velocities), as described in Section 3.26.4.2.

For HTN, HTNC, and HTNS ties, the above calculations are automatic by default, although they can be customized. For HTU, HTUS, and HTP ties, which are not associated with any path, no such calculations are performed by default, although the user can modify this selection by providing extra information in the form of a velocity, a path, or an offset (DTEF). These choices are regulated by the MODW integer control parameter:

MODW = 0 . . . Use TL (static temperature) for nonvolatile/noncondensible fluids (DTEF is therefore zero in such cases). This is the default for HTU and HTUS ties. Subcooled and superheated local wall state corrections might still apply.*

MODW = 1 . . . User-specified velocity UVEL (turbulent flow assumed in the recovery factor calculation).
To apply only this high speed correction while disabling two-phase (superheated/subcooled) corrections, use MODW=-1

MODW = 2 . . . User-specified path identifier IPUV whose velocity is used to determine UVEL. This is the default for HTN, HTNC, and HTNS ties, in which case IPUV is *not* a user parameter since it is already known.
To apply only this high speed correction while disabling two-phase (superheated/subcooled) corrections, use MODW=-2

MODW = 3 . . . User-specified offset temperature DTEF (equal to TEF - TL).

In all of the above cases except MODW=3, the adiabatic surface temperature may be overridden by the saturation temperature as needed to model subcooled boiling and superheated condensation. If this is not desired, setting negative MODW overrides all such phase change calculations. In cases where DTEF or UVEL are calculated, they are available as output variables for inspection.

Examples:

```
T HTN, 1, 2, wall.3, 4        $ path 4 is IPUV, all rest is automatic
T HTNC, 3, 4, hotw.5, 6,1.,7  $ paths 6 and 7 define UVEL, IPUV unusable
T HTN, 30, 40, spot.50, 60, MODW = 3, DTEF=0.0 $ TL=TEF (NO corrections)
T HTU, 5, 44, steel.55, MODW=1, UVEL=Veloc5 $ user-specified velocity
T HTU, 5, 44, steel.55, MODW=2, IPUV=504 $ user-specified path
```

* Use either MODW=-1 with UVEL=0.0, or MODW=3 with DTEF=0.0, to disable all wall state corrections.

3.6.6 Advanced Single-phase Forced Convection Heat Transfer Options

Single-phase forced convection heat transfer in HTN, HTNC, and HTNS ties is calculated either as a constant Nusselt number for laminar flow (defaulting to Nu_{lam}=3.66), or via the Dittus-Boelter correlation (Section B.1, Section 7.11.3) for turbulent flow (defaulting to $Nu_{turb} = 0.023*Re^{0.8}*Pr^{0.4}$). In the transition region (approximately 2000 < Re < 6500 by default), Hausen's method is used (Section B.1).

The user has the ability to extensively customize these relationships for both laminar and turbulent flow* as needed to model noncircular ducts, rough walls (Section 7.11.4.4), entrance length effects (developing thermal flow, Section 7.11.4.6), compact heat exchangers (Section 7.11.11.1), coiled tubes (Section 7.11.4.5), enhanced heat transfer (Section 7.11.4.7), cryogenic and near-critical fluids with strong property variations, etc.

For laminar flow, the Nusselt number can be overridden using the XNUL tie parameter, which defaults to 3.66. A multiplier and bias also exist for the laminar Nusselt number: Nu = XNLM*XNUL + XNLA. Furthermore, a viscosity correction exponent UEV can be applied as a correction, yielding the final relationship:

$$Nu_{lam} = XNLM*XNUL*(\mu_r/\mu_{wall})^{UEV} + XNLA$$

where μ_r is the reference viscosity (usually the same as that calculated at the film temperature) divided by the viscosity at the wall, μ_{wall}. UEV defaults to zero, but many references list it as 0.14 for liquids when the freestream temperature is used as the reference point (i.e., MODR=0 as described below). UEV applies to *both* laminar and turbulent flow.

* Laminar applies to Re < RLAM, and turbulent to Re > RTURB. In between (RLAM < Re < RTURB), a numerical smoothing is used to preserve continuity in order to avoid convergence or time step issues. This smoothing is numerically complex, and is not based on any specific correlation. Rather, it attempts to fit a second order polynomial to the curve of heat transfer coefficient versus Re within the transition zone. Continuity at the two endpoints (laminar, turbulent) is required, of course, but the fit also attempts to match the slope at the turbulent transition. If it cannot achieve that result with a monotonic increase in heat transfer coefficient within the transition range, then zero slope at the laminar transition point is used as a fallback position.
Either way, this smoothing is made for numerical expediency, and it does not correspond to the scaling within the default Hausen correlation. Therefore, even a very minor change in a parameter (say CDB = 0.0231 instead of the default 0.023) will invoke this transitional smoothing, resulting in potentially very different predictions within the transition zone.

The reference state is the temperature and pressure at which all baseline properties (including Reynolds and Prandtl numbers) are calculated. This choice is controlled by the MODR parameter:

MODR = 0. . . . Use TL (static temperature). This was the default for Version 4.5 and earlier.
MODR = 1. . . . Use a simple average of TL and T_{wall} as the reference state
MODR = 2. . . . Use T_{wall}
MODR = 3. . . . Use Eckert's reference enthalpy. This is the default choice. For low speed flow, this is the same election as MODR=1. For high speed flow, the reference temperature is 0.32*TL + 0.18*T_{as} + 0.5*T_{wall} (in the limit of calorically perfect gases at least).
MODR = 4. . . . Use van Driest's reference enthalpy. For low speed flow, this is the same election as MODR=1. For high speed flow, the reference temperature is 0.28*TL + 0.22*T_{as} + 0.5*T_{wall} (in the limit of calorically perfect gases).
MODR = 5. . . . Use the Schacht-Quentmeyer integrated property method. This method does not define a single reference state, but rather specifies that each property be calculated as a temperature-weighted average over the interval TL to T_{wall}. Specifically, p = int(p*dT)/(T_{wall}-TL) where *p* represents any fluid property and *int()* represents an integral.

Caution: For most correction exponents listed below (UEV, UET, etc.) the default of MODR=3 should be overridden with MODR=0. Check the basis of the correlation chosen.

For turbulent flow, the user can address all of the coefficients of the Dittus-Boelter coefficient, plus apply correction exponents, plus apply a multiplier (XNTM) and a bias (XNTA). Thus:

$$Nu_{turb} = XNTM * Nu_{DB} + XNTA$$

where for heating (T_{wall} > TEF):

$$Nu_{DB} = CDB * Re^{UER} * Pr^{UEH} * (\mu_r/\mu_{wall})^{UEV} * (\rho_r/\rho_{wall})^{UED} * (k_r/k_{wall})^{UEK} * (Cp_{avg}/Cp_{wall})^{UECP} * (P/P_{crit})^{UEP} * (T_r/T_{wall})^{UET}$$

For cooling (T_{wall} < TEF), the Prandtl exponent becomes UEC instead of UEH.

As a sample usage, for liquid metals: XNTA = 4.8, CDB = 0.0156, UER= 0.85, and UEC=UEH=0.93 (isothermal wall, Notter and Sleicher), and for laminar flow XNUL=4.1807 and XNLA=-0.0439*Pe where Pe is the Peclet number (Pe=RePr).

Notice that almost all properties (including Nu, Re, Pr) are calculated at the reference state except as noted. For the UECP correction term, note that the specific heat is calculated as an average* (Cp_{avg} = 0.5*Cp_r + 0.5*Cp_{wall}), and that the static pressure is used in the UEP term.

* This is strictly true only for MODR=1, and other values of MODR dictate other schemes for determining "CP_{avg}." For example, with MODR=5, this average is replaced by an integral.

The tie parameters for single-phase heat transfer are summarized below:

MODR . . . . . Reference state method (default: 3 for Eckert's reference enthalpy)
XNUL. . . . . Laminar Nusselt number (default: 3.66)
XNLM . . . . . Laminar Nusselt number multiplier (default: 1.0)
XNLA. . . . . Laminar Nusselt number bias (default: 0.0)
XNTM . . . . . Turbulent Nusselt number multiplier (default: 1.0)
XNTA . . . . . Turbulent Nusselt number bias (default: 0.0)
CDB . . . . . Dittus-Boelter scaling parameter (default: 0.023)
UER . . . . . Reynolds number exponent (default: 0.8)
UEC . . . . . Prandtl number exponent for cooling (default: 0.3)
UEH . . . . . Prandtl number exponent for heating (default: 0.4)
UEV . . . . . Correction factor exponent for viscosity, applies to both laminar and turbulent flow (default: 0.0)
UED . . . . . Turbulent correction factor exponent for density (default: 0.0)
UEK . . . . . Turbulent correction factor exponent for thermal conductivity (default: 0.0)
UECP . . . . . Turbulent correction factor exponent for specific heat (default: 0.0)
UEP. . . . . Turbulent correction factor exponent for pressure (default: 0.0)
UET . . . . . Turbulent correction factor exponent for temperature (default: 0.0)
RLAM. . . . . Laminar is indicated below this Reynolds number (default: about 2000)
RTURB. . . . . Turbulent is indicated above his Reynolds number (default: about 6500)

Examples:

```
C use a viscosity correction
T HTN, 1, 2, wall.3, 4, MODR=0, UEV=0.14


C use Schacht-Quentmeyer effective properties, with slightly different
C coefficients in the DB equation
T HTNC, 1, 2, wall.3, 4, 0.5, 5, 1.0
      MODR=5, CDB = 0.025, UEC=0.333, UEH= 0.333
```

3.6.7 Advanced Two-phase Forced Convection Heat Transfer Options

This section details the methods used for forced convection boiling (vs. pool boiling) heat transfer, though brief mention of superheated condensation is included as well. Section 3.6.7.3 provides a summary of the sequence of calculations that are described in the first two sections.

3.6.7.1 Subcooled Boiling and Superheated Condensation

The TEF parameter (Section 3.6.5) is primarily applicable to high speed flows, but is also involved in the subcooled boiling and superheated condensation regimes. For subcooled boiling, it is important to note that incipient superheat (the "excess temperature" needed to nucleate bubbles) is assumed to be zero. *This assumption means that the onset of nucleate boiling (ONB) will occur as soon as the wall temperature exceeds saturation.* Setting MODW to -1 or -2 will disable all boiling before the bulk fluid is saturated (liquid), and will similarly disable all condensation until the bulk fluid is saturated (vapor). MODW can therefore be controlled by the user as needed to represent incipient superheat. The rest of this section describes actions taken when MODW is defaulted to 2 in HTN, HTNC, and HTNS ties.

If the fluid is saturated, the TEF parameter will represent this saturated temperature. If the wall is hotter than saturation and the fluid is still subcooled, both the heat transfer coefficient and the TEF will be proportioned according to the temperature difference ratio:

$$U_{eff} = U_{liquid} + A_{2P}*(U_{boil} - U_{liquid})$$
$$TEF = TL_{liquid} + A_{2P}*(T_{sat} - TL_{liquid})$$

where (Xc being the mass fraction of the condensible species in the mixture):

$$A_{2P} = [(T_{wall} - T_{sat})/(T_{wall} - TL_{liquid})]^{10**(-4Xc)}$$

In other words, A_{2P} represents the fraction of the wall that is boiling; the remainder is subcooled.* This apportioning method has been shown to roughly correspond to Tu and Yeoh's correlation for water.† At the entrance of a liquid-flowing pipe with heat applied, the TEF parameter will begin to depart from that of the fluid temperature, approaching the saturation temperature. The subcooled boiling regime is denoted by an "S" in the "2P" column of the TIETAB routine (Section 7.11.8).

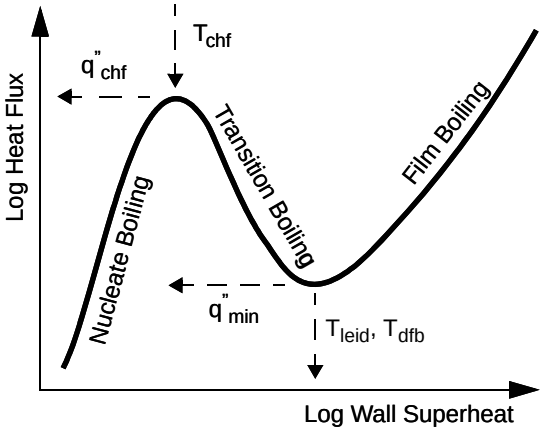
The above treatment also applies analogously to superheated condensation. The superheated condensation regime is also denoted by an "S" in the "2P" column of the TIETAB routine (Section 7.11.8). *This mode may also apply to HTU, HTUS, and HTP ties,* but without affecting UA since that is user-provided. Rather, for those ties "S" in TIETAB indicates that TEF is saturation, and not

* If the wall rises above T_{dnb} (the temperature for departure of nucleate boiling), the transition and film boiling regimes begin. In this case, A_{2P}=1.0 so TEF=T_{sat} and U_{eff} =U_{boil}. Predictions are questionable in this regime because the film and transition boiling correlations (see Section 3.6.7.2) are not meant to cover subcooled boiling, and also because FLUINT lacks an "inverted annular" regime with vapor surrounding a liquid core. This situation sometimes arises in analyses of cryogenic quenching: a cold liquid injected into a room temperature duct. Note that T_{dnb} is always less than the critical temperature for the volatile species.

† Tu and Yeoh, "On Numerical modeling of low-pressure subcooled boiling flows." International Journal of Heat and Mass Transfer, 2002.

the bulk fluid temperature TL. Setting MODW to -2 disables this phase change correction for HTN, HTNC, and HTNS ties. The routine TIE2P or the output parameter NTWOP (Section 3.6.7.3) can be used to query this status for any tie, if this information is required in logic blocks.

3.6.7.2 Critical Heat Flux (CHF) and Post-CHF Heat Transfer



FLUINT uses Chen's nucleate boiling correlation, as described in Section B.2, for low quality (X<XNB) flows with low wall superheats. 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 temperature, T_{leid}, or the departure from film boiling temperature, $T_{dfb,}$ whichever is smaller), 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.

The critical heat flux (CHF) is estimated and post-CHF heat transfer coefficients are estimated if the CHF is exceeded.* The CHF is estimated using a modified version of Zuber's pool boiling correlation, subject to a user scaling factor CHFF (a tie parameter defaulting to $\pi/24$):†

$$q''_{CHF} = CHFF \cdot \rho_v h_{fg} \left[\frac{\sigma g(\rho_l - \rho_v)}{\rho_v^2} \right]^{\frac{1}{4}} \left(\frac{\rho_l + \rho_v}{\rho_l} \right)^{\frac{1}{2}} (1-\alpha) \left(\frac{D_{8mm}}{D} \right)^{\frac{1}{2}}$$

This Zuber-Griffith correlation is modified by scaling with the diameter (Wong, 1996). The above estimation is further modified using a subcooling correction suggested by Gambill.‡ ***The above estimate is often in error (both high and low) by up to an order of magnitude.*** As with all boiling calculations, strong caution is advised due to the large uncertainties involved in estimating these parameters: sensitivity studies and other assessments of risk should be performed.

---

* The CHENNB routine calculates these factors and communicates to the STDHTC routine via common blocks. If either of these routines is provided in SUBROUTINES, this communication will be disrupted and the critical heat flux will be assumed infinite. The CHFF and HFFL factors will be ignored. If necessary to override these routines in SUBROUTINES, contact CRTech for common block formats.

† N. Zuber, "On the Stability of Boiling Heat Transfer," Trans. ASME, 1958. The (1-α) factor corresponds to the Zuber-Griffith 1977 correlation.

‡ As listed in Seader et al, NASA CR 243, "Boiling Heat Transfer for Cryogenics."

To assist in such sensitivity studies, a negative value of CHFF is interpreted to invoke Griffith's 1957 correlation (best-estimate curve-fit values, per Krieth, Basic Heat Transfer, 1980). In this mode, CHFF = -1.0 means to use a multiplier of unity on Griffith's estimate, and CHFF = -2.0 means to use twice the critical heat flux, etc. The differences between CHFF = $\pi/24$ (the default modified Zuber-Griffith correlation) and CHFF = -1.0 (the optional Griffith correlation) can be used as a measure of the uncertainties involved, but the user is urged to use significant margin on top of these estimates (whether high or low) in the absence of more specific correlations or data.

If the wall temperature is high enough such that the CHF is exceeded, transition boiling or film boiling is simulated. The film boiling correlation for low quality flows is based on *external* boiling on a horizontal cylinder, but is nonetheless applied as an approximation* for flow *within* a duct:

$$U_{FB} = 0.62\left[\frac{k_v^3 \rho_v g(\rho_l - \rho_v)(h_{fg} + 0.4C_{p,v}\Delta T)}{L_h \cdot \mu_v \Delta T}\right]^{\frac{1}{4}}$$

When applied to external film boiling, L_h is the cylinder diameter. To instead apply to internal flows, the smaller of DH/2, the Helmholtz ("Taylor Instability") wavelength $(2\pi[\sigma/(\rho_l-\rho_v)]^{1/2})$, and the recommendation of Leonard† is used:

$$L_{h,max} = 8.646\left[\frac{\sigma^4 \cdot h_{fg}^3 \cdot \mu_v^5}{\rho_v \cdot (\rho_l - \rho_v)^5 \cdot g \cdot k_v^3 \cdot \mu_v \cdot \Delta T^3}\right]^{\frac{1}{11}}$$

For higher quality flows, Groeneveld's correlation‡ is used if the resulting coefficient is larger, making sure the estimate is also at least as large as a single-phase vapor correlation (Dittus-Boelter) would yield.

Radiation, important at very high wall temperatures, is included as a correction to the calculation of the above film boiling coefficient assuming unit emissivity for the droplets and a wall emissivity of 0.1.**

* This classic correlation, originally by Bromley, is the basis of many other film boiling correlations, including channel (internal) flow. For various flow situations, the Bromley correlation is used with a more specific estimation of the critical thickness, L_h. Because of the potential for large uncertainty, an advanced user override is available for film boiling. Contact CRTech for details about "USERFB" if this is a concern.

† J.E. Leonard et al, "Low-Flow Film Boiling Heat Transfer on Vertical Surfaces, Part 2 Empirical Formulations and Application to BWR-LOCAD Analysis," Solar and Nuclear Heat Transfer, AIChE Symposium Series, No 164, Vol 73, 1977. Corrected using J.G. Andersen, "Low-Flow Film Boiling Heat Transfer on Vertical Surfaces, Part 1 Theoretical Model," Solar and Nuclear Heat Transfer, AIChE Symposium Series, No 164, Vol 73, 1977.

‡ Per Rohsenow, Handbook of Heat Transfer, 3rd Ed., p15.134, 1998.

** Lindon Thomas, Heat Transfer, p. 606, 1993.

To establish and maintain full film boiling, the wall temperature must exceed the Leidenfrost point, corresponding to a minimum heat flux of:*

$$q''_{MIN-FB} = HFFL \cdot \rho_v h_{fg} \left[\frac{\sigma g(\rho_l - \rho_v)}{(\rho_l + \rho_v)^2} \right]^{\frac{1}{4}}$$

where again, HFFL is a tie parameter that is useful as a scaling factor. Unfortunately, the above estimate is too high, and must be reduced for wall temperatures approaching the critical point. The lower if this point and the departure from film boiling, T_{dfb}, is used based on estimates from Ramilison and Leinhard:

$$T_{dfb} = 0.97 \cdot T_{crit}\left(0.932 + 0.077\left(\frac{T_{sat}}{T_{crit}}\right)^9\right) + 0.03 \cdot T_{sat}$$

There is no agreed-upon correlation for the Leidenfrost point, and the values measured can be a strong function of surface finish and other properties not included in the above estimate. As with other calculations in this section, ***uncertainties can be very large***, with measured Leidenfrost temperatures differing on the order of 100K for water at the same pressure on different surfaces, as an example.†

If the above limit is employed, then HFFL loses the ability to scale the minimum flux. Furthermore, if this limit is less than the temperature corresponding to the critical heat flux, CHFF may too become irrelevant as a scaling factor.

Above the CHF wall temperature and below the Leidenfrost temperature (or T_{dfb}), transition boiling exists. In this highly under-characterized and chaotic regime, the heat transfer coefficient is estimated using a nonlinear interpolation between nucleate boiling and film boiling based on scaling laws proposed by Ramilison and Leinhard.‡

The transition boiling regime is denoted by a “T” in the “2P” column of the TIETAB routine (Section 7.11.8). The film boiling regime is denoted by an “F” in the same column. These indicators can be queried in logic blocks using the TIE2P routine, or via the output parameter NTWOP (Section 3.6.7.3).

Note that transition and film boiling can occur if the fluid quality is zero (100% and perhaps subcooled liquid).

Even if the fluxes and wall temperatures are low, if the flow is sufficiently dry, then nucleate boiling will no longer dominate as dry spots form and thin liquid films superheat and evaporate off their surfaces. The default limit to nucleate boiling is assumed to represent a flow quality of 70%

* N. Zuber, “On the Stability of Boiling Heat Transfer,” Trans. ASME, 1958.
† J.D. Bernadin et al, “The Leidenfrost Point: Experimental Study and Assessment of Existing Models,” Trans. ASME, Vol. 121, p. 984, 1999.
‡ As listed in Rohsenow, Handbook of Heat Transfer, 3rd Ed., p .15.71, 1998.

(XL=0.7 for homogeneous steady flows). Past this point, the heat transfer coefficient is degraded by simple interpolation into single-phase vapor heat transfer. The user can adjust this limit using the XNB tie parameter, which defaults to 0.7.

To summarize the boiling heat transfer tie parameters:

CHFF . . . . . scaling parameter on CHF (upper limit to nucleate boiling) estimation (default: π/24). Negative values signal an alternate correlation, and are applied (in absolute value) as a scaling factor on the results of that correlation.

HFFL . . . . . scaling parameter on Leidenfrost flux/temperature (lower limit to film boiling) estimation (default: 0.09)

XNB. . . . . upper flow quality limit on nucleate boiling (default: 0.7). Past this quality, the film coefficient will transition to single-phase vapor using interpolation.

Guidance: Significant hysteresis exists in the boiling curve, especially for constant flux boundary conditions. If the wall temperature exceeds that corresponding to the critical heat flux (CHF), it will likely continue to rise to full film boiling, and will be unlikely to return to lower temperatures (i.e., nucleate boiling) without a significant reduction in flux. Numerical perturbations in steady state solutions can therefore artificially result in a film boiling solution. Double check answers by using large CHFF (>1.0e20) to disable CHF calculations and post-CHF reductions in heat transfer, then lower CHFF and repeat the solution in the same run using the colder wall temperatures as initial conditions. Using a register-based expression, such as "CHFF= *CHfactor**pi/24," represents a convenient mechanism to change many ties at once:

```
CHfactor = 1.0e30
CALL STEADY     $ run with disabled CHF calcs
CHfactor = 1.0
CALL STEADY     $ see if cold walls persist
```

3.6.7.3 The NTWOP Status Code

NTWOP is an integer output tie parameter that indicates the two-phase heat transfer status of HTN, HTNC, HTNS, and HTP ties. NTWOP is zero for single-phase or nonvolatile flows.

For condensation, values are as follows:

-1 . . . . . superheated condensation

-2 . . . . . saturated condensation

For evaporation, values of the singles digit are as follows:

1. . . . . subcooled nucleate boiling

2. . . . . saturated nucleate boiling

3. . . . . transition boiling

4. . . . . film boiling

However, in the case of evaporation, it is also possible for the heat transfer to be limited by exceeding XNB (defaulted to a quality of 0.7) simultaneously with other phenomena. This is indicated by a “1” in the tens digit.

For example, NTWOP=12 indicates that the regime is saturated nucleate boiling, but that the quality and void fraction are high enough that the wall is starting to dry out. Therefore, these additional values of NTWOP can exist:

11 . . . . . subcooled nucleate boiling, with partial wall dry-out (rare)
12 . . . . . saturated nucleate boiling, with partial wall dry-out
13 . . . . . transition boiling, with partial wall dry-out
14 . . . . . film boiling, with partial wall dry-out

3.6.7.4 Tie Algorithm for Boiling Heat Transfer

This section provides a pseudo-code explanation of how the boiling logic is performed: Table 3-11. Refer to the two prior sections for definitions of the parameters named in this table.

This logic is contained within the default routines STDHTC and CHENNB, with other correlations referenced as well (DITTUS, HTCMIX). Refer to Section 7.11.3 for descriptions of these routines.

Table 3-11 Boiling Heat Transfer Calculation for HTN, HTNC Ties

X is the flow quality (may or may not be lump XL), and FR is the mass flow rate

Calculate U_{nb}, the nucleate boiling coefficient, using CHENNB at a quality of MIN(X,XNB)
Calculate U_{vap}, the vapor film coefficient, at a flow rate of X*FR using DITTUS
Calculate U_{fb}, the film boiling coefficient (see Section 3.6.7.2), using U_{vap} if it is greater than U_{fb}
Calculate T_{dfb}, departure of film boiling temperature, roughly 0.9*T_{crit} (see Section 3.6.7.2)

If $T_{wall} > T_{dfb}$, then
  use $U_{boil} = U_{fb}$
else
  Calculate T_{leid}, the Leidenfrost temperature (roughly $q''_{min}/U_{fb} + T_{sat}$, where q''_{min} is based on HFFL)
  If $T_{wall} > T_{leid}$, then
    use $U_{boil} = U_{fb}$
  else
    Calculate q''_{chf} based on CHFF (different correlations depending on sign of CHFF)
    Calculate q''_{nb} based on U_{nb}
    If $q''_{nb} > q''_{chf}$, then
      use U_{boil} = transition boiling based on T_{wall}, $T_{leid,}$ T_{chf} (Ramilison and Leinhard, Section 3.6.7.2)
    else
      use $U_{boil} = U_{nb}$ (nucleate boiling)
    end if
  end if
end if

If subcooled boiling ($T_{wall} > T_{sat}$ but TL < T_{sat} and X=0), then
  Calculate U_{liq}, the liquid film coefficient based on all liquid (full FR)
  Update U_{boil} by interpolation between U_{boil} and U_{liq} based on temperatures (see U_{eff}, A_{2P}, Section 3.6.7.1),
    using an exponent based on fraction of volatile, if mixture
else if Dry (X > XNB), then
  Update U_{boil} by interpolation between U_{boil} and U_{vap} based on X
endif

If liquid is a mixture, then
  Update U_{boil} by interpolation between U_{boil} and HTCMIX based on fraction of volatile liquid
endif

3.6.8 How Ties are Effected

This section contains an overview of how and when heat transfer calculations are performed. It is best used as a source of reference material since much of the discussion, while specific to ties, requires an understanding of the material in Section 4 as a prerequisite.

Ties make energy disappear from one submodel and reappear in another. Lump QDOT terms and node Q terms are affected. The heat rate through the tie, QTIE, is summed into the QDOT term of the lump (multiplied by DUPL) and subtracted from the Q term of the node (multiplied by DUPN). Unlike path duplication factors, if *both* DUPL and DUPN are zero, the heat transfer calculations for such ties are suspended.

When a lump is tied, the program automatically updates the QDOT value for that lump, leaving the user one opportunity to modify QL for tanks and junctions independently. Ties are effected (and QTIEs, UAs, UBs, UEDTs, and QDOTs updated) between FLOGIC 0 and FLOGIC 1 in all routines.

The rules governing Qs for tied nodes are slightly less clear. The nodal term is summed into (or subtracted from), rather than replaced. This summation occurs after VARIABLES 0 but before VARIABLES 1 in all routines: ties are treated like time-varying sources. Recall that nodal Qs are reset every solution step, unlike lump QLs. Because VARIABLES 0 is called only once at the beginning of a steady-state solution (STEADY or STDSTL), nodal Qs are reset to their post-VARIABLES 0 (e.g., time-dependent) value at the start of every iteration, and the effect of ties is then summed into those values. However, for stability reasons tied lumps are treated like boundary nodes from the point of view of the thermal solution during steady-states—even though the Qs are updated rigorously, they are irrelevant in steady-states. Thus, the UA values are used by the thermal solution in steady-states, but the QTIE values are used in transients. Even in transients, however, the effective UA value is summed into the CSGMIN calculation for stability reasons.

When using the TIETAB output routine or network mapping routines, it is often possible that the QTIE value does not exactly correspond to the UA value multiplied by the temperature difference, even for converged solutions. The cause of this apparent discrepancy is often that the QTIEs were calculated based on lump and node states at the beginning of the solution interval, rather than on the values that appear in output listings at the end of the solution interval. Other reasons include internal damping, etc. For HTUS ties, the discrepancy is inherent since the UA value is user-provided but the QTIE value reflects the augmentation of the upstream state.

3.7 Fties (Fluid to fluid Heat Exchange)

Fties (fluid to fluid ties) may be used to establish heat transfer directly between lumps, and are therefore directly analogous to linear SINDA conductors. However, fties cannot extend between fluid submodels: they must attach to lumps within the same fluid submodel.

There are two common uses for fties: (1) axial conduction or "back conduction" of heat through the fluid itself, which is otherwise ignored as negligible compared to advection, and (2) direct contact or thin-walled fluid-to-fluid heat exchange within the same fluid system, such as happens in regenerators. With the exception of highly conductive fluids (such as liquid metals and superfluid helium) or systems with extremely small flow rates (such as loop heat pipes), axial conduction is normally negligible: fties are not often required.

Guidance: The lump output variable QDOT for each lump reports the total heat rate on that lump, which is the sum of QL plus the effects of any ties and any fties that are currently active: QDOT = QL + ΣQTIE + ΣQF + QTM + QCHEM, where QTIE is the heat rate through a tie and QF is the heat rate through an ftie (rectified according to the directionality of the ftie). This section only deals with QF effects on QDOT.

Restriction: Both endpoint lumps on an ftie must reside within the current submodel. Otherwise, to tie two lumps together that exist in different submodels, an intermediate node is required and ties (not fties) must be used.*

There are three types of fties: USER, AXIAL, and CONSTQ as described below.

All fties have a conductance "GF" (in the same units as the G of a linear conductor or the UA of a tie), and a heat rate "QF" (positive from the first defined lump to the second). All fties also have duplication factors DUPC and DUPD which apply to the first (c^{th}) and second (d^{th}) lump, respectively, in order of their listing in the ftie subblock:

$$QF_{ftie} = GF_{ftie} \cdot (TL_c - TL_d)$$

Excepting CONSTQ fties, fties may also be twinned. AXIAL fties *must* be twinned if used along with twinned tanks and twinned paths, as will be described below.

3.7.1 USER Fties

User fties are similar to HTU ties. The user has complete control over GF, and the resulting QF value is available for inspection or for use in expressions.

* For example, assume two lumps from different models must be interconnected "directly" using a conductance of "U." Create an intermediate SINDA arithmetic node, and connect to it from each of the lumps with an HTU (or perhaps HTUS) tie of conductance 2*U (taking into account the ties will be in series). Note that all three submodels must be built, that to satisfy SINDA network construction rules a small but nonzero conductor (say, G=1.0E-10) must also be attached to the node that was introduced.

Auxiliary routines are available to help calculate GF in special cases such as conduction through porous media (cylindrical or planar), perhaps including the effects of heat exchange between the liquid and the porous media on the conduction term. See WETWICK, PLAWICK, and CYLWICK in Section 7.2.

USER Subblock Format:

```
FT USER,id,lc,ld, GF=R, [,DUPC=R][,DUPD=R][,FTWIN=I]
```

where:

`id`..... ftie id
`lc`..... lump id with which heat will be transferred. This is the c^{th} lump and QF will be positive when heat is flowing from it.
`ld`..... lump id with which heat will be transferred. This is the d^{th} lump and QF will be negative when heat is flowing from it.
`GF`..... conductance of the ftie. The conductance must be positive or zero.
`DUPC` ..... lump c duplication factor (number of this ftie seen by lc)
`DUPD` ..... lump d duplication factor (number of this ftie seen by ld)
`I` ..... id of the twinned ftie to generate

defaults:

`DUPC` 1.0
`DUPD` 1.0
`FTWIN`..... no twin (single ftie created)

3.7.2 CONSTQ Fties

CONSTQ fties extract heat from one lump (at a constant rate) and inject it into another lump irrespective of temperature differences. The user supplies the QF value. The GF value is ignored.

CONSTQ fties can be used *between* a pair of twinned tanks (see below), *but cannot themselves be twinned*. If generated between a set of twinned tanks as part of the definition of those tanks, the QF will be calculated automatically (and perhaps in conjunction with vaporization/condensation modeling in the superpath, as described in Section 3.25).

CONSTQ Subblock Format:

```
FT CONSTQ,id,lc,ld, QF=R [,DUPC=R][,DUPD=R]
```

where:

id . . . . . ftie id
lc . . . . . lump id with which heat will be transferred. This is the c^{th} lump and QF will be positive when heat is flowing from it.
ld . . . . . lump id with which heat will be transferred. This is the d^{th} lump and QF will be negative when heat is flowing from it.
QF . . . . . heat flow through the ftie
DUPC. . . . . lump c duplication factor (number of this ftie seen by lc)
DUPD. . . . . lump d duplication factor (number of this ftie seen by ld)

defaults:

DUPC. 1.0
DUPD. 1.0

3.7.3 AXIAL Fties

The AXIAL ftie is an aid in simulating axial conduction along a duct (within the fluid itself). The user assigns a tube or STUBE connector to the ftie, and the properties of that path (flow regime, shape: TLEN, AF, AFI, AFJ) and of the attached lumps (fluid conductivity, void fraction) are used to automatically compute the GF of the AXIAL ftie.

The effects of AFI and AFJ will be taken into account. The upstream fraction UPF will also be used to calculate the effective conductance. If two-phase flow is present, the total thermal conductivity will be assumed to be a weighted fraction of phasic thermal conductivity based on void fraction.

For twinned tanks, axial conduction is assumed to take place in parallel within each phase (i.e., within each parallel twinned ftie) if the flow regime is stratified or annular. When the regime is bubbly or slug, however, the vapor phase is not continuous and hence no conduction occurs along it. In bubbly or slug flow, the *conductivity* used for the vapor and liquid twin will be the same, although the *conductance* (GF) itself may differ between the two according to the volumetric fractions of each phase.

As an input convenience, AXIAL fties can be generated in parallel to any tube or STUBE by specifying “FTIE = AXIAL” in any appropriate block (tube, STUBE connector, GENPA, LINE, or HX macro, or even within path defaults). The generated ftie will have the same ID as that of the tube or STUBE connector, and will automatically extend between the same lumps and the path DUPI and DUPJ factors will be used as the DUPC and DUPD factors of the AXIAL ftie.

Restrictions:

1. If generated independently (i.e., not using the “FTIE=AXIAL” method but using the FT subblock listed below), then the tube or STUBE connector must connect to the same lumps as does the ftie. This rule is satisfied automatically for AXIAL fties generated in the definition of the path.

2. For independently generated AXIAL fties, the DUPC and DUPD ftie factors must be the same as the corresponding path DUPI and DUPJ factors (DUPC = DUPI *or* DUPJ, depending on the order in which they were input). This rule is satisfied automatically for AXIAL fties generated in the definition of the path.

3. If *either* of the endpoint lumps is twinned, and the tube or STUBE connector is also twinned, then the ftie must also be twinned. However, if the tube or STUBE is twinned and the endpoint lumps are not, then a single (untwinned) ftie may be used. Or, if the endpoint lump is twinned but a homogeneous path connects just to one of the tanks, then the ftie similarly attaches to the same tank and need not be twinned. These rules are obeyed automatically by the "FTIE=AXIAL" generation option. If the tube or STUBE is twinned and attaches to a twinned tank, then the ftie will be twinned (using the same ID as the tube or STUBE twin).

Guidance: *Use the FTIE=AXIAL format preferentially.* The independent subblock format is available for customization and for consistency, but is otherwise cumbersome.

AXIAL Subblock Format

To generate an AXIAL ftie independently of a tube or STUBE connector:

```
FT AXIAL,id,lc,ld, IDTUBE=I [,DUPC=R][,DUPD=R][,FTWIN=I]
```

where:

`id`..... ftie id
`lc`..... lump id with which heat will be transferred. This is the c^{th} lump and QF will be positive when heat is flowing from it.
`ld`..... lump id with which heat will be transferred. This is the d^{th} lump and QF will be negative when heat is flowing from it.
`IDTUBE` .... identifier of the tube or STUBE to use as the basis of the AXIAL ftie
`DUPC` ..... lump c duplication factor (number of this ftie seen by lc)
`DUPD` ..... lump d duplication factor (number of this ftie seen by ld)
`FTWIN`..... id of the twinned ftie to generate

defaults:

`DUPC` 1.0
`DUPD` 1.0
`FTWIN`..... no twin (single ftie created)

Restriction: If either lc or ld is twinned and the tube or STUBE connector is twinned, then the ftie itself must be twinned. Otherwise, a single ftie can be used in parallel with twinned paths if the endpoint lumps are not twinned tanks.

AXIAL as part of a tube or STUBE connector or Duct Macro subblock:

To generate an AXIAL ftie within tube or STUBE or duct macro (Section 3.9.2) blocks, simply use the command:

```
[,FTIE = AXIAL]
```

This command will cause either an ftie or perhaps twinned fties to be automatically generated in parallel with each tube or STUBE connector as needed to model axial conduction. The generated ftie will use the same id as the path, and the duplication factor values will be inherited from the corresponding path DUPI and DUPJ factors.

3.8 Interface (iface) Elements and Subvolumes

When lumps are interconnected by paths, mass and energy flow between the lumps according to pressure drops and other forces.

Tanks (and only tanks*) may also be interconnected by interface elements (interfaces or *ifaces* for short), a FLUINT network building block. Unlike a path, no mass flows through an iface. Rather, an iface describes the wall between the two tanks and how it behaves. It tells the program that two adjacent control volumes (tanks) share a common boundary, and describes how their pressures and volumes are interrelated.

For example, consider the simplest and most common iface: a FLAT iface. Placing a FLAT iface between two tanks means that the two tanks share an infinitely flexible boundary. Pressure differences between the two tanks will be essentially zero, since any change in pressure in one tank would cause the interface to shift: one tank will grow and the other will shrink until pressures are equalized. Implicit in this behavior is that the volumetric growth of one tank is equaled by the shrinkage of the other: the volume of the pair of tanks is conserved.

In other words, if a path (such as a LOSS connector) had been placed between the two tanks, mass would have moved from one to another until the pressures were equal, but the control volume boundaries would not have changed. Instead, when a FLAT iface is placed between them, the boundary moves but no mass crosses the interface. Of course, ifaces and paths can be used concurrently.

Physically, a FLAT iface can be envisioned as a piston with zero mass (by default) and zero resistance, or as a infinitely flexible membrane, as depicted in Figure 3-17.

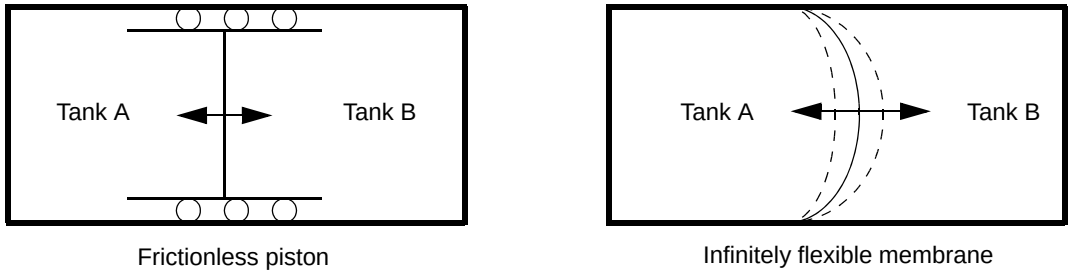


Figure 3-17 Examples of Modeling with a FLAT iface

The user may optionally add mass to any iface, thereby converting it from an instantaneous element to a time-dependent one with inertia. For example, the mass of the piston in the above figure could be added, creating a lag between the build-up of pressure forces and the motion of the boundary according to Newton's law (F=ma). The mass of an iface is zero by default.

* Since tanks are treated as junctions in STEADY ("FASTIC"), ifaces do not exist in STEADY. See Section 3.8.6.

Of course, despite these physical examples, ifaces are mathematical concepts that can be used to model a variety of phenomena. For example, a FLAT interface might be used to define a liquid/vapor interface with negligible curvature, or might be used to arbitrarily subdivide a larger control volume into smaller ones for certain modeling purposes such as thermal stratification.

By default, the volumes of either tank may grow arbitrarily large or small. However, the user may also set an upper and/or lower limit on the volumes. When the upper or lower limit is reached, the iface is deactivated until pressure forces or other changes return it to within the active range.

IFACEs may only be placed between two tanks within the same fluid submodel. The TANK_LINKER utility (Section 7.11.9.3) is available for linking tanks in different submodels.

3.8.1 Subvolumes

In the above example, the two tanks interconnected by an iface form a *subvolume*, or subdivided control volume. If additional ifaces had been added to either of the two original tanks, the newly connected tanks would have been included in the subvolume. In other words, *there is no limit to the number of tanks that can be interconnected by ifaces, forming an arbitrarily complex subvolume*.

The total volume of a subvolume will be conserved. In addition, if the subvolume is entirely interconnected by massless FLAT ifaces, the pressures of all the tanks in the subvolume will be essentially equal.

The concept of a subvolume is important to the operation of ifaces. During the solution, the code keeps track of subvolumes formed by the presence or absence of ifaces. Since ifaces can be activated and deactivated during a solution, the subvolume to which a tank is currently assigned can change. The user can check to see the current subvolume assignment via standard output routines such as IFCTAB (Section 3.8.6.7), but normally this information is important only for verifying inputs or debugging problems, and not for modeling purposes.

Actually, too many active ifaces over-prescribe the problem mathematically. Consider the three lump system depicted in the left of Figure 3-18. Because an interface already exists between tanks 1 and 2 and another between 2 and 3, a subvolume is already defined so the addition of a third iface between 1 and 3 is redundant. Turning off (inactivating) or deleting any one of the ifaces (as depicted at the right of Figure 3-18) makes the problem once again mathematically consistent. While the user can activate or deactivate ifaces at any time during the solution using duplication factors (Section 3.8.5), the code internally activates and deactivates ifaces as well. In fact, FLUINT detects over-specification automatically and will deactivate any redundant ifaces. Therefore, the user need not worry about accidentally adding too many ifaces in a complex (perhaps 2D or 3D) subvolume.

Because more than one iface between any two pairs of tanks is similarly redundant and over-specifies the problem, all but one will be deactivated by the program. However, if in doing so the program detects that the extra ifaces were not identical (and therefore it cannot decide which to use), an error will be issued and execution will terminate. The only reason for adding more than one iface between any two tanks is to be able to use different types of ifaces at different times during a single solution. Therefore, if this is the case it is the user's responsibility to deactivate the unneeded iface(s).

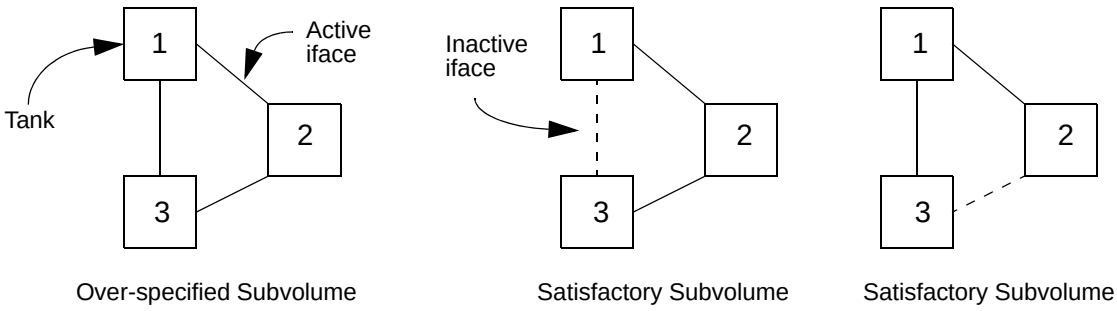


Figure 3-18 Deactivization of Redundant ifaces

3.8.2 Interface Equations

3.8.2.1 Pressure/Volume Relationship

Each active iface defines a relationship between the volume (VOL) of the tanks it connects and their pressures (PL). For a massless generic (NULL) iface, this relationship is an algebraic (time-independent) equation based on five parameters. For the i^{th} iface that interconnects tank "a"* with tank "b," this relationship is:

$$VA_i \cdot \Delta VOL_a + VB_i \cdot \Delta VOL_b + PA_i \cdot \Delta PL_a + PB_i \cdot \Delta PL_b + FPV_i = 0$$

The above relationship essentially defines VA, VB, PA, PB, and FPV. These terms and the above equation are analogous to the formulation of the NULL connector, which uses parameters like GK, HK, EI, EJ, etc. *Like the NULL connector, the NULL iface is rarely used.* Rather, it represents access to the underlying equations for the advanced user.

Instead, almost all modeling needs can be met by defining an iface *device*, such as the aforementioned FLAT iface. When a non-NULL device type has been assigned, the user may still inspect or modify the underlying parameters VA, VB, PA, PB, and FPV in FLOGIC 1, but the program will calculate those parameters automatically based on the iface device type and other inputs, as described in Section 3.8.3. *Therefore VA, VB, PA, PB, and FPV can be ignored by all but the most advanced user.*

For example, consider once more the FLAT iface. Neglecting body forces and inertia, its pressure-volume relationship is described by PL_a=PL_b, which expressed in differential form for a time step becomes:

$$\Delta PL_a - \Delta PL_b = PL_{b,\,current} - PL_{a,\,current}$$

* Tank A, which is defined first in FLOW DATA, becomes the reference lump for the iface.

and therefore, for a FLAT iface:

$$PA_i = 1$$
$$PB_i = -1$$
$$FPV_i = PL_{a,\,current} - PL_{b,\,current}$$

3.8.2.2 Inertia (Mass) Effects

By default, all ifaces are massless and hence the boundary between the two tanks moves instantaneously. If needed, the inertia of the boundary (or perhaps of the fluid forming the boundary, or of the fluid that must move when the boundary moves, etc.) may be modeled via the EMA parameter.

EMA defaults to zero. Otherwise, it may be specified for any iface (including NULL ifaces) as the mass of the boundary divided by its surface area squared. Hence, in standard FLUINT units EMA has units of either kg/m^4 or lb_m/ft^4. For example, to model a piston of mass M and cross sectional area A, set $EMA = M/A^2$. This mass is attributed only to movements resulting from changes in the volume of Tank A (the reference lump for the iface), so if the areas are unequal, "A" refers to the area on the Tank A side of the iface.

If EMA is nonzero, the above equation is treated as a pressure relationship, resulting in the differential equation in time:

$$VA_i \cdot \Delta VOL_a + VB_i \cdot \Delta VOL_b + PA_i \cdot \Delta PL_a + PB_i \cdot \Delta PL_b + FPV_i = EMA_i \cdot \frac{d^2}{dt^2} VOL_a$$

3.8.2.3 Conservation of Volume

Volume is conserved within a subvolume. In a network consisting of N active ifaces which form M subvolumes, there will exist N-M pressure volume equations as defined above, plus M "conservation of volume" equations for each subvolume. Summing over each of the j^{th} tanks in any one of the M subvolumes yields:

$$\sum_j (\Psi_j \cdot \Delta VOL_j) = \sum_j [\Psi_j \cdot (VDOT_j \cdot \Delta t + VOL_j \cdot COMP_j \cdot \Delta PL_j)]$$

where Δt is the time step, and Ψ represents a multiplication factor that takes into account duplication factors (Section 3.8.5), and is normally unity.

For a subvolume consisting of a simple pair of tanks (connected by a single iface) with unit duplication factors and zero VDOT and COMP, the above relationship reduces to:

$$\Delta VOL_a + \Delta VOL_b = 0$$

3.8.2.4 Volume Limits

With any iface (other than a NULL iface), the user may elect to specify a maximum and/or minimum value of the volume of the reference tank, Tank A. VOL_a will be restricted to be less than VHI (which defaults to 1.0E30) and/or a minimum value of VLO (which defaults to zero). If an iface hits one of these limits, and pressure forces are such that it will not move off of it, then the iface is internally disconnected and deactivated. This may delete the subvolume from the current list. If Tank A or Tank B are connected to other tanks via other ifaces, this deactivation may cause a subvolume to be subdivided into smaller subvolumes. If pressure forces later change such that Tank A moves off the stop (would tend to grow more than VLO or less than VHI), the iface is automatically reactivated by the code.

VLO and VHI apply only to Tank A, the reference lump for the iface. However, it is presumed in a simple (two-tank) subvolume that an upper limit of VHI on Tank A defines a lower limit on Tank B, and that a lower limit of VLO on Tank A defines an upper limit on Tank B:

$$V_{tot} = VOL_a + VOL_b$$
$$VOL_{b,\,min} = V_{tot} - VOL_{a,\,max} = V_{tot} - VHI$$
$$VOL_{b,\,max} = V_{tot} - VOL_{a,\,min} = V_{tot} - VLO$$

3.8.3 Interface Device Types

The previous examples have used the FLAT iface, which is the most common option. However, other more complicated iface types are also available to suit special modeling needs, as described in this section.

3.8.3.1 FLAT iface

A massless FLAT iface means that there should be no pressure difference between tanks, other than small errors caused by computational approximations and perhaps by hydrostatic (body force) terms. The pressure relationship for a FLAT iface is therefore PL_a=PL_b, or differentially:

$$\Delta PL_a - \Delta PL_b = PL_{b,\,current} - PL_{a,\,current}$$

Other than VLO, VHI, and EMA, a FLAT iface has no additional *descriptors* or modeling parameters.

FLAT ifaces are useful for modeling liquid/vapor interfaces* with negligible curvature (negligible surface tension forces), very flexible membranes, frictionless pistons, or even as general arbitrary boundary between two control volumes: FLAT ifaces may be used to subdivide a control volume in one or two or three dimensions.

* No attempt is made to adjust the control volume boundaries etc. based on the phase of connected tanks. Such processes represent mass transfer, and must be accomplished using paths or other tools.

Subdividing a control volume is useful when the user wishes to keep phases and/or temperatures separate (i.e., thermal stratification). However, this does not mean that the problem is then treated by FLUINT two or three dimensionally: no CFD-like 2D or 3D velocity field calculations are invoked using ifaces, since ifaces cannot transfer fluid. If there is strong fluid movement within a control volume, then it should be considered perfectly mixed. *Interfaces are intended instead for thermal or phasic subdivision of quasi-stagnant control volumes.*

If inertia is added to the FLAT iface via the parameter EMA (Section 3.8.2.2), then the pressure difference between the tanks will not equalize immediately, but rather will tend to equalize over time. This simulates perhaps a horizontal piston or liquid slug, or perhaps the mass of fluid that must be moved if the boundary moves. To simulate a mass that imposes a force on the interface such as a vertical piston, an OFFSET iface should be used as described next.

3.8.3.2 OFFSET iface

An OFFSET iface is similar to a FLAT interface, but allows the user to specify a pressure difference, DPAB, between tanks (in addition to those caused perhaps by body force terms). The pressure relationship for an OFFSET iface is therefore $PL_a=PL_b+DPAB_i$, or differentially:

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,current} - PL_{a,\,current}) + DPAB_i$$

In addition to VLO, VHI, and EMA, an OFFSET iface has the single descriptor: DPAB. Setting DBAB=5.0 results in Tank A being 5 units of pressure higher than Tank B. An OFFSET with DPAB=0.0 is identical to a FLAT iface.

OFFSET ifaces are useful for modeling vertical pistons with weight, the weight of liquid slugs in a small diameter line, capillarity within a subsaturated wick (see also Section 3.8.3.4), etc. Since DPAB may be modified during a solution (either in logic or using dynamic registers), other modeling purposes may be met by this generic device.

3.8.3.3 SPRING iface

A SPRING iface is similar to an OFFSET interface, but instead of a constant offset term DPAB, the pressure difference between tanks (in addition to those caused perhaps by body force terms) follows a spring relationship. In a linear spring, the force is equal to a displacement off of the equilibrium (zero force) position: $F = k(x-x_o)$. Dividing by cross sectional area squared yields a pressure/volume relationship: $\Delta PL = k'(VOL-VOL_o)$, where $k' = k/A^2$ (for SI units ... see footnote on next page for ENG units).

Tank A is used as a reference. Using "DPDV" as the volumetric spring constant (k' in the above equation*) and VZRO as the volume of Tank A at which there is no pressure drop between it and Tank B, the pressure relationship for a massless SPRING iface is therefore $PL_a = PL_b + DPDV_i*(VOL_a-VZRO_i)$, or differentially:†

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,current} - PL_{a,\,current}) + DPDV_i \cdot (VOL_a - VZRO_i)$$

A SPRING iface has five descriptors: DPDV, VZRO, VLO, VHI, and EMA.

VZRO can be any value, including negative. Along with DPDV, it simply determines the spring force line in point/slope form.

DPDV has units of either N/m^5 or psi/ft^3. DPDV can be any value. Setting DPDV=0.0 (in which case the value of VZRO is irrelevant) makes a SPRING iface identical to a FLAT iface.

VLO and VHI may be used to represent mechanical stops, as depicted in Figure 3-19.

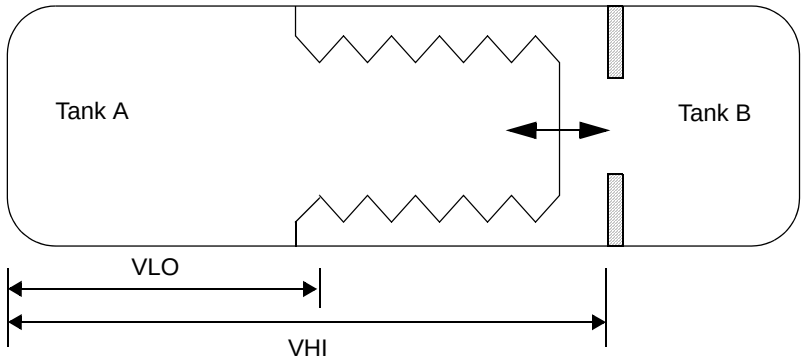


Figure 3-19 Bellows Representation of a SPRING iface

If a SPRING iface hits one of the stops, and pressure forces are such that it will not move off of it, then the iface is internally disconnected and deactivated. If pressure forces later change such that Tank A moves off the stop (would tend to grow more than VLO or less than VHI), the SPRING is automatically reactivated by the code.

Other uses for SPRING ifaces include modeling bellows accumulators, elastomeric bladders, meniscus spring action (see also Section 3.8.3.4), and certain servo controlled valves. Since DPDV, VZRO, VLO, and VHI can all be changed during the solution, other modeling needs may also be served.

---

* In SI units, PL is in Pascals (N/m^2), volume VOL has units of m^3, area A is in m^2, and the spring rate k is in N/m. However, in ENG units, PL is in psi (lb_f/in^2), volume VOL has units of ft^3, area A is ft^2, and the spring rate k is often lb_f/in, so DPDV = k' = $k/(12A^2)$ such that units of psi/ft^3 result.

† Actually, not shown is the VA term: $VA_i = -DPDV_i$

To add mass (inertia) to the spring, use the EMA parameter as with any iface.

Caution: "Floating" Springs: Tank A is the reference by which VLO, VHI, and VZRO are defined. However, if Tank A has nonzero compliance (COMP) or nonzero volumetric rate (VDOT), or if Tank A is connected to any tank other than Tank B via another iface, then the spring may appear to "float" relative to the volume of Tank B. In other words, the spring rate equation will be obeyed relative to Tank A but may be violated from the perspective of Tank B: the spring will appear to have moved from the viewpoint of Tank B. If the user is modeling a real spring-like device with fixed mounts, and if it is not desired to zero COMP and VDOT for Tank A and delete other ifaces, then it may be necessary to switch Tank A with Tank B. (This of course presuming Tank B has zero COMP, VDOT, and no other active ifaces and hence can be used as a reference.) To switch the tanks, the input order on the iface definition must be changed, and the values of VLO, VHI, and almost certainly VZRO will need to be changed to reference the new Tank A. Similarly, the value of DPDV should change sign if Tank A and B are reversed.

If the COMP value has been added in order to simulate the compressibility of liquid (perhaps via the COMPLQ routine), consider instead using a compressible liquid via the 6000 series FPROP DATA option COMPLIQ (Section 3.21.7).

3.8.3.4 WICK iface

A WICK iface is principally used to model a liquid-vapor interface within or near a capillary structure such as a porous material. It is often used in parallel with CAPIL connectors (Section 3.5.3) and CAPPMP macrocommands (Section 3.9.3) to more completely model a section of wick, and in fact a wick iface can be directly linked to those elements.

When the wick is flooded, it behaves similar to a FLAT iface since no meniscus exists.* When the wick is subsaturated (liquid/vapor interface deep within the wick), it behaves similar to an OFFSET iface with the pressure difference between the tanks dominated by the capillary wicking force $2\sigma/r_c$, where σ is the surface tension and r_c is the effective 2D pumping radius of the wick. When the wick is saturated and the meniscus is greater than r_c but is not infinite (the flooded or flat interface state), then the WICK iface is similar to a nonlinear SPRING interface within its active range.

Tank A is used as a reference. By default, Tank A should contain liquid and Tank B should contain vapor for the wick to be active. For Tank A to be the designated vapor lump, the user should instead specify "VAPOR=A" in the definition of the WICK iface, otherwise VAPOR=B is the default.

In addition to VLO, VHI, and EMA, a WICK interface has many descriptors: RCAP, RBP, VZRO, VSUB, XIL, XIH, and ICAP.

* Actually, some curvature (and therefore some pressure difference) can be assigned to the flooded state using the RBP factor described later.

RCAP represents the effective maximum two dimensional pumping radius of the wick. If Tank A is the liquid tank, and if the volume of Tank A ever drops below VSUB (the subsaturated volume), the meniscus is assumed to be maximum and constant (unlike real wicks, which exhibit hysteresis and liquid within them does not recede or fill as a flat front), resulting in the following pressure relationship:

$$VOL_a < VSUB_i$$

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,current} - PL_{a,\,current}) - 2\sigma / RCAP_i$$

VZRO defines the point of volume A where the wick is fully saturated, and the meniscus is absent (the interface is flat). At this point, the same relationship holds as is used in the FLAT interface if RBP is infinite (the default 1.0E30):

$$VOL_a > VZRO_i$$

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,current} - PL_{a,\,current}) - 2\sigma / RBP_i$$

Unlike a SPRING iface, VZRO therefore has physical meaning and cannot be zero or negative, and in fact it must be greater than VSUB. *There is no default value for VZRO. If Tank A is the designated liquid lump, then VZRO must exceed VSUB.*

The above relationships assume the default configuration with Tank B representing the vapor side of the wick. If instead VAPOR=A is specified, then VSUB must exceed VZRO since they are now defined relative to the vapor side of the wick instead of the liquid side.

If the difference between VZRO and VSUB is very small, either small time steps or jittery pressure differentials can result.

RBP is the characteristic minimum curvature in the meniscus even while flooded: the “back-pressure” meniscus. RBP may be any positive value, but it is normally much larger than RCAP and therefore produces a much smaller pressure difference. It defaults to infinity (1.0E30), which means zero pressure difference between liquid and vapor in a flooded state. While RBP may have physical meaning based on the characteristic dimensions of the vapor space, its main use is to avoid artificial flooding as might be caused if capillary paths are in parallel to the iface and the liquid tank is subcooled. For this usage, RBP should be chosen to create a backpressure greater than the error of the solution (Section 3.8.6.8): about 0.01 psid or 70 Pa.

Again assuming Tank A is the designated liquid lump, then if the volume of Tank A is greater than VSUB but less than VZRO, the WICK iface behaves like a spring. The pressure force is prorated according to the current position based on a cubic spline function, as depicted in the middle of Figure 3-20.*

$$VSUB_i < VOL_a < VZRO_i$$

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,curr} - PL_{a,\,curr}) - f(RCAP_i, RBP_i, VZRO_i, VSUB_i, VOL_a)$$

These three modes of behavior are summarized in Figure 3-20 assuming the default VAPOR=B. Note that if the vapor tank is instead used as the reference (VAPOR=A), a left-to-right mirror image of the curve presented in Figure 3-20 would apply, since VSUB must then exceed VZRO and the vapor side of the WICK will have higher pressures than the liquid side.

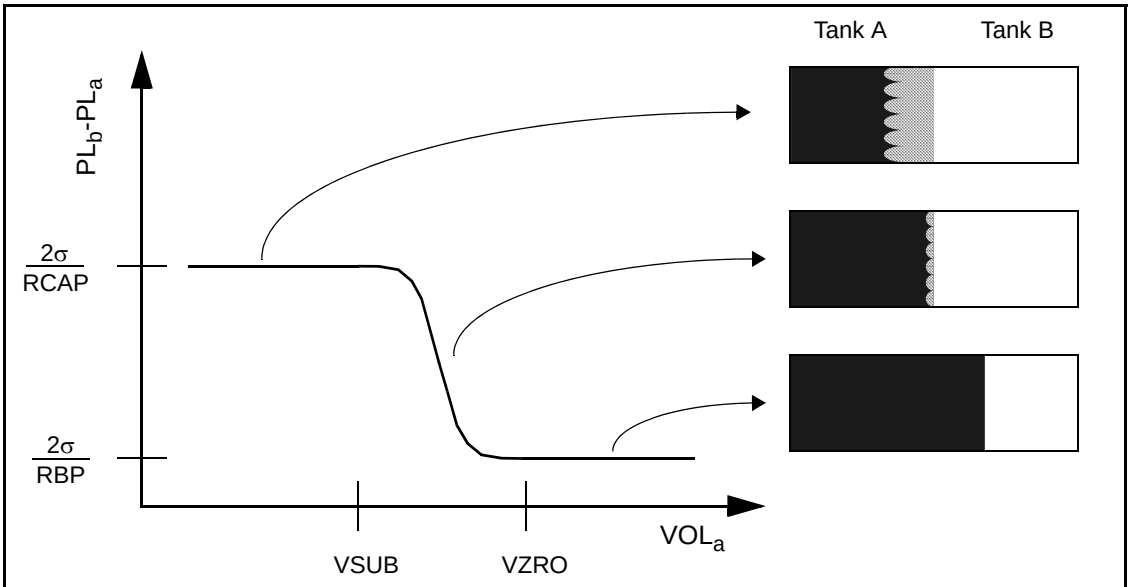


Figure 3-20 Behavioral Modes of a WICK iface with Tank B Containing Vapor

XIL and XIH describe the upper and lower hysteresis band limits on quality for the iface to be active. They inactivate the iface if the vapor tank is too wet or if the liquid tank is too dry for a meniscus to be present. If the quality of the liquid tank (Tank A by default) ever exceeds XIH or if the quality of the vapor tank (Tank B by default) ever drops below XIL, the iface is inactivated. To be reactivated, the quality of the liquid tank must drop below XIL and the quality of the vapor tank must exceed XIH.

A wick with large RCAP and RBP (say, 1.0E30) behaves like a FLAT interface. Neither RCAP nor RBP can be zero or negative (unlike the RC in a CAPIL or CAPPMP, in which a nonpositive value signals infinite capillary pressure).

Although the WICK iface is somewhat specific to its namesake application, it might also be used as a spring with no limits.

Perfect gases (8000 series fluids) cannot be used alone as working fluids with a WICK iface, since they alone do not permit surface tension values to be input.

If RCAP is defaulted to infinity (1.0E30), then RBP can be used to represent an arbitrary capillary interface between a liquid and vapor/gas phase, with RBP perhaps being adjusted in logic blocks.

* Once again, a VA term is missing for clarity, as is the EMA term and other secondary effects.

The EMA (inertial term) can be used to model the inertia of the liquid contained within the wick if this effect is deemed important. In this case, the EMA term can be calculated as the mass of the liquid within the wick divided by the square of the fluidic surface area of the wick (approximately equal to the total surface area times the porosity of the wick). When the wick is flooded, however, the total (not fluidic) surface area should be used instead.

Note that, like all ifaces, a WICK iface does not represent any transport of mass, but rather the shifting of a control volume boundary. Other modeling tools* must be used to represent phenomena such as the flow resistance of a wick (perhaps using a CAPIL connector) or the vaporization at the surface of the wick (perhaps using a CAPPMP macro).

ICAP: Association with a CAPIL connector or CAPPMP Macro: A common usage is to place a WICK iface in parallel with a CAPIL connector or a CAPPMP macrocommand. A CAPIL (Section 3.5.3) represents the mass flow through the wick, perhaps zero if a meniscus is present. A CAPPMP macro (Section 3.9.3) performs this feat and perhaps capillary pumping (vaporization at the meniscus) as well. (In such models, RBP should be specified to prevent artificial flooding of the vapor tank due to the assumption of perfect mixing within it.)

To facilitate this usage, the user may associate a WICK iface to a CAPIL connector or CAPPMP macro. To do so, the numeric path identifier of the CAPIL or the identifier of the first NULL in a CAPPMP macro can be listed in the WICK iface as the ICAP value. This association is permanent: ICAP cannot be changed in logic. A value of zero for ICAP (the default) means no such association has been made.

Although the WICK iface should in general extend between the same tanks as does the associated CAPIL or CAPPMP, no effort is made to enforce this rule to permit modeling flexibility. In fact, no effort is made to assure than the vapor side of the CAPPMP is Tank B for the WICK iface. Therefore, it is the user's responsibility to assure that a correct association has been made.

When an association is made, the RCAP value for the WICK iface is taken as the RC value of the associated CAPIL or CAPPMP (provided RC is positive, *otherwise it is ignored*). Also, the XIH and XIL values are taken from the analogous XVH and XVL values of the associated CAPIL or CAPPMP.

Association with a CAPIL or CAPPMP is more than just a sharing of common data. When an association is made, the internal pressure drop through the wick is taken into account in the behavior of the WICK iface. Although this internal pressure drop represents a static shift in the pressure/volume curve presented in Figure 3-20, it can have important effects when modeling transient events including flooded or depleted wicks.

Slight (about 4-8%) hysteresis is used in the determination of wicking pressure within CAPIL and CAPPMP models for numerical stability. A larger amount (up to 50%) is often present in real porous structures. This means that a CAPIL (for example) won't deprime (i.e., let vapor pass) until a delta pressure of slightly more than $2\sigma/RC$ has occurred, and once deprimed it won't reprime (i.e., stop vapor flow) until the delta pressure drops slightly under $2\sigma/RC$.

* C&R Technologies maintains class notes on modeling capillary systems using basic building block elements in FLUINT. Please contact CRTech for a copy (see title page of this manual for contact information).

Often RCAP is the same value as the RC of a parallel CAPIL or CAPPMP, perhaps through an association via ICAP. If this is true, then a CAPIL or CAPPMP might not deprime (or if it does it might immediately reprime) due to strong adverse pressure differences until the liquid side of the WICK iface drops to its lower limit or the vapor side grows to its upper limit, as determined by VLO and VHI. This models depletion of the wick in a one-dimensional fashion.* When the WICK iface hits such a volumetric limit (VLO or VHI), it will be deactivated (movement of the liquid front will stop) and the pressure drop will likely then exceed $2\sigma/RC$, letting vapor backflow to the liquid tank through the CAPIL or CAPPMP.†

If the pressure difference then drops below $2\sigma/RC$, the iface may be reactivated and the liquid front begins to recede into the wick. However, until the pressure drop falls below the $2\sigma/RC$ minus a few percent (i.e., the lower end of the hysteresis limit), vapor will continue to backflow through the CAPIL or CAPPMP. In other words, the liquid tank might grow until it again reaches the outer dimensions of the wick (VSUB) before the wick is fully saturated, and the pressure difference drops enough to reseal opened vapor passages and thus reprime the capillary device.

Movement of the liquid front causes secondary pressure gradients via the resistance of the wick and perhaps (for nonzero EMA) the inertia of the fluid. These secondary forces are often large enough to overcome the small 4-8% hysteresis in CAPILs and CAPPMPs, thus causing the device to prime and deprime repeatedly.

Placing more CAPIL and CAPPMP elements in parallel may be required to model 2D effects and realistic pore size distributions.‡

VLO and VHI are important modeling tools for WICK ifaces, especially when they are combined with parallel CAPIL and CAPPMP elements. Setting realistic values of these parameters can have a strong effect on the results.

For example, setting the “flooding limit” VHI (or VLO if VAPOR=A) determines how far a WICK iface will flood before turning itself off and letting fluid pass through a parallel CAPIL or CAPPMP. If there is any backpressure (RBP) associated with a wick, it will disappear when the WICK becomes inactive. VHI (or VLO if VAPOR=A) should be chosen to allow the interface to at least reach VSUB, and perhaps somewhat beyond it.

Setting the “depletion limit” VLO (or VHI if VAPOR=A) determines how far into a wick the interface is allowed to recede before the WICK iface turns itself off and lets vapor pass through a parallel CAPIL or CAPPMP (in other words, before the capillary element deprimes). In general, vapor will escape through a slab of wick that is exceeding its capillary limit while most of the wick still contains liquid: the liquid/vapor interface is highly nonplanar in a subsaturated wick.

---

* True depletion (subsaturation) is anything but one dimensional. Multiple parallel ifaces and paths may be necessary to model a real distribution of pore sizes and uneven movement of the liquid/vapor interface within a porous structure.

† Due to the assumption of perfect mixing, vapor flowing back into the liquid tank often just heats and repressurizes that tank instead of displacing its liquid. Special measures must be made to model vapor injection into a subcooled liquid space.

‡ Regrettably, only one CAPIL or CAPPMP may be associated with a WICK iface, and no more than one WICK iface may be active between the same two tanks. Thus, parallel CAPIL or CAPPMPs may imply a spatial subdivision of the liquid and/or vapor spaces into several tanks.

Caution: "Floating" Wicks: Tank A is the reference by which VLO, VHI, EMA, VSUB, and VZRO are defined. However, if Tank A has nonzero compliance (COMP) or nonzero volumetric rate (VDOT), or if Tank A is connected to any tank other than Tank B via another iface, then the wick and any meniscus associated with it may appear to "float" relative to the volume of Tank B. In other words, the wick equations will be obeyed relative to Tank A but may be violated from the perspective of Tank B: the wick structure will appear to have moved from the viewpoint of Tank B. If the user is modeling a real and fixed wick structure, and if it is not desired to zero COMP and VDOT for Tank A and delete other ifaces, then it may be necessary to switch Tank A with Tank B. (This of course presuming Tank B has zero COMP, VDOT, and no other active ifaces and hence can be used as a reference.) To switch the tanks, the input order on the iface definition must be changed, and the values of VLO, VHI, and almost certainly VSUB and VZRO will need to be changed to reference the new Tank A. Similarly, the designated VAPOR side if the wick should change if Tank A and B are reversed.

If the COMP value has been added in order to simulate the compressibility of liquid (perhaps via the COMPLQ routine), consider instead using a compressible liquid via the 6000 series FPROP DATA option COMPLIQ (Section 3.21.7).

Caution: Directional WICK ifaces and Nondirectional CAPIL Elements: Like a CAPPMP macro, a WICK iface is directional: it has a designated vapor side. However, a CAPIL connector is not directional: it will behave symmetrically independent of the input order. Therefore, if a CAPIL is associated with a WICK iface and if vapor (and hence a meniscus) may appear on either side of the wick represented by this pair of elements, then a second WICK iface should be added between the two tanks to represent the other side of the wick. If one WICK uses Tank A as the vapor side, the other WICK element should use Tank B as the vapor side. Also, the values of VLO, VHI, VZRO, and VSUB will probably be different for each WICK iface since they are defined using different tanks as reference, or at least represent opposite sides of the wick structure being modeled. During the course of the solution, only one of the two parallel WICK elements will be active at a time. If both sides of the wick are dry or if both sides are full of liquid, then both ifaces will be inactive. No intervention is necessary on the part of the user to control which iface is on or off.

3.8.3.5 SPHERE iface

A SPHERE iface is similar to an OFFSET interface, except the pressure difference between tanks (in addition to that caused perhaps by body force terms) is calculated assuming Tank A is a spherical bubble. The pressure relationship for a massless SPHERE iface is therefore $PL_a = PL_b + 2\sigma/r_c$, or:

$$\Delta PL_a - \Delta PL_b = (PL_{b,\,current} - PL_{a,\,current}) + 2\sigma/r_c$$

$$r_c = (0.75 \cdot VOL_a/\pi)^{1/3}$$

Other than VHI, VLO, and EMA, SPHERE ifaces have two descriptors: XIL and XIH. These values describe the upper and lower hysteresis band limits on quality for the iface to be active, and have the same meaning as they do in the WICK iface. They inactivate the iface if the vapor tank is

too wet or if the liquid tank is too dry for a meniscus to be present. If the quality of Tank A ever drops below XIL or the quality of Tank B ever exceeds XIH, the iface is inactivated. To be reactivated the quality of Tank A must exceed XIH and the quality of Tank B must be below XIL.

Like a WICK iface, the working fluid must have a surface tension. Like other ifaces, any mass transfer across the boundary (e.g., dissolution, condensation, etc.) must be represented by paths.

SPHERE ifaces are useful for modeling certain aspects of bubbles. A plurality of bubbles and perhaps distributions of bubbles can be modeled using duplication factors (Section 3.8.5).

EMA (the inertial term) may be used with the SPHERE iface, perhaps to represent the added (virtual) mass effect: although the bubble itself might appear to have little inertia as it grows or collapses, it actually does have "borrowed" inertia because of the liquid it displaces as it changes size.

3.8.3.6 NULL iface

The NULL option assumes no device type: update of the underlying modeling parameters VA, VB, PA, PB, and FPV are left to the user for specialized modeling needs. Because of the sensitivity of model reaction to these values, this option should be considered only by the advanced user.

Note that EMA may be nonzero for a NULL iface, in which case the pressure/volume governing equation (Section 3.8.2.1) is understood to have units of pressure. Furthermore, PA should be unity if EMA is nonzero.

Unlike any other iface, VLO and VHI cannot be used with a NULL iface. Use zero DUPA and DUPB to disable a NULL iface if necessary.

As an example of a NULL iface, consider a hydraulic pressure intensifier (also known as a pressure booster): essentially a piston connecting two cylinders of unequal cross-sectional area. Assuming that the mass of the piston is *Pmass*, that area of side A is *areaA* and the area of side B is *areaB* (and defining for convenience *Aratio*=areaB/areaA), and assuming that the pressure between the piston heads is *Penv*, then in the iface subblock in FLOW DATA:

```
VA = 0.0, VB = 0.0
PA = 1.0, PB = -Aratio
FPV = PL#tankA - Aratio*PL#tankB - Penv*(1.0-Aratio)
DUPA = 1.0, DUPB = Aratio
EMA = Pmass/areaA^2          $ optional mass of undamped piston
```

3.8.4 Interface Input Formats in FLOW DATA

3.8.4.1 Defining Individual ifaces

The format for inputting an individual ifaces is:

```
IF type,ifid,laid,lbid [,DUPA=R] [,DUPB=R]
       [,VLO=R] [,VHI=R] [,EMA=R] [,descriptor=R]
```

where:

`type` ..... FLAT, OFFSET, SPRING, WICK, SPHERE, or NULL
`ifid` ..... interface (iface) id
`laid` ..... id of tank A (in the current submodel only)
`lbid` ..... id of tank B (in the current submodel only*)
`DUPA` ..... tank A duplication factor (number of times tank B seen by tank A)
`DUPB` ..... tank B duplication factor (number of times tank A seen by tank B)
`VLO` ..... lower limit on tank A's volume for the iface to be active (not for NULL)
`VHI` ..... upper limit on tank A's volume for the iface to be active (not for NULL)
`EMA` ..... inertial term, mass divided by cross sectional area squared
`descriptor` . initial value of device-specific descriptors:
FLAT .... none
OFFSET .. DPAB
SPRING .. DPDV, VZRO
WICK .... RCAP, VZRO, VSUB, RBP, XIL, XIH, ICAP, VAPOR
SPHERE .. XIL, XIH
NULL .... VA, VB, PA, PB, FPV

defaults:

`DUPA` 1.0
`DUPB` 1.0
`VLO` 0.0
`VHI` 1.0E30
`EMA` 0.0
`DPAB` 0.0
`DPDV` 0.0
`VZRO` ..... 0.0 for SPRINGs, no default for WICKs
`VSUB` 0.0
`RCAP` ..... 1.0E30 (inherit from ICAP-designated CAPIL/CAPPMP RC if positive)
`RBP` 1.0E30
`XIL` ..... 0.95 (inherit from ICAP-designated CAPIL/CAPPMP XVL)
`XIH` ..... 0.99 (inherit from ICAP-designated CAPIL/CAPPMP XVH)

* If Tank A and Tank B cannot be placed in the same submodel, see the TANK_LINKER utility (Section 7.11.9.3).

ICAP. 0
VAPOR . . . . . .B
VA 0.0
VB 0.0
PA 1.0
PB . . . . . .-1.0
FPV. 0.0

Guidance: References to processor variables within expressions may refer to “#this” as the iface identifier, “#tankA” as the identifier of tank A, and “#tankB” as the identifier of tank B.

Restrictions: VLO must be less than or equal to VHI. XIL must be less than or equal to XIH

Restrictions: WICKs: VSUB must be less than (but not equal to) VZRO. Also, VZRO must be positive and has no default. RCAP must also be positive: zero is not allowed.

Restrictions: If more than one iface is specified between any two tanks, redundant ones will be deactivated by the program, but errors will result if the parallel ifaces conflict with each other. Similarly, redundant ifaces within subvolumes will be deactivated. To turn off a redundant iface, set *both* DUPA and DUPB to zero.

Restrictions: If DUPA is zero, DUPB must be too. Neither should be negative, but noninteger values are acceptable. Errors will result if a complex subvolume is defined in which duplication factors are not used consistently even in ifaces that are automatically inactivated by the subvolume builder.

Restrictions: A subvolume (perhaps consisting only of the two tanks joined by an iface) must contain at least one vapor- or gas-containing tank, or one compliant (COMP>0.0) tank, or one tank held via HLDLMP (Section 7.11.1.4). However, no more than one HLDLMP may be used within a complex subvolume (see Section 3.8.6.2 and Section 3.8.6.3).

Examples:

```
IF FLAT,101,100,1       $ Generates a FLAT iface
IF WICK, 2,3,4          $ A wick IFACE. #3 is Tank A
      RCAP = 1.0E-6, VZRO = 0.5*VolLiq
      VSUB  = 0.5*VolLiq - 5.0E-6
      ICAP  = 103       $ 103: CAPIL or CAPPMP NULL
```

3.8.4.2 Generating ifaces

The format for generating one or more ifaces of the same type is:

```
M GENIF,mid,type,ifid,laidi,lbidi
        [,N=I] [,IFINC=I] [,L1INC=I] [,L2INC=I] [,ICINC=I]
        {iface device descriptions}
```

where:

`mid` ..... unique macro identifier
`type` ..... FLAT, OFFSET, SPRING, WICK, SPHERE, or NULL
`ifidi`..... initial interface (iface) id
`laidi`..... initial id of tank A
`lbidi`..... initial id of tank B
`N`..... number of ifaces to generate
`ICINC`..... increment on ICAP (for wick ifaces only)
`IFINC`..... increment of ifidi (interface id)
`L1INC`..... increment of laidi (tank A id)
`L2INC`..... increment of lbidi (tank B id)

defaults:

`N`..... 1
`ICINC`..... 1
`IFINC`..... 1
`L1INC`..... 1
`L2INC`..... 1

Restrictions: All macro information (including IFINC, ICINC, N, etc.) must precede all iface device specifications within the input block. This means if a wick is being input, ICINC must be in the macro (first) portion of the block, and ICAP in the iface (second) portion.

Examples:

```
M GENIF,11,FLAT,10,20,30,N=9$ Generates 9 FLAT ifaces ‘tween
                        $ tanks 20 to 30, 21 to 31, etc.
M GENIF,32,SPRING,110,100,10
      N     = 8, IFINC = 110, L1INC = 100, L2INC = 10
      DUPA  = 2.0,DUPB  = 1.0
      DPDV = -.103, VZRO = -0.3
      VLO   = 0.03,     VHI = 2.5
```

3.8.5 Interface Duplication Factors

As with paths and ties (Section 3.12), duplication factors may be used with ifaces as well.

For example, consider a model of 20 bubbles of identical size in a liquid control volume. Using tank A to represent the liquid, tank B might be created to represent a single bubble. A SPHERE iface might then be added between them with DUPA=20.0 and DUPB=1.0. Tank A "sees" the effects of 20 Tank B's. The total volume of the control volume, DUPB*VOL_a + DUPA*VOL_b = VOL_a + 20*VOL_b, is conserved.

Duplication factors are also useful for handling pistons with variable area (dual pistons connected by a shaft), or other symmetries such as liquid within the grooves of axially grooved heat pipes, etc.

To turn off an iface (deactivate it), set both DUPA and DUPB to zero.

However, there are several important restrictions involved with iface duplication factors that do not apply to those of paths and ties.

First, both DUPA and DUPB must be zero if either is zero.

Second, while noninteger values are acceptable, negative values are not.

Third, consistency is not enforced for path and tie factors (and indeed inconsistency has occasional modeling usefulness), but iface duplication factors must be consistent. For example, assume there are three tanks interconnected by three ifaces as was shown in the left of Figure 3-18. If the iface from tank 1 to 2 (A and B) had used DUPA=2.0 and DUPB=1.0, then tank 2 would have appeared twice from the viewpoint of tank 1. If the remainder of the duplication factors had defaulted at unity, an inconsistency would have been created: tank 2 appears twice to tank 1, but tank 2 appears once to tank 3 and tank 3 appears once to tank 1. (Making tank 3 also appear twice to tank 1 would have resolved the inconsistency, as would have making tank 2 appear twice to tank 3, or making tank 3 appear 1/2 times to tank 2, etc.) Even though one of the ifaces will be deactivated by the program, since the subvolume will be changed by the choice of which iface to deactivate, the program instead issues an error message and terminates.

3.8.6 Usage Notes

This section describes interactions of ifaces with other program options. The section closes with various warnings and important reminders.

3.8.6.1 COMP and VDOT: Using with ifaces

When a tank is linked into a subvolume via an active iface, it may change volume as needed to satisfy the iface pressure-volume and conservation of volume relationships. Because of this effect, a noncompliant (COMP=0.0) or a tank filled with incompressible liquid assumes a measure of compliance by sharing a subvolume with a compliant or vapor-containing tank: *compliance and compressibility of any tank in a subvolume is "shared" by the other tanks within that subvolume*.

VDOT and COMP may be used with tanks that are contained within subvolumes. However, their effects might seem strange. First, a VDOT applied to any tank within a subvolume affects all tanks within that subvolume equally: the sum of all VDOTs of all tanks (perhaps multiplied by duplication factors) is applied to the entire subvolume as a unit. Therefore, applying VDOT=1.0 to one tank in a subvolume and VDOT=-1.0 to another tank in the same subvolume has no effect (presuming unit duplication factors): they simply cancel each other out. One tank does not grow at the expense of another, since that would violate the constraints place on the tanks by the presence of the iface(s) between them.

Similarly, a COMP value applied to any tank within a subvolume applies as well to all tanks within that subvolume. However, the total compliance of the subvolume will be somewhat less because the definition of the compliance has not changed: compliance is scaled with the volume of the tank to which it is applied (COMP = (dVOL/dPL)/VOL), and not with the volume of the subvolume. To apply a compliance to a subvolume, either the same compliance can be supplied to all tanks, or the compliance of a single tank can be increased by the ratio of the volumes: $V_{subvolume}$/VOL.

For WICK or SPRING ifaces, using nonzero COMP or VDOT on tank A may cause the wick or spring to appear to float (versus appear to be fixed to a reference point). In such cases, tank A and B should be switched (assuming the other tank does not also use nonzero COMP nor VDOT). If the COMP value has been added in order to simulate the compressibility of liquid (perhaps via the COMPLQ routine), consider instead using a compressible liquid via the 6000 series FPROP DATA option COMPLIQ (Section 3.21.7).

3.8.6.2 STEADY Solutions

Since ifaces can only interconnect tanks, and since STEADY treats tanks as junctions, ifaces do not exist (are ignored) during STEADY solutions.

Thus, STEADY solutions may strand a tank that is otherwise connected to the network via ifaces, resulting in aborted runs. More likely, however, is that the network will be solved but that the pressures resulting from the solution will not be good starting points for a subsequent transient (or STDSTL) solution in which the ifaces are again active. Unfortunately, such poor initial conditions can be problematic for ifaces (Section 3.8.6.8). A few calls to CHGLMP (Section 7.11.1.1) may therefore be needed to set the correct initial conditions resulting from a STEADY solution.

One of the purposes of HLDLMP (Section 7.11.1.4) is to be able to use STEADY on a network that would otherwise be illegal if all tanks were treated as junctions. HLDLMP can similarly be used with ifaces present, but if the hold is not released before a transient (using RELLMP), additional limitations apply as described below.

3.8.6.3 HLDLMP: Using with ifaces

HLDLMP (Section 7.11.1.4) is a routine which converts a tank or junction temporarily into a plenum, preserving its thermodynamic state and making it an inexhaustible source or sink of mass and energy. RELLMP releases the hold, returning the tank or junction to its native state.

In STEADY, HLDLMP can be used without restriction since ifaces are not active in that solution routine. However, in STDSTL or transient routines, there are limitations on the use of HLDLMP if ifaces are active. The rest of this subsection pertains to such solutions, and not to STEADY solutions.

No more than one HLDLMP can be placed within a single subvolume. To do so overspecifies the problem mathematically. If more than one HLDLMP is detected within a subvolume, the program will issue a message and abort.

However, an important exception exists to the above constraint: ifaces between held lumps are ignored, and hence two HLDLMPs on adjacent tanks are legal, having the effect of subdividing a subvolume into two subvolumes. Thus, *if a subvolume simply consists of two tanks (and hence one active iface between them), as will often be true, then holding both lumps is legal*.

If only one tank within a subvolume is held via HLDLMP, then it appears to be an inexhaustible supply of not only mass and energy but also *volume*. Its pressure will stay constant, and the rest of the subvolume will adjust itself according to the ifaces it contains. However, the "conservation of volume" relationship is violated since the held tank appears (to other tanks in the subvolume) to grow or shrink arbitrarily as needed to satisfy iface relationships, yet the volume (VOL) of the held tank never changes.

Thus, upon releasing a HLDLMP, the user may need to use the CHGVOL routine (Section 7.11.1.1) to adjust the volume of the previously held tank to a desired value.

For example, consider the situation presented in Figure 3-21. Consider a model of a cylindrical pressurant tank that has been subdivided axially into 10 equal-sized tanks interconnected by massless FLAT iface elements. Mass is extracted from one end (Tank #10) of this subvolume. If a HLDLMP had been issued to Tank 1, then the pressure of the entire vessel would remain constant during the mass extraction, and Tank 10 would quickly be depleted. Tanks 2 through 9 would grow in volume as tank #10 shrinks, and the system would behave as if Tank 1 were growing as well, though it isn't because of the HLDLMP action.

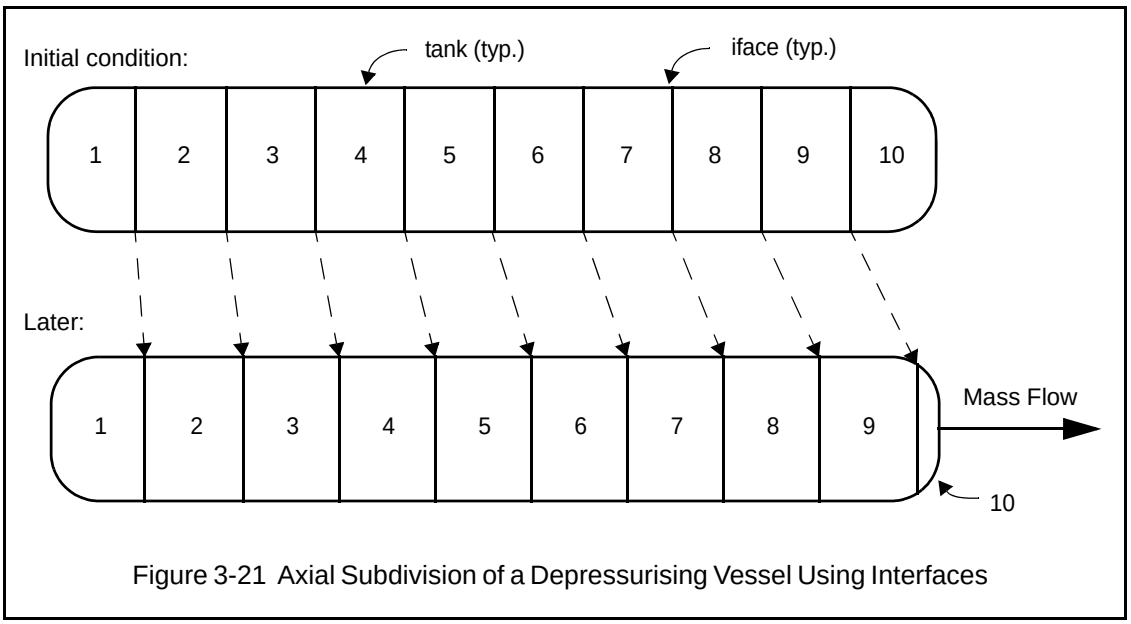


Figure 3-21 Axial Subdivision of a Depressurising Vessel Using Interfaces

If a HLDLMP were placed instead on Tank 5 in the middle of the vessel, the effects would have been similar to those described above. But if an additional HLDLMP had been placed on Tank 6, then Tanks 1 through 5 would have formed one (unchanging) subvolume, and Tanks 6 through 10 would have formed a second subvolume that behaves in the manner described previously: Tanks 7, 8 and 9 growing at constant pressure while Tank 10 shrinks.

If instead a HLDLMP had been placed on Tank 10, then no changes would have occurred throughout the entire subvolume.

3.8.6.4 Using Twinned Tanks with ifaces

Generation of a twinned tank (Section 3.25) automatically generates a FLAT iface between the two tanks. While a FLAT iface must be used with a duct macro (Section 3.9.2), a different type of iface may be specified between an isolated pair of twinned tanks, which overrides the default of FLAT. Note that twinned tanks may be joined with other subvolumes via additional ifaces to other (perhaps twinned tank), but that the above subvolume rules (Section 3.8.1) still apply. Therefore, one need only attach an iface to one of a pair of twinned tanks, and ifaces themselves need not (and in fact cannot) be twinned.

3.8.6.5 Body Forces

As with paths, body force terms (Section 3.13) are superimposed upon the ifaces depending on the values of the acceleration vector (ACCELX, ACCELY, ACCELZ), the coordinate location of the tanks (CX, CY, CZ), and their current densities (DL_a, DL_b). By default, the acceleration (usually gravity) and the coordinates are zero, so body forces are neglected. When they are nonzero, an *average* density is used to compute the pressure difference between the two tanks. There are no phase "suction" options for ifaces since interfaces do not transfer mass, and since vapor/liquid interfaces are usually modeled by an iface and therefore do not occur *within* the tanks in most modeling applications.

However, this treatment means that if paths that use phase suction (including capillary elements and twinned tubes or STUBE connectors) are placed in parallel with ifaces, then a conflict of assumptions will occur and one of the tanks will likely shrink continuously (or at least as long as the two phases persist).

This effect can be useful in modeling: it can be used to position the iface at the liquid/vapor boundary. For example, consider a liquid-fill tank with a two-phase tank above it in a gravity field. Assume a FLAT, WICK, or SPHERE iface had been placed in parallel with a CAPIL connector* (which uses liquid-suction STAT=DLS internally) between these two tanks. Because the connector will only "see" liquid in the tank above, it will feel a greater body force than will the iface in parallel. This will cause liquid to flow down from the upper tank into the lower tank, and the iface will cause the lower tank to grow while the upper tank shrinks (in effect, the control volume boundary moves

* A CAPIL was used in this example because few other connectors can tolerate the extremes of 100% vapor on one side and 100% liquid on the other, with zero or very small or reversing flows.

up through the liquid towards the liquid/vapor interface). This action stops when the upper tank is depleted of liquid, and the CAPIL primes. At this point, the iface separates the liquid from the vapor in the subvolume formed by the two tanks.

3.8.6.6 Logical References

All iface descriptors (e.g., DUPA, DPDV, VSUB, VHI, EMA, etc.) except ICAP and VAPOR can be addressed in logic blocks for inspection or modification, as can the underlying parameters VA, VB, PA, PB, and FPV. Register-containing formulas may also be used to define these parameters, which can then be adjusted using the dynamic register features (Section 4.9).

However, as with connectors, the modeling parameters (descriptors, including duplication factors) should only be modified in FLOGIC 0 blocks (although OPERATIONS, PROCEDURE, OUTPUT CALLS and a few other blocks may be more appropriate for certain analyses), and the underlying parameters (VA, VB, etc.) should be inspected or modified in FLOGIC 1 blocks by advanced users.

3.8.6.7 Output Options

The IFCTAB tabulation-style output routine can be used to tabulate ifaces and their status, including whether or not they are active, the pressure difference between the lumps, the current subvolume to which they belong if active, and the sum of the tank volumes. For example:

```
CALL IFCTAB('ALL')
```

The subvolume number is arbitrarily assigned by the program starting with 1 and incrementing by 1. Since subvolume assignments will change as ifaces are activated or deactivated, this assignment may change from time step to time step. However, the user can verify the subvolumes created by the presence of ifaces by comparing these numbers: all the tanks that list subvolume “2,” for example, will share a common conservation of volume relationship.

The tabulation also shows “A+B Volume” or the sum of volumes of each tank. However, because of duplication factors this volume may appear strange. Actually, the tabulated result is VOL(A)*DUPB + VOL(B)*DUPA, such that the total subvolume volume appears constant or at least can be compared to previous or subsequent tabulations.

However, if the subvolume contains more than two tanks, then this “A+B Volume” number is nearly useless since it will change over time, and the sum of all these values will exceed the total subvolume volume since tanks will be listed more than once.

Similar information is printed for tanks by LMPMAP and FLOMAP.

3.8.6.8 Miscellaneous Cautions

This subsection lists various problems that might be encountered with the use of ifaces.

Directionality: Unlike paths, the user should be cognizant of the fact that ifaces exhibit directionality: the behavior of most devices (FLAT being an exception) is dependent on the order in which the tanks are specified in the definition of the iface. The first tank identified is the reference tank and is referred to as "A" whereas the second tank is referred to as "B." The meaning of the input parameters (e.g., VHI, VLO, EMA, DPDV, VZRO, DPAB, etc.) will change if this order is reversed.

Diminishing Volumes: Interfaces, when placed in many models, will often cause some tanks to grow and others to shrink monotonically. For example, consider a model of a cylindrical pressurant tank that has been subdivided axially into 10 equal-sized tanks interconnected by FLAT iface elements (Figure 3-21 again). If mass is extracted from one end of this subvolume, the tank from which the mass is extracted will rapidly shrink and the other 9 tanks will gradually expand (having constant mass). In other words, all the mass exhausted from the vessel is coming from a single tank, which initially held only 10% of the mass of the fluid in the vessel.

A diminishingly small tank will require smaller and smaller time steps, and eventually if no action is taken, either the mass or the volume of the tank will become negative and the run will abort.

It is the user's responsibility to take appropriate action when a tank shrinks below acceptable limits. This "action" may entail deactivating an iface (setting DUPA=DUPB=0.0), applying a lower limit to a tank volume using VLO for Tank A (and VHI for Tank B) such that the iface deactivates automatically, or terminating the run and investigating modeling changes.

Network Construction Requirements (Inactive Paths): Interfaces do not eliminate the need for each lump to have at least one path connected to it. If no path is truly desired or needed, such as might be the case in the analysis example presented above in Figure 3-21, then the user might generate MFRSET connectors between the tanks whose initial flow rate and SMFR value are set to zero. Such paths satisfy the network construction rules, but are essentially inactive during the analysis. Similarly, connectors with DUPI=DUPJ=0.0 may be added but will have no effect on the solution.

Poor Initial Conditions: Interfaces can be sensitive to extremely poor initial conditions. This includes the use of STEADY-generated results to start a transient routine without having reinitialized the tanks properly.

Interfaces connect two time-dependent elements together. However, like connectors, ifaces themselves are time-independent. This can cause rapid changes on the adjacent tanks in a manner that is beyond the time step controller's ability to regulate. For example, if two tanks are interconnected by a FLAT iface, and begin a transient with a large pressure difference between them, then the iface will attempt to eliminate that pressure difference *in one time step, regardless of the size of that time step*. This can cause severe changes to the model.

Therefore, the user should attempt to assure that the initial conditions of the model (especially tank pressures) or the results of a STEADY run (Section 3.8.6.2) are reasonably consistent with the iface specifications before beginning a transient (or STDSTL).

Drastic Changes: For most of the reasons stated above, drastic changes to an iface's descriptors may not be well tolerated by some models. Changes should be gradual. Similarly, changes to a lump's pressure via calls to CHGLMP should not create an artificial pressure gradient: both tanks (on each side of the iface) should be changed if one is changed, perhaps using the CHALLP routine if appropriate.

Imperfect (perhaps nonzero) Pressure Differences: Volumes are tracked quite accurately. However, if any tank within a control volume contains a void (vapor or gas), then pressures are solved indirectly instead of directly, and slight errors may be evident due to numerical truncations and other approximations.

In other words, the pressure difference across a FLAT iface may be nonzero, although it should be acceptably small for most modeling purposes: typically no more than about 0.01 psid or about 70 Pa. This error will tend towards zero because of the presence of the "$PL_{b,current} - PL_{a,current}$" term in the iface pressure-volume equations presented above, and should only be noticeable when rapid changes are occurring in the model.

However, problems may occur when low-resistance connectors or capillary components (CAPIL connectors or CAPPMP macros) are placed in parallel with an iface (including flooded WICK ifaces, FLAT ifaces, OFFSET ifaces with DPAB=0, and SPRING ifaces with DPDV=0). Such paths can be very sensitive to small changes in pressure differential. It may be necessary to increase the resistance of such connectors, or consider other measures with capillary elements to prevent artificial flooding. Such measures might include setting RBP on a WICK iface to offset the pressure errors, adding a small gravity gradient to accomplish the same end, or perhaps setting CFC of the CAPIL or CAPPMP temporarily to zero if the liquid tank pressure exceeds the pressure of the vapor tank.

WAVLIM: The WAVLIM routine is currently unable to account for the compliance associated with a tank's presence in a subvolume. It should therefore not be used when ifaces are active.

Time Step Limits: Normally, ifaces do not require any smaller time steps than those required for any tank with variable volume. However, ifaces can require additional transition limits as the volume of Tank A passes near to VLO or VHI. Similarly, WICK ifaces can require additional transition limits as the volume of Tank A passes near to VSUB or VZRO. At these points, the behavior of the iface changes and therefore the time step controller attempts to avoid stepping past these transitions in order to maintain accuracy. If the difference between VZRO and VSUB is very small, either small time steps or jittery pressure differentials can result.

Dynamic Registers: The volume (VOL) of a tank is normally an input variable. However, VOL becomes an output variable if the tank is attached to an active iface. This means that if the volume of the tank is defined via a formula (i.e., an expression containing either registers or processor variables), updates to that formula or its value during a solution (STEADY, FORWRD, or TRANSIENT) may conflict with SINDA/FLUINT calculations. A warning will be produced if such a problem is detected. See Section 4.9.3.

3.9 Macros: Element Generation and Component Models

By now, the reader will have noticed the very large number of input parameters and options in FLUINT. One feature useful for reducing inputs is the ability to explicitly define or redefine default values at any point in the input stream. Also, macrocommands (or *macros*) are available that generate many elements with a single command, thereby decreasing the amount of input and increasing the readability of the input file.

Actually, such *element generation* or GEN macros are but one of three types of FLUINT macros. Another type, called a *component model*, generates a fixed number of network elements (lumps, paths, ties, fties, ifaces) that represent a specific device. Currently, there is only one such device: the capillary evaporator pump or CAPPMP. While GEN macros create an indefinite number of network elements that are otherwise unrelated, the CAPPMP macro generates a fixed number of specific elements in a set arrangement that are linked together to simulate an evaporator pump.

A third type of macro, the *duct macro*, is somewhat intermediate between the above two extremes. They generate an indefinite number of duct sections, where each duct section is composed of lumps, paths (tubes or STUBE connectors), and perhaps ties, fties, and ifaces. Unlike element generation macros, there is a defined sequence and relationship between generated elements: they are linked in series and are specifically intended to represent a continuous flow passage.

Duct macros are very important modeling tools in SINDA/FLUINT. They able to take into account axial gradients in flow area, density, and side flow passages (Section 3.15). They represent a continuous 1D momentum relationship along a series of lumps and paths with continuous (if not constant) flow area. Sinaps and FloCAD Pipes represent a duct macro, but those programs are able to create more complex duct macros that include multiple Pipes, allow REDUCER and EXPANDER connectors to join a duct macro, allow twinned and homogeneous paths to be mixed within one duct macro, allow tubes and STUBEs to be mixed, and allow tanks and junctions to be mixed (Section 3.9.2.2).

The remainder of this section describes the input and use of macrocommands, which are summarized below.

Element generation commands:

GENLU. . . . . Generate several lumps of the same type
GENPA . . . . . Generates several paths of the same type
GENT . . . . . Generates several ties of the same type
GENFT . . . . . Generates several fties of the same type
GENIF . . . . . Generates several ifaces of the same type

Component models:

CAPPMP . . . . Generates a model of a generic capillary pump

Duct macros:

LINE . . . . . Generates a string of lumps and paths representing an adiabatic duct (heat transfer can be added separately)
HX . . . . . Generates a string of lumps, paths, and ties representing a diabatic duct (heat exchanger segment)

(extended) . . . . See Section 3.9.2.2

Restriction: For all of the following element generation and duct macros, *in order to use increments the starting value must be explicitly stated within the macro subblock* (i.e. starting values cannot be set in a DEF subblock).

When referencing processor variables within expressions in macrocommands, ***the user is reminded of the existence of indirect operators and increment operators,*** described in Section 2.8.8.

Also, references to dissolved gas modeling options, described in Section 3.23, are available in these macrocommands but those descriptions will not be repeated here.

3.9.1 Element Generation Macros

Five GEN macros are available, corresponding to the five FLUINT network elements:

GENLU . . . . . Generates several lumps of the same type, or several pairs of twinned tanks
GENPA . . . . . Generates several paths of the same type, or several pairs of twinned paths of the same type
GENT. . . . . Generates several ties of the same type, or several pairs of twinned ties of the same type. Can be used to add another row of ties to a LINE or HX macro, even if centered
GENFT . . . . . Generates several fties of the same type
GENIF . . . . . Generates several ifaces of the same type

The FLOW DATA formats for these GEN macros are as follows.

Lump generator format:

```
M GENLU,mid,type,ilid[,N=I][,LUINC=I][,LTWIN=I]
        [,TLINC=R][,PLINC=R][,XLINC=R or ALINC=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 ...
```

where:

mid ..... macro identifier
type ..... TANK, JUNC, or PLEN
ilid ..... ID of first lump
N ..... number of lumps to generate (minimum of one). The value of N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
LUINC..... lump ID increment
TLINC..... lump TL increment
PLINC..... lump PL increment
XLINC..... lump XL increment
ALINC..... lump AL increment
VOLINC .... tank VOL increment (ignored if not tank)
QDINC..... lump QL increment (ignored if plenum)
VDINC..... tank VDOT increment (ignored if not tank)
CXINC..... lump CX increment
CYINC..... lump CY increment
CZINC..... lump CZ increment
LTWIN..... unique identifier of initial secondary twin tank to be generated (for non-equilibrium modeling), where ilid is the corresponding primary tank
LTINC..... LTWIN increment for twinned tanks (Section 3.25)
IFINC..... IDFACE iface increment for twinned tanks (Section 3.25)
IDPINC .... IDPATH superpath increment for twinned tanks (Section 3.25)
IDFTNC .... IDFTIE CONSTQ ftie increment for twinned tanks (Section 3.25)
specs..... remainder of lump specifications (treated as initial values for the increments) as described in previous LU TANK, LU JUNC, and LU PLEN sections. See also Section 3.25 for twinned tank options and Section 3.23 for gas dissolution/evolution options.

defaults (if not previously defined in an LU DEF subblock):

N..... 1
LUINC..... 1
TLINC..... 0.0
PLINC..... 0.0
XLINC..... 0.0
ALINC..... 0.0
VOLINC 0.0
QDINC..... 0.0
VDINC..... 0.0

CXINC 0.0
CYINC 0.0
CZINC 0.0
LTWIN .....no twinned tanks (single homogeneous lumps created)
LTINC .....LUINC
IFINC .....LUINC
IDPINC.....LUINC
IDFTNC.....LUINC
specs .....see LU TANK, LU JUNC, and LU PLEN defaults

Guidance: Values not applicable to the desired lump type are ignored (e.g., VDOT and VDINC for junctions). LU DEF subblocks can be used prior to a GENLU call if desired.

Example:

```
M GENLU,213,TANK,1,N=10, VOL=1.0, XL=0.0, TL=500.0,
       TLINC=10.0,PL=14.7, PLINC=0.1, COMP = 0.01/14.7
```

Path generator format:

```
M GENPA,mid,type,ipid,il1id,il2id[,N=I][,PAINC=I]
       [,L1INC=I][,L2INC=I][,FRINC=R]
       further/initial path specifications ...
```

where:

mid.....macro identifier
type.....TUBE or CONN
ipid.....ID of first path
il1id .....ID of first upstream lump
il2id .....ID of first downstream lump
N.....number of paths to create (minimum of one). The value of N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
PAINC .....path ID increment (also applies to tid if pairs of twins are generated)
L1INC .....upstream lump ID increment
L2INC .....downstream lump ID increment
FRINC .....increment of initial flow rate
specs .....remainder of TUBE or CONN specifications (treated as initial values for the increments) as described in previous PA CONN and PA TUBE sections. See also Section 3.23 for gas dissolution/evolution options. For connectors, all device specific inputs must follow the DEV=C in the subblock.

defaults (if not previously defined in a PA DEF subblock):

N 1
PAINC 1
L1INC 1
L2INC 1
FRINC 0.0
STAT . . . . . NORM
TWIN . . . . . no twins (single homogeneous paths generated)
DUPI . . . . . 1.0 (or value of DUP)
DUPJ . . . . . 1.0 (or value of DUP)
DUP 1.0
specs . . . . . see PA CONN and PA TUBE defaults

Guidance: Path specifications can vary greatly, especially for connector devices. The GENPA macro simply generates several of the same type path. As such, PA DEF subblocks can be used in conjunction with these macros as if the user had individually specified each path.

Guidance: The use of duplication factors with this macro is legal but should be used carefully. Duplication factors usually replace the need for this macro and vice versa.

Restriction: All device specific inputs for connectors must follow the DEV=C, as with individual connector subblocks, and these inputs must occur at the end of the subblock.

Restriction: The default MCH for NULL, MFRSET, and VFRSET connectors cannot be set in a PA DEF subblock.

Example:

```
M GENPA,11,CONN,10,10,20, N=9, PAINC=1, L1INC=0,
        L2INC=0, FR=1.0, DEV=STUBE, DH=0.01,TLEN=2.0
        AF=1.0
```

Tie generator format:

The generic format for generating ties of all types is:

```
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 ...
```

where:

`mid`.....macro identifier
`type`.....HTN, HTNC, HTP, HTU, HTNS, HTUS, or HXC
`itid`.....ID of first tie
`ilpid` .....ID of first lump (HTN, HTNC, HTU, or HXC), or ID of the first segment path (HTNS, HTUS)
`inid`.....ID of first node (smn.nid format)
`fs` .....fraction of the segment paths' wetted area (HTNS only)
`ipid`.....ID of first path (HTN, HTNC, and HXC only). Alternatively, the path id of the primary path of a first pair of twins (the involvement of the secondary path is implied).
`f1` .....fraction of #1 paths' wetted area (HTN, HTNC, and HXC only). For the HXC option, this is the area fraction factor applied to *all* generated ties, similar to the AFRACT keyword in an HX macro.
`ip2id` .....ID of first #2 path (HTNC only). Alternatively, the path id of the primary path of a second pair of twins (the involvement of the secondary path is implied).
`f2` .....fraction of #2 paths' wetted area (HTNC only)
`N`.....number of ties to generate (minimum of one). The value of N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
`LUINC` .....lump ID increment (HTN, HTNC, HTU, and HXC only)
`PAINC` .....path ID increment (HTN, HTNC, HTNS, and HXC only, #1 for HTNC)
`PA2INC`.....#2 path ID increment (HTNC only)
`NINC`.....Node ID increment (in same thermal submodel)
`TIINC` .....tie ID increment
`TTINC` .....twinned tie ID increment
`specs` .....remainder of HTN, HTU, HNTC, HTUS, or HTNS specifications (treated as initial values for the increments) as described in Section 3.6.

defaults:

`N`.....1
`LUINC` 1
`PAINC` 1
`PA2INC`.....1
`NINC`.....1
`TIINC` 1
`TTINC` 1

`fs`..... 1.0 for HTNS ties
`f1`..... 1.0 for HTN and HXC ties, required for HTNC ties
`f2`..... 0.5
`specs`..... see T HTN, T HTP, T HTU, T HNTC, T HTUS, and T HTNS defaults (Section 3.6)

Guidance: Inputs which may be required, depending on the type selected, appear between {} characters. Note that this macro generates ties between one fluid submodel and one thermal submodel only.

Guidance: The HTN, HTNC, HTNS, HTP, HTUS, and HTU options generate ties of those types. "HXC" does not correspond to a tie type. The GENT,HXC option generates an additional axial "row" of HTNC ties for a centered duct macro. (Up or downstream ducts can use the GENT,HTN option for additional axial rows.) This option differs from the GENT,HTNC option in that it automatically accounts for the area fraction distributions appropriate to centered duct macros, as shown in the example below. Thus, even though the GENT,HXC option generates HTNC ties, its input requirements are more similar to the GENT,HTN option.

Restrictions: *Neither ilpid, ipid, ip2id, nor any increments can reference a secondary twin, or a path other than a tube or STUBE. (ilpid can reference any type of path for HTUS ties other than secondary twins.)*

Because most inputs vary by type of tie to be generated, the following type-dependent formats may be easier to use (where the use of ilpid has been replaced by ilid or ipid depending on whether the required input is a path or lump ID number):

```
M GENT,mid,HTU,itid,ilid,inid, UA = R
        [,N=I][,LUINC=I][,NINC=I]
        [,TIINC=I][,TTINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HTUS,itid,ipid,inid, UA = R
        [,N=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,TTINC=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][,TTINC=I][,DUPN=R][,DUPL=R]

M GENT,mid,HTNS,itid,ipid,inid [,fs]
        [,N=I][,PAINC=I][,NINC=I]
        [,TIINC=I][,TTINC=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][,TTINC=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][,TTINC=I][,DUPN=R][,DUPL=R]
```

Examples:

```
M GENT,100,HTN,10,10, BATT.10,9,1.0
       N=10, TIINC=10, NINC=0
M GENT,120,HTNS,20,20, BATT.192, N=13, STAY
M GENT,101,HTNC,300,300, BATT.10, 299,1.0,301
M GENT,102,HXC,1,1,  WALL.1, 1, N=5
```

where the last example (HXC option) is equivalent to:

```
T HTNC, 1, 1, WALL.1,1,1.0,2,0.5
T HTNC, 2, 2, WALL.2,2,0.5,3,0.5
T HTNC, 3, 3, WALL.3,3,0.5,4,0.5
T HTNC, 4, 4, WALL.4,4,0.5,5,0.5
T HTNC, 5, 5, WALL.5,5,0.5,6,1.0
```

Ftie generator format:

```
M GENFT,mid,type,iftid,il1id,il2id[,N=I][,FTINC=I]
       [,L1INC=I][,L2INC=I][,TUBINC=I]
       further/initial ftie specifications ...
```

where:

`mid` ..... macro identifier
`type` ..... USER, CONSTQ, or AXIAL
`iftid` ..... ID of first ftie
`il1id` ..... ID of the first c^{th} lump
`il2id` ..... ID of the first d^{th} lump
`N` ..... number of fties to generate (minimum of one). The value of N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
`FTINC` ..... ftie ID increment
`L1INC` ..... c^{th} lump ID increment
`L2INC` ..... d^{th} lump ID increment
`TUBINC` .... tube or STUBE connector ID increment (AXIAL only)
`specs` ..... remainder of ftie specifications (treated as initial values for the increments) as described in Section 3.7

defaults:

`N` 1
`FTINC` 1
`L1INC` 1
`L2INC` 1
`TUBINC` 1
`specs` ..... see Section 3.7 for defaults

Guidance: Values not applicable to the desired ftie type are ignored.

Example:

```
M GENFT,101,AXIAL,10,10,11,N=10
```

Iface generator format:

```
M GENIF,mid,type,ifid,il1id,il2id[,N=I][,IFINC=I]
        [,L1INC=I][,L2INC=I][,ICINC=I]
        further/initial iface specifications ...
```

where:

`mid`.....macro identifier
`type`.....FLAT, OFFSET, WICK, SPHERE, SPRING, NULL
`ifid`.....ID of first iface
`il1id` .....ID of the first ath lump (reference tank)
`il2id` .....ID of the first bth tank
`N`.....number of ifaces to generate (minimum of one). The value of N can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
`IFINC` .....iface ID increment
`L1INC` .....ath tank ID increment
`L2INC` .....bth tank ID increment
`ICINC` .....associated CAPIL or CAPPMP connector ID increment (WICK only)
`specs` .....remainder of iface specifications (treated as initial values for the increments) as described in Section 3.8

defaults:

`N`.....1
`IFINC` 1
`L1INC` 1
`L2INC` 1
`ICINC` 1
`specs` .....see Section 3.8 for defaults

Guidance: Values not applicable to the desired iface type are ignored.

Example:

```
M GENIF,101,FLAT,10,10,11,N=10
```

3.9.2 Duct Macrocommands

Duct macros are very important modeling tools, and are common to most flow models.* Although Sinaps and FloCAD can create more complicated duct macros (Section 3.9.2.2), within a SINDA/FLUINT text input file, there are two types: LINE and HX.† The LINE macro allows fast generation of pipe or line models without heat transfer ties. The HX macro performs the same function as LINE macros, but automatically adds a row of radial heat transfer ties to thermal submodels for modeling heat exchangers and other diabatic lines. (The HX macro is equivalent to a LINE macro plus a GENT,HXC macro: it is simply a LINE that has built-in heat transfer options. Despite the name, the HX macro is *not* the primary way of modeling a heat exchanger. See Section 7.11.10 and Section 7.11.11 instead.)

At first, the LINE and HX macros may seem more like element generation commands rather than component models. However, there are important differences between lumps and paths generated or input separately and those generated in order to represent a duct with internal variations in properties due to heat transfer or side flow passages. The difference is that FLUINT assigns no higher level meaning to a simple, perhaps chance catenation of lumps and paths, whereas a duct macro is known to represent a continuous flow passage. Additional terms‡ and methods are applied to duct macros that are not applied to elements that are input separately. Duct macros use a default system that eliminates many of the complex options available with nonequilibrium (Section 3.25) and dissolution/evolution (Section 3.23) modeling.** Therefore, ***the analyst should preferentially use these duct macros whenever modeling an axially subdivided duct or other continuous flow passage.***

* FloCAD and Sinaps users should pay particular attention to duct macros, since both FloCAD and Sinaps use duct macros to represent the fluidic portions of “pipes,” and since both automatically aggregate all appropriate lumps and paths in series into duct macros, which causes differences in results for compressible flows, flows with side branches, etc. (see Section 3.15.3, Section 3.9.2.2)

† Despite the name, an HX macro is rarely the same as a “model of a heat exchanger.” Refer to Section 7.11.10 and Section 7.11.11 for more details on heat exchanger design and simulation.

‡ Use of duct macros invoke logic that automatically calculates AC and FG factors. These factors take into account axial variations in velocity (Section 3.15.3) and the momentum transfer associated with phase change (if paths are twinned, Section 3.4.1.1). Causes for velocity gradients include density changes, area changes, side flow passages, and transient storage or release of mass in compressible tanks. Thus, these models will respond differently than if the elements had been input individually—FLUINT treats them like continuous ducts rather than as isolated elements.

** Other than listing increment keywords, those options are not described in this section. However, note that when using macros to generate twinned tanks, FLAT ifaces are the only type allowed between each twinned pair.

The LINE macro has three options, corresponding to the discretization schemes depicted in Figure 3-22:

U Option . . . . . Generates pairs of lumps and paths in series: lump —> path —> lump —> path... that model a line or pipe. Lumps are upstream of the corresponding path. Paths are tubes or STUBE connectors. Lumps are tanks, junctions, or plena. No ties are generated. Suitable for gas purge modeling (Section 3.27).

D Option . . . . . Generates pairs of paths and lumps in series: path —> lump —> path —> lump ... (Same as U option but lumps are at downstream end of each section.) Suitable for modeling filling of lines with liquid (Section 3.27).

C Option . . . . . Generates a series of paths and lumps that begin and terminate with paths: path —> lump...(more path/lump pairs)... —> path. (Lumps are centered between corresponding paths.)

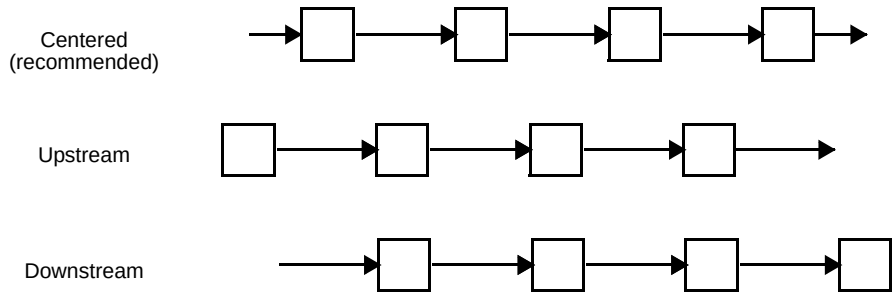


Figure 3-22 Duct Macro Options (Four Sections Each)

The HX macro has four options, corresponding not only to the three discretization options available in the LINE macro, but also to the optional inclusion of segment-oriented ties in the downstream discretization option:

U Option . . . . . Same as U option with LINE but generates HTN ties automatically. Allows fast building of heat exchanger sections.

D Option . . . . . Same as D option with LINE but generates HTN ties automatically. Allows fast building of heat exchanger sections.

DS Option . . . . Same as D option with LINE but generates HTNS ties automatically. Allows fast building of heat exchanger segments.

C Option . . . . . Same as C option with LINE but generates HTNC ties automatically. Allows fast building of heat exchanger sections.

The differences between the U, C, and D options may seem confusing at first. Because centered (C) options start and end with paths, they may be treated as a single “superpath” and may be easier to apply. For this reason, and because the centered option is recommended for two-phase flows, ***use the centered (C) option if there is any doubt.***

To change the dimensions of an entire macro during program execution, use register-based expressions. In the special case of a conical duct, consider the MACONE routine (Section 7.2).

Note that the PPSAVE option will not expand FLUINT macrocommands into the individual elements because information is lost when elements are input separately. However, the preprocessor output file will contain the data generated by the macros for verification purposes.

Caution in Modeling Manifolds using Duct Macros: The duct macros are intrinsically 1D elements, such that when side flow passages are attached the velocities of those passages are assumed to be perpendicular to the macro vector and therefore do not inject or extract momentum, only mass. This can cause inaccuracies in models of manifolds. If the manifold pressure drop is negligible, use a single lump (i.e., assume a perfect manifold) and consider also duplication factors (Section 3.12) on the side flow passages. If a precise model of the manifold is required, consider using the wye/tee simulation routines (YTKALI, YTKALG, YTKONV as listed in Section 7.2) instead of duct macros. See also Section 3.15 for a more complete discussion of manifolds.

The FLOW DATA formats for the LINE and HX macros are as follows. For brevity, lump, path and tie inputs will not be repeated here *unless they differ in meaning from those inputs in individual element subblocks, or if they do not apply equally to all generated elements.* For individual lump, path, and tie inputs, refer to Sections 3.3 through 3.6.

LINE (Untied Duct) Macro Format:

```
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 or AL=R][,QL=R]
        [,LUINC=I][,TLINC=R][,PLINC=R]
        [,XLINC=R or ALINC=R][,QDINC=R]
        [,PAINC=I][,TWIN=I]
        [,CX=R][,CXINC=R][,CY=R][,CYINC=R][,CZ=R][,CZINC=R]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=R]
        [,DUPI=R][,DUPJ=R][,DUP=R][,LTWIN=I]
        [,LTINC=I][,IFINC=I][,IDPINC=I][,IDFTNC=I]
        further lump and path tie specifications ...
```

where:

- `mid` ..... macro ID
- `opt` ..... U, D, or C (Up, Down, or Center)
- `ilid` ..... ID of first lump
- `ipid` ..... ID of first path
- `l1id` ..... ID of lump to connect to (not generated,) at downstream end for opt = U, else at upstream end
- `l2id` ..... ID of 2nd lump to connect to (not generated,) for opt = C only, at downstream end

NSEG. . . . . Number of segments (lumps) to generate (minimum of one, NSEG+1 paths will be generated if opt = C, twice that number if twinned). The value of NSEG can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.

STAT. . . . . phase suction status flag:
NORM: . . .normal (extracts both phases from upstream lump)
LRV . . . . . .Liquid suction from upstream, vapor suction from downstream
VRL. . . . . .Vapor suction from upstream, liquid suction from downstream
These options will be applied to all paths. OTHER STAT VALUES ARE ILLEGAL FOR DUCT MACROS.

TLENT . . . . . total length of line to be generated

DHS. . . . . diameter of line (DH of each path. DEFF may be used in addition.)

AFS. . . . . flow area of line (AF of each path)

AFI/AFJ. . . . flow area at the defined inlet and outlet of *the entire macro.* If either AFI or AFJ is used, they both must be used.

LU . . . . . lump type: TANK, JUNC, or PLEN

PA . . . . . path type: TUBE or STUBE (connector)

LUINC . . . . . lump ID increment

TL . . . . . init. temp. of first lump

PL . . . . . init. pressure of first lump, PL! means PL has priority over TL

XL . . . . . init. quality of first lump

AL . . . . . init. void fraction of first lump

QL . . . . . init. heat rate of first lump

TLINC . . . . . lump TL increment

PLINC . . . . . lump PL increment

XLINC . . . . . lump XL increment

ALINC . . . . . lump AL increment

QDINC . . . . . lump QL increment (*not* “QLINC” for historical reasons)

PAINC . . . . . path ID increment (also applies to twin path itid)

CX . . . . . location on the X axis

CXINC . . . . . lump CX increment

CY . . . . . location on the Y axis

CYINC . . . . . lump CY increment

CZ . . . . . location on the Z axis

CZINC . . . . . lump CZ increment

CXI etc . . . . . upstream port coordinates (applies only to upstream end of first path, ignored if opt = U)

CXJ etc . . . . . downstream port coordinates (applies only to downstream end of last path, ignored if opt = D)

TWIN. . . . . unique identifier of initial secondary twin path to be generated (for slip flow modeling), where ipid is the corresponding primary twin

DUPI ..... total upstream duplication factor (applies only to upstream end of first path, ignored if opt = U)
DUPJ ..... total downstream duplication factor (applies only to downstream end of last path, ignored if opt = D)
DUP ..... Sets both DUPI and DUPJ (one of which will be ignored unless opt = C)
LTWIN..... unique identifier of initial secondary twin tank to be generated (for non-equilibrium modeling), where ilid is the corresponding primary tank
LTINC..... LTWIN increment for twinned tanks (Section 3.25)
IFINC..... IDFACE iface increment for twinned tanks (Section 3.25)
IDPINC .... IDPATH superpath increment for twinned tanks (Section 3.25)
IDFTNC .... IDFTIE CONSTQ ftie increment for twinned tanks (Section 3.25)

defaults (if not previously defined in a PA DEF or LU DEF subblock):

NSEG 1
DHS ..... none, except calculated for each individual path automatically if DHS is not input but AFI and AFJ are input
AFS ..... negative (from DHS assuming circular x-section), or calculated as the average flow area for each path if AFI and AFJ are input
AFI/AFJ ... -1.0 (not used)
LU..... TANK
PA..... TUBE
LUINC..... 1
TLINC..... 0.0
PLINC..... 0.0
XLINC..... 0.0
ALINC..... 0.0
QDINC..... 0.0
PAINC..... 1
CX..... 0.0
CXINC..... 0.0
CY..... 0.0
CYINC..... 0.0
CZ..... 0.0
CZINC..... 0.0
CXI etc..... no upstream port defined
CXJ etc..... no downstream port defined
TWIN ..... no twins (single homogeneous paths created)
DUPI ..... 1.0 (or DUP)
DUPJ ..... 1.0 (or DUP)
DUP 1.0

LTWIN ..... no twinned tanks (single homogeneous lumps created)
LTINC ..... LUINC
IFINC ..... LUINC
IDPINC ..... LUINC
IDFTNC ..... LUINC

Guidance: A line is a series of paths and lumps that simulates a pipe, duct, or long passage. Lines can be composed of tanks, junctions, or plena, and tubes or STUBE connectors. If tanks are used, VDOT terms are assumed to be zero initially, and volumes are automatically calculated according to segment size and other options. If plena are used (perhaps for boundary conditions for a thermal model) QL and QDINC are ignored. PA DEF and LU DEF options apply to most inputs and should be used for clarity unless increment options are used.

Guidance: The configuration of each segment depends on the option used. If opt=U, NSEG segments of equal length are generated, with the lump upstream of the corresponding path. In this case, the user must name the lump at the downstream end of the line into which the last path will flow. If opt = D, NSEG segments of equal length will be generated, with the lump downstream of the corresponding path, and the user must name the lump at the upstream end of the line. If opt = C,* NSEG lumps are generated, centered between NSEG+1 paths. In this case, all lumps are of equal size (if AFI=AFJ), but the first and last paths are one half the length of the remaining paths, and the user must name both the upstream and downstream lumps to which these half-sized paths will attach.

Guidance: The AC factor is calculated automatically for spatial accelerations within the duct due to heating, area changes, or side flow (Section 3.15). Note that rapid cooling or condensation or extreme suction can cause numerically destabilizing decelerations. Make sure all vapor-side losses (such as manifold entrances) are included as LOSS connectors or as FK factors applied at the entrance and exit. If paths are twinned, the FG factor is also automatically calculated.

Guidance: AFI is applied as the AFI value of the first path, and AFJ is applied as the AFJ value of the last path. In between, the flow areas are assumed to vary linearly with length. This results not in a conical shape, but in a horn shape. Use the reinitialization routine MACONE (Section 7.2) to apply an assumption of linearly varying diameters instead.

Guidance: If LU=TANK, the volumes of the tanks will be calculated as AF*TLENT/NSEG where AF may be variable may be variable along the length of the duct if AFI and AFJ are specified. If register-containing expressions are used to define the variables underlying the volume calculation (e.g., DH, TLEN, AFI, etc.) then changes to those registers will be propagated automatically to tank volumes.

* Recommended option

Caution: If flow area changes are modeled using AFI and AFJ, be sure to include FK factors for the irrecoverable losses associated with such area changes. (FKCONE, Section 7.11.1.9)

Caution: TLENT is the *total* length of the line to be generated, not the length of each segment. *Both TLENT and DHS are required inputs with no defaults.*

Restriction: Only tubes and STUBE connector devices may be used in LINE macros. Also, no heat transfer ties are generated, but these may be added via the GENT macro. The GENT,HTN is applicable if opt=U or D, else use GENT,HXC.

Restriction: Only FLAT ifaces may be used between twinned tank pairs used in duct macros.

Example:

```
M LINE,1,U,2,20,1,NSEG=10, TLENT=10.0
      DHS=1.0/12.0,AFS=-1.0,FR=13.0
      LU=JUNC, PA=STUBE
      QL=0.0, QDINC=100.0
      TL=400, TLINC=10.0, UPF=1.0
      FTIE  = AXIAL
```

HX (Tied Duct) Macro Format:

```
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 or AL=R][,QL=R]
        [,TLINC=R][,PLINC=R][,XLINC=R or ALINC=R]
        [,QDINC=R][,PAINC=I]
        [,CX=R][,CXINC=R][,CY=R][,CYINC=R][,CZ=R][,CZINC=R]
        [,CXI=R][,CYI=R][,CZI=R] [,CXJ=R][,CYJ=R][,CZJ=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 ...
```

where:

`mid`.....macro ID
`opt`.....U, D, DS, or C (Up, Down, Down Segmented, or Center)
`ilid`.....ID of first lump
`ipid`.....ID of first path
`itid`.....ID of first tie
`inid`.....ID of first node (smn.id) (tied but not generated)
`l1id`.....ID of lump to connect to (not generated,) at downstream end for opt =U, else at upstream end
`l2id`.....ID of 2nd lump to connect to (not generated,) for opt = C only, at downstream end
`NSEG`.....Number of segments (lumps) to generate (minimum of one, NSEG+1 paths will be generated if opt = C, twice that number if twinned). The value of NSEG can be defined using an expression that results in an integer value (it will be rounded to the nearest integer if not), but changes to that expression during execution will not change the network: element generation is a one-time event performed by the preprocessor based on initial register values.
`NINC`.....node ID increment (in same thermal submodel as first node)
`STAT`.....phase suction status flag:
NORM: . . .normal (extracts both phases from upstream lump)
LRV . . . . . .Liquid suction from upstream, vapor suction from downstream
VRL. . . . . .Vapor suction from upstream, liquid suction from downstream
These options will be applied to all paths. OTHER STAT VALUES ARE ILLEGAL FOR DUCT MACROS.
`TLENT` .....total length of line to be generated
`DHS`.....diameter of line (DH of each path. DEFF may be used in addition.)
`AFS`.....flow area of line (AF of each path)
`AFI/AFJ`....flow area at the defined inlet and outlet of *the entire macro.* If either AFI or AFJ is used, they both must be used.
`LU` .....lump type: TANK, JUNC, or PLEN
`PA` .....path type: TUBE or STUBE connector
`LUINC` .....lump ID increment
`TL` .....init. temp. of first lump
`PL` .....init. pressure of first lump, PL! means pressure has priority over TL
`XL` .....init. quality of first lump
`AL` .....init. void fraction of first lump
`QL` .....init. heat rate of first lump
`TLINC` .....lump TL increment
`PLINC` .....lump PL increment
`XLINC` .....lump XL increment
`ALINC` .....lump AL increment
`QDINC` .....lump QL increment

PAINC..... path ID increment
CX..... location on the X axis
CXINC..... lump CX increment
CY..... location on the Y axis
CYINC..... lump CY increment
CZ..... location on the Z axis
CZINC..... lump CZ increment
CXI etc..... upstream port coordinates (applies only to upstream end of first path, ignored if opt = U)
CXJ etc..... downstream port coordinates (applies only to downstream end of last path, ignored if opt = D)
TWIN ..... unique identifier of initial secondary twin path to be generated (for slip flow modeling), where ipid is the corresponding primary twin
DUPI ..... total upstream duplication factor (applies only to upstream end of first path, ignored if opt = U)
DUPJ ..... total downstream duplication factor (applies only to downstream end of last path, ignored if opt = D or opt = DS)
DUP ..... Sets both DUPI and DUPJ (one of which will be ignored unless opt = C)
DUPN ..... node side tie duplication factor (applies only all ties)
DUPL ..... lump side tie duplication factor (applies only all ties)
AFRACT .... heat transfer area multiplier (ratio of heat transfer area to total wetted area)
TIINC..... tie ID increment
STAY ..... valid only if opt = DS. If present, indicates that the HTNS ties should remain on the defined downstream end of each segment even if the flow rate reverses; otherwise the ties will automatically jump to the current downstream end
LTWIN..... unique identifier of initial secondary twin tank to be generated (for non-equilibrium modeling), where ilid is the corresponding primary tank
LTINC..... LTWIN increment for twinned tanks (Section 3.25)
IFINC..... IDFACE iface increment for twinned tanks (Section 3.25)
IDPINC .... IDPATH superpath increment for twinned tanks (Section 3.25)
IDFTNC .... IDFTIE CONSTQ ftie increment for twinned tanks (Section 3.25)
TTINC..... TTWIN increment for twinned ties along with twinned tanks (Section 3.25)

defaults (if not previously defined in a PA DEF or LU DEF subblock):

NSEG 1
NINC 1
STAT ..... NORM
MCH ..... -2: nonequilibrium phasic expansion with Bursik's metastable two-phase speed of sound (see Section 3.18)
HCH ..... 0.02 (2% hysteresis on critical flow rate, see Section 3.18)

DHS.....none, except calculated for each individual path automatically if DHS is not input but AFI and AFJ are input
AFS.....negative (from DHS assuming circular x-section), or calculated as the average flow area for each path if AFI and AFJ are input
AFI/AFJ....-1.0 (not used)
LU .....TANK
PA .....TUBE
LUINC 1
TLINC 0.0
PLINC 0.0
XLINC 0.0
ALINC 0.0
QDINC 0.0
PAINC 1
CX 0.0
CXINC 0.0
CY 0.0
CYINC 0.0
CZ 0.0
CZINC 0.0
CXI etc .....no upstream port defined
CXJ etc .....no downstream port defined
TWIN.....no twinned paths (single homogeneous paths created)
DUPI.....1.0 (or DUP)
DUPJ.....1.0 (or DUP)
DUP.....1.0
DUPN.....1.0
DUPL.....1.0
TIINC 1
AFRACT.....1.0
LTWIN .....no twinned tanks (single homogeneous lumps created)
LTINC .....LUINC
IFINC .....LUINC
IDPINC.....LUINC
IDFTNC.....LUINC
TTINC .....LUINC

Guidance: An HX command performs the same functions as a LINE command, except that ties are generated between each lump and a thermal node. HTN ties are generated for options U and D, HTNS ties are generated for option DS (which is otherwise identical to option D), and HTNC ties are generated for option C. An HX model is a series of paths and lumps that simulates a pipe or long passage with radial heat addition or rejection. HX segments can be composed of tanks, junctions, or plena, and tubes or

STUBE connectors. If tanks are used, VDOT terms are assumed to be zero initially, and volumes are automatically calculated according to segment size and other options. PA DEF and LU DEF options apply to many inputs, and should be used when possible for clarity except when increment options are desired.

Guidance: The configuration of each segment depends on the option used. If opt=U, NSEG segments of equal length are generated, with the lump upstream of the corresponding path. In this case, the user must name the lump at the downstream end of the line into which the last path will flow. If opt = D or opt = DS, NSEG segments of equal length will be generated, with the lump downstream of the corresponding path, and the user must name the lump at the upstream end of the line. If opt = C,* NSEG lumps are generated, centered between NSEG+1 paths. In this case, all lumps are of equal size, but the first and last paths are one half the length of the remaining paths, and the user must name both the upstream and downstream lumps to which these half-sized paths will attach. AFRACT is a multiplier that may be used to alter the area fractions of all ties. For circumferential wall gradients, additional axial "rows" of ties may be added to the duct via the GENT macro. Use GENT,HTN for opt=U or D, the GENT,HTNS for opt = DS, use GENT,HXC for opt = C.

Guidance: The AC factor is calculated automatically for spatial accelerations within the duct due to heating, area changes, or side flow (Section 3.15). Note that rapid cooling or condensation or extreme suction can cause numerically destabilizing decelerations. Make sure all vapor-side losses (such as manifold entrances) are included as LOSS connectors or as FK factors applied at the entrance and exit. If twinned, the FG factor is also automatically calculated.

Guidance: AFI is applied as the AFI value of the first path, and AFJ is applied as the AFJ value of the last path. In between, the flow areas are assumed to vary linearly with length. This results not in a conical shape, but in a horn shape. Use the reinitialization routine MACONE (Section 7.2) to apply an assumption of linearly varying diameters instead.

Guidance: If LU=TANK, the volumes of the tanks will be calculated as AF*TLENT/NSEG where AF may be variable along the length of the duct if AFI and AFJ are specified. If register-containing expressions are used to define the variables underlying the volume calculation (e.g., DH, TLEN, AFI, etc.) then changes to those registers will be propagated automatically to tank volumes.

Caution: If flow area changes are modeled using AFI and AFJ, be sure to include FK factors for the irrecoverable losses associated with such area changes. (FKCONE, Section 7.11.1.9)

Caution: TLENT is the *total* length of the line to be generated, not the length of each segment. *Both TLENT and DHS are required inputs.*

* Recommended option

Caution: Use of opt = DS without the STAY command will cause the first tie to jump off the macro in the event that the flow rate in the first segment reverses.

Restriction: Only tubes and STUBE connector devices may be used in duct macros. (Extended duct macros offer other choices, as explained below.)

Restriction: Only FLAT ifaces may be used between twinned tank pairs used in duct macros.

Example:

```
M HX,1,C,2,20,2,BATT.3,1,101, NSEG=5, NINC=10
      TIINC=10,AFRACT=0.75
      TLENT=50.0, DHS=0.01, PA=TUBE, LU=TANK
      LUINC=1, PAINC=20, PL!=14.7, PLINC=-0.01
```

3.9.2.1 Macro-level Output Options

Duct macros contain both lumps and paths, and understanding their current status is often difficult by postprocessing individual elements or by tabulating lumps and paths separately.

Two specialized tabulation-style output routines are available for summarizing the inputs and current solution status of duct macros: MACINTAB and MACROTAB. MACINTAB is intended to be used as an input check, and therefore is most useful when called from OPERATIONS. MACROTAB is provides current information on flow rates, pressures, etc. and is therefore most useful when called from OUTPUT CALLS. See Section 7.11.8 for full details.

3.9.2.2 Extended Macros in Sinaps and FloCAD

FloCAD and Sinaps “Pipes” are either represented directly as a single SINDA/FLUINT LINE or HX (“duct macro”) in a one-to-one relationship, or they are combined with several other similar Pipes into an “extended” duct macro. They can also collect various lumps and paths into a duct macro, even if no Pipe is involved.

A duct macro includes important momentum gradient terms in the mathematical solution that would be absent in a set of individually input series of lumps and paths. (Section 3.15.3 contains more details on spatial acceleration terms.)

For this reason, both Sinaps and FloCAD collect any serial flow sections (path-lump-path- ... sequences) they can* into automatically generated duct macros.

For the same reason, whether using a GUI or using the text-based SINDA/FLUINT input, when modeling a long continuous pipeline, it is recommended that a single duct macro be used.

* Any tube or STUBE or REDUCER or EXPANDER that meets another tube (or STUBE or REDUCER or EXPANDER) at a point of equal flow area is susceptible to being combined into an automatic duct macro, with the lump in between the two paths aggregated into the macro in the process. More complicated explanations of the sometimes complex decision making process are not required since the user can have the automatic macros depicted (by color) at any time, and has discretion to avoid the aggregation at any point in the network. Details of both depicting these automatic aggregations and controlling them are specific to each GUI, so the user is referred to those user’s manuals for more information. For example, use the command “RCMACROLIST” in FloCAD to see what it has collected into duct macros.

To explain this statement, consider a single pipe which has been modeled using a single duct macro of adequate axial resolution (say, 30 lumps). If instead that pipe had been divided into two half-length duct macros of 15 lumps each, the predictions *would not be identical* because each duct's momentum gradients would be solved independent of the other's. In other words, the momentum control volume in the middle of the original single-duct model would be split in the second two-duct model, and carry-over (momentum flux gradient) terms would be lost at that point. Normally, this effect is small, but there is still a finite penalty to be paid for splitting a single long duct into two or more shorter ducts.

LINE and HX options, and individual Sinaps/FloCAD Pipes, can *only* contain similar elements (e.g., all tanks or all junctions). If tanks are only needed in one section of the duct (say, the inlet), all lumps must be tanks. If a vertical up-flow section requires twinned tubes for slip flow, all paths must be twinned. Furthermore, it can be difficult to customize a long pipeline (e.g., change elevations) in SINDA/FLUINT since complex user logic may be required to accomplish this initialization task.

To overcome these difficulties, and to encourage (rather than discourage) the use of many specialized pipes, perhaps in series, Sinaps and FloCAD not only automatically collect any possible individual lumps and paths into a macro, they collect Pipes too: a single extended macro may therefore consist of several Pipes as well as individually input lumps and paths. These extended macros may contain mixed types of elements (tanks and/or junctions, tubes and/or STUBEs, twinned or not, etc.) such that the momentum carry-over (gradient) terms and seldom lost due to modeling choices.

It is important to note that these extended macros cannot be created using the SINDA/FLUINT text-based input method, since in SINDA/FLUINT the only duct macros are those that are created using a LINE or HX macrocommand.*

Extended duct macros in Sinaps and FloCAD can collected together into a single duct macro:

1. Zero, one, or more Pipes.
2. A mix of tubes and STUBE connectors. (In FloCAD, this mix can also include REDUCER and EXPANDER connectors.)
3. A mix of twinned and untwinned tube and STUBE connectors.
4. A mix of tanks, twinned tanks, and junctions.

For flat front modeling (Section 3.27), it is best to avoid junctions, STUBE connectors, and best to stay consistent (e.g., twinned tanks with twinned tubes). For these reasons, flat front duct macros are usually limited to a single Pipe.

* Sinaps and FloCAD accomplish this feature by communicating directly with the second pass of the SINDA/FLUINT preprocessor, which is not available to the text-based SINDA/FLUINT user. The user should not alter this communication (evidenced by the "MACRO=i,j,k" lines in the *.sin or *.cc files) nor attempt to mimic it with manually inserted "MACRO= ..." mark-ups, otherwise serious processor problems may arise.

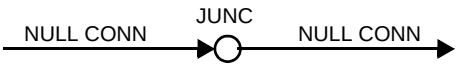
3.9.3 Capillary Evaporator Pump Model

A large number of common fluid system components may be modeled in FLUINT by using a single element (this includes device models for connectors). However, certain complex components require the characteristics of both lumps and paths and perhaps ties, fties, and ifaces in their descriptions. Such components are handled using multi-element *component models*.

Like macros, component model commands generate several elements. Unlike macros, model commands only generate a fixed number of elements. These elements are handled together as a single unit by model simulation logic that is automatically invoked by naming a model. Although users must be aware of the elements being generated (in order to provide them with identifiers), they do not need to remember the specifics of every parameter for every element. For example, although connectors used in component models may not correspond to any device model (such as STUBE or CAPIL) and may be denoted as NULL connectors, the user is not responsible for manipulating their GK and HK values. The parameters of each element in the component model are manipulated collectively inside the processor.

The remainder of this section describes the only* component model currently available, the CAPPMP. See also the discussion on capillary modeling and phase suction options in the next section, the SPRIME routine described in Section 3.11 and documented in Section 7.11.1.7, WICK ifaces presented in Section 3.8.3.4, and USER fties (Section 3.7) along with the WETWICK, CYLWICK, and PLAWICK auxiliary routines for modeling back conduction (Section 7.11.9.4).

The CAPPMP Macro—Capillary pump model (*applicable only in two-phase single-constituent submodels*). This macro generates a junction between two NULL† connectors:



The simplest way to think of a CAPPMP model is a CAPIL connector with a junction in the middle. The presence of the junction allows heat to be added or removed from a CAPPMP model, whereas a CAPIL connector is adiabatic like any path. *It has no other physical meaning, and the state of the junction is largely meaningless.* If the heat rate into the junction is zero, the CAPPMP behaves exactly like a CAPIL connector. *Unlike a CAPIL, the CAPPMP is direction sensitive.* The vapor end (the lump at which heat transfer may be assumed to occur) must be designated.

If the junction QDOT‡ is positive, the CAPPMP simulates an evaporator-pump. Liquid at the designated liquid end (i.e., *not* the designated vapor end) can be evaporated and forced into the designated vapor end, performing pumping by capillary action. If the pressure drop is too big or a capillary interface cannot otherwise exist, the device will *deprime*. This will generate error messages if the flow rate through the junction suddenly drops yet the QDOT continues to apply. See the

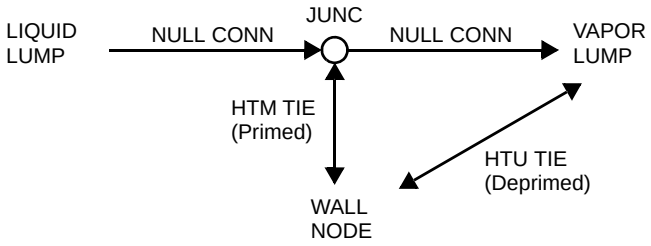
* In previous versions, a hardware-specific model was also available; others may be added in the future.

† Connectors generated as part of models that do not correspond to any existing device are called NULL connectors. GK and HK are calculated internally and are not the responsibility of the user.

‡ Sum of QL plus perhaps a QTIE from a CAPPMP HTM tie, plus perhaps any QFs of fties active on that junction. However, fties are strongly discouraged from being attached to the CAPPMP junction. Normally, the QDOT of the junction will either be user-specified QL (NOTIE option) or the QTIE resulting from the HTM tie (TIE option).

discussion in Section 3.11 and the subroutine SPRIME (Section 7.11.1.7). If the junction QDOT is negative, any vapor present in the designated vapor side will be condensed, and any liquid will be allowed to pass to the liquid side if it is at a lower pressure.

As an alternative to directly specifying the QL of a CAPPMP, the user may generate an HTM tie and provide a UA coefficient. *No other ties may be added to a CAPPMP junction, and neither should fties be placed onto that junction* (they should be applied instead to the endpoint lumps). Because the junction within a CAPPMP macro is really only a location into which heat is added or subtracted and has no physical significance, the heat rate into the tie is *not* simply the product of the UA coefficient times the temperature difference between the node and the junction. When primed, the saturation temperature is used instead of the junction temperature. *When deprimed, the tie will spontaneously jump to the designated vapor lump and become an HTU tie with the same UA value*. This usually eliminates the error messages that would result when a CAPPMP deprimes with a constant heat rate still being applied. If the CAPPMP reprimes, the tie will reappear at the central junction, and once again become an HTM tie. In either mode, the UA value can be modified. When tied, the CAPPMP macro can be schematically represented as (with path flow rates shown positive for evaporation):



The values of RC, CFC, XVH, and XVL are available to the user in logic blocks for inspection or modification. The ID of the first connector should be used to reference these parameters. In other words, the first (defined upstream) connector may be treated like a CAPIL connector for translation purposes.

The user should understand the assumptions made in this simplified model and their impact on modeling a real capillary evaporator-pump, especially in off-design simulations:

1) The CAPPMP, like any FLUINT element, is fundamentally a one-dimensional component: it must be either deprimed or not. Real devices can exhibit more two-dimensional effects such as preventing vapor backflow and even pumping in some regions of the wick, while simultaneously leaking vapor back into the liquid space in other locations. (Several parallel CAPPMPS might be used to simulate such effects.) Furthermore, meniscus recession into the wick is not included except as modeled separately using a parallel (and strongly recommended) WICK iface (Section 3.8.3.4).

2) The endpoint lumps are perfectly mixed, having a single thermodynamic state. While twinned tanks (Section 3.25) have been used successfully to model deprime, and the WICK iface (Section 3.8.3.4) has been used to model the spring-like meniscus, special care is needed to model behavior during deprime, or during clearing of the vapor lump during start-up.

3) While real wicks can exhibit considerable hysteresis (on the order of 50%) between the pressure differences at bubble formation and cessation, only a very small hysteresis (about 4%) is employed internally, purely for numerical stability.

The user should use junctions cautiously at the ends of a CAPPMP model. Junctions are much more likely to cause inadvertent deprime because they must react instantly to changes. On the other hand, rapid quality changes in an endpoint junction can adversely affect CAPPMP stability.

Finally, note that a much more rugged but inflexible model of a capillary pump is simply an MFRSET connector and a heater junction (e.g., a junction whose enthalpy is fixed by a call to HTRLMP, Section 7.11.1.4). The enthalpy of the junction is set to saturated or slightly superheated vapor, and the SMFR of the connector is set to the desired heat rate divided by the enthalpy rise (i.e, that of the junction minus the inlet). Such a model cannot deprime.

Training notes specific to modeling capillary devices are available from CRTech. See the preface section for contact information.

CAPPMP (Capillary Pump) Format:

```
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 or AL=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]
```

where:

`mid` . . . . . macro ID
`opt` . . . . . TIE. . . . . Generate a tie and provide a UA coefficient.
NOTIE. . . . Don’t generate tie, provide QL instead.
`ucid` . . . . . ID of generated upstream connector
`jid` . . . . . ID of generated junction
`dcid` . . . . . ID of generated downstream connector
`ulid` . . . . . ID of nongenerated lump upstream of ucid
`dlid` . . . . . ID of nongenerated lump downstream of dcid
`tid` . . . . . ID of generated HTM tie
`tsmn.nid` . . thermal node representing structure
`RC` . . . . . effective two-dimensional capillary radius
`CFC` . . . . . capillary flow conductance through entire passage
`XVH` . . . . . capillary priming dryness, upper quality limit
`XVL` . . . . . capillary priming dryness, lower quality limit
`TL` . . . . . init. temp. of junction jid
`PL` . . . . . init. pressure of junction jid
`XL` . . . . . init. quality of junction jid
`AL` . . . . . init. void fraction of junction jid
`QL` . . . . . init. heat rate of junction jid

FR. . . . . init. flow rate through CAPPMP
DUPI . . . . . total upstream duplication factor (applies only to upstream end of ucid)
DUPJ . . . . . total downstream duplication factor (applies only to downstream end of dcid)
DUP . . . . . sets both DUPI and DUPJ
DUPN . . . . . node side tie duplication factor (for TIE option)
DUPL . . . . . lump side tie duplication factor (for TIE option)
VAPOR. . . . . "vapor" side: CAPPMP endpoint lump (ulid or dlid) at which heat transfer occurs (location of deprimed tie for TIE option)
UA. . . . . initial heat transfer conductance for the tie (required if opt=TIE)
AF. . . . . flow area for kinetic energy and choking calculations
AFTH . . . . . throat area for choking calculations
MCH . . . . . choked flow detection and modeling method flag (see Section 3.18)
HCH . . . . . hysteresis fraction to apply to critical flow rate (see Section 3.18)

defaults (if not previously set in a PA DEF or LU DEF subblock):

XVH 0.99
XVL 0.95
XL. . . . . use AL if input, else 0.0 if neither is input
QL. 0.0
FR. 0.0
DUPI . . . . . 1.0 (or DUP)
DUPJ . . . . . 1.0 (or DUP)
DUP 1.0
DUPN 1.0
DUPL 1.0
VAPOR. . . . . dlid (default is valid positive flow for evaporation)
AF. . . . . -1.0 (A nonpositive value signals that AF should be calculated by assuming a circular cross section of diameter DH if DH was defaulted in a prior PA DEF block. Otherwise, if AF is nonpositive then no kinetic energy can be extracted nor choking calculations applied). For porous materials, consider a value of the frontal area multiplied by the porosity.
AFTH . . . . . AF
MCH . . . . . -2: nonequilibrium phasic expansion with Bursik's metastable two-phase speed of sound (see Section 3.18)
HCH . . . . . 0.02 (2% hysteresis on critical flow rate, see Section 3.18)

Guidance: Similar to a CAPIL connector, but generates a junction between two connectors. Pumping may occur according to the heat source/sink term on the junction: QDOT(jid). This source may either be input and/or calculated with an HTM tie given user-input UA, which operates similar to an HTU tie. Note that RC, CFC, XVH, and XVL are available for manipulation within user logic blocks as RC(ucid), CFC(ucid), XVH(ucid), and XVL(ucid). Nonpositive RC will be interpreted as a perfect, infinite

pressure difference capillary structure. See also the SPRIME routine in Section 7.11.1.7.

Guidance: XVH and XVL represent the limits of a hysteresis cycle. An adjacent lump with a quality higher than the current value (XVH or XVL, depending on past history) is considered dry enough to permit the device to prime, or perhaps too dry to permit priming, depending on whether the lump has higher or lower pressure than the opposite lump.

Guidance: When a TIE option CAPPMP deprimes, the tie jumps to the designated vapor endpoint lump and becomes an HTU tie. It returns to the central junction and becomes an HTM tie if it reprimes. No analogous movements of the QL are possible for the NOTIE option. Refer to Section 3.11.2 for prime/deprime logic.

Caution: Changes to QL(jid) or UA during execution should be gradual except when changing to or from zero.

Restriction: No other ties may be placed on the central junction.

Restriction: Cannot evaporate in the presence of nonvolatile liquids in a mixture if a junction (or STEADY tank) is used to represent the liquid side. Also, cannot condense in the presence of noncondensible gases if a junction (or STEADY tank) is used to represent the vapor side. The program will stop if a CAPPMP is used in an inappropriate mixture, or may issue a one-time warning and continue if the restriction is conditional on whether the CAPPMP evaporates or condenses.

Restriction: Owing to the unique nature of CAPPMP macros, the user should refrain from using advanced expression indirect referencing on certain path and tie terms to avoid ambiguities. For example, referring to “PL#up” or “AL#down” on CFC, RC, DUPI, or DUPJ terms can have unexpected results since two (not one) NULL paths are generated. Similarly, referring to “TL#lump” on the UA, DUPN, or DUPL of the HTM tie can lead to unexpected results since a nonphysical junction state is sometimes referenced, and sometimes the tie can move (when deprimed) to the vapor lump. Note that such a concern does not exist with the node side (“T#node”) of the tie.

Examples:

```
M CAPPMP,10,NOTIE,12,2,23,1,3,XL=1.0
      CFC=1.0E-10, RC=1.0E-4
      QL=1000.0, FR=-100.0
M CAPPMP,11,TIE,22,223,23,1,3,223,WALL.220
      CFC=1.0E-10, RC=1.0E-4
      UA=10.0,XVH = 0.85, XVL = 0.8
```

3.10 Sample Fluid Submodel

To help clarify FLOW DATA input procedures, a sample fluid submodel is given. A model description is given in Table 3-12 with the input file in Table 3-13.

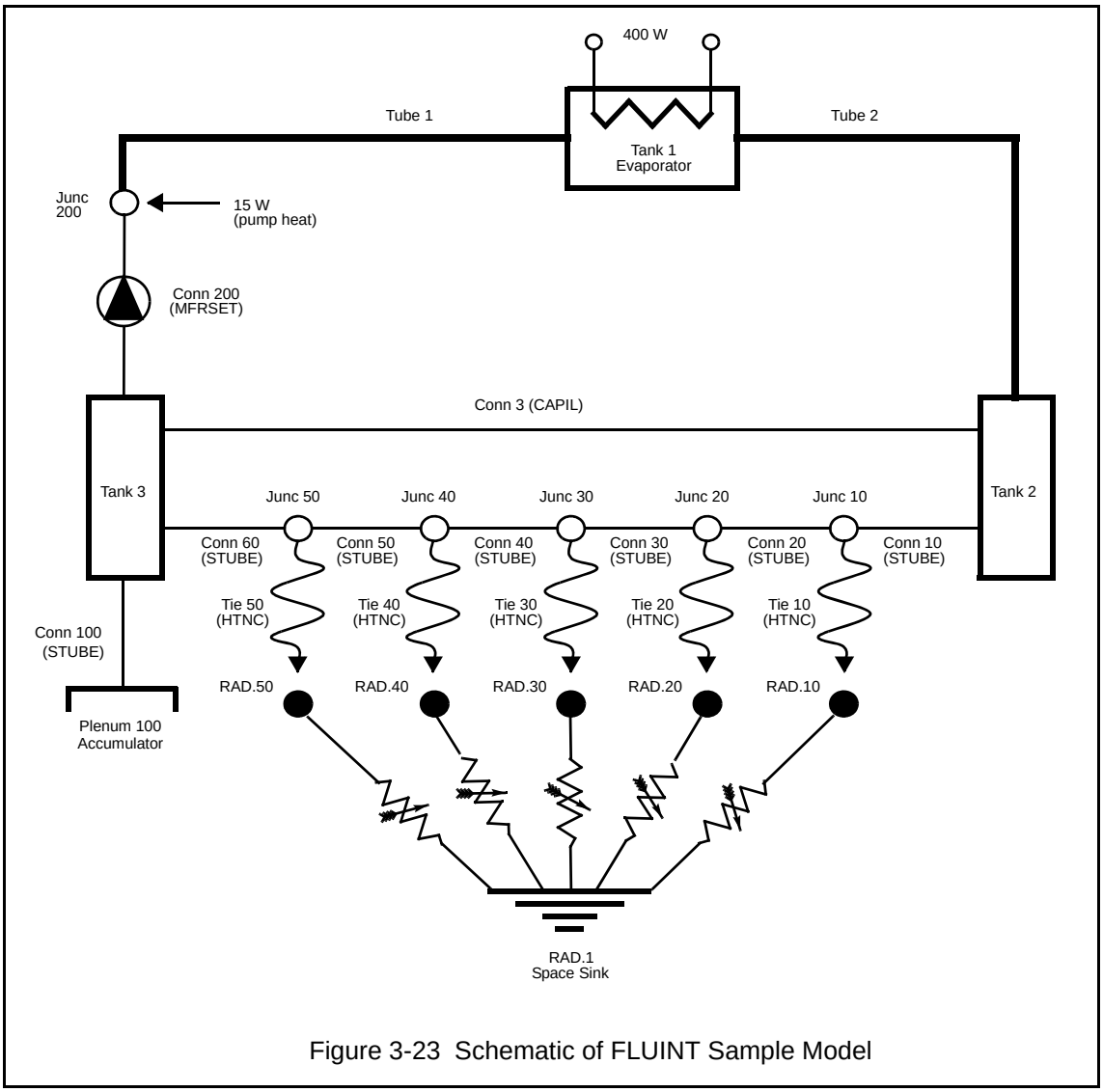


Figure 3-23 Schematic of FLUINT Sample Model

The sample fluid submodel consists of the simple fluid loop pictured in Figure 3-23. The system consists of a pump circulating pure ammonia between a heat source and a heat sink. The system will be initialized as a two-phase transport loop, but FLUINT will continue to operate if the system becomes hard-filled or all vapor. A wicked bypass is being investigated to allow excess liquid to bypass the condenser. The element breakdown is described in Table 3-12. Note that the heat sink is a radiator modeled by a thermal network called RAD. The diameter of all lines are the same: 0.008. To facilitate input and to enable consistent changes and parametric analyses, this value has been stored as a register named “Diam.”

Table 3-12 Sample Model Description

ELEMENT	DESCRIPTION
TUBE 1	Liquid feed line from pump to evaporator.
TANK 1	Evaporator, containing all of the volume from the pump outlet to the condenser inlet.
TUBE 2	Vapor or two-phase line from evaporator to condenser.
TANK 2	Condenser inlet, containing half of the condenser volume.
JUNCTIONS 10-50	Heat exchanger segments that divide up the condenser into five parts.
STUBEs 10-60	Line segments in the condenser, corresponding to the above. #10 and #60 are half-length.
HTNC TIES 10-50	Tie LOOP.N invokes automatic convection heat transfer between junction LOOP.N and node RAD.N. Note centered differencing and area fractions mean #10 includes all of STUBE #10 and 1/2 of STUBE #20, etc.
TANK 3	Tee at the outlet of the condenser that connects the condenser, the bypass, the line to the accumulator, and the pump. Contains the second half of the condenser volume.
CAPIL 3	Wicked bypass connecting condenser inlet and outlet
STUBE 100	Small line to link the accumulator into the system.
PLENUM 100	Accumulator. Stagnant.
MFRSET 200	Idealized pump (constant mass flow rate).
JUNCTION 200	Between pump and delivery line (TUBE 1).
NODES 10-50	Diffusion nodes, each modeling 1/5 of the radiator.
NODE 1	Boundary node, representing effective radiation sink.

Table 3-13 FLOW DATA Block for Sample System

```
HEADER FLOW DATA,LOOP,FID=7717                $ AMMONIA FILLED “LOOP”
C
C DEFINE MODEL DEFAULTS
C
LU DEF, TL  = 300.0                           $ DEFAULT TEMPERATURE IS 300 K
           PLFACT     = 101325.0              $ PL MULTIPLIER (INPUT IN ATM)
           PL         = 6.0                   $ 6 ATM IS DEFAULT PRESSURE
           XL         = 0.0                   $ DEFAULT STATE IS LIQUID
PA DEF, FR  = 5.0E-4                          $ FLOW RATE IS 0.5 GRAMS/SEC
           DH         = Diam                  $ DEFAULT DIAMETER IS 8 mm
           TLEN       = 2.0                   $ DEFAULT DUCT LENGTH IS 2 METERS
           WRF        = 1.0E-6                $ ASSUME FAIRLY SMOOTH PIPE
           UPF        = 0.5                   $ USE AVG. OF UP/DOWNSTREAM
           IPDC       = 6                     $ USE FLOW REGIME MAPPING
C
C START AT PUMP OUTLET, GO FLOWWISE AROUND LOOP
C
PA TUBE,1,200,1                               $ TUBE FROM PUMP TO EVAPORATOR
C
LU TANK,1                                     $ EVAPORATOR CONTROL VOLUME
           VOL        = 4.0*Diam^2*0.25*pi
           QL         = 400.0
           XL         = 0.1                   $ NOTE PL WILL BE P AT SATURATION
C
PA TUBE,2,1,2                                 $ TUBE FROM EVAP TO SEPARATOR
C
LU TANK,2                                     $ SEPARATOR AND UPPER 1/2 OF COND
           VOL        = 2.5*Diam^2*0.25*pi
           XL         = 0.1                   $ NOTE PL WILL BE P AT SATURATION
C
C WICK A = 0.02, T = 1 cm.
```

Table 3-13 FLOW DATA Block for Sample System (concl)

```
C
PA CONN,3,2,3, FR = 1.0E-4,DEV = CAPIL
                     RC = 16.5E-6,CFC = 2.6E-13*0.02/0.01
C
C START CONDENSER LINE (THE FOLLOWING 20 LINES WERE REPLACED BY
C AN M HX,C MACRO SUBBLOCK. THIS INCLUDES DECELERATION OF VAPOR)
C USE CENTERED DIFFERENCING
C
CLU JUNC,10, XL = 0.5          $
CLU JUNC,20, XL = 0.3          $ NOTE QUALITY "GRADIENT"
CLU JUNC,30, XL = 0.1          $
CLU JUNC,40, TL = 290.0        $ DEFAULT XL=0.0
CLU JUNC,50, TL = 270.0        $ SUBCOOLED SECTION
C
C NOW DESCRIBE STUBE CONNECTORS THAT CORRESPOND TO ABOVE JUNCS
C
CPA DEF, TLEN = 1.0, FR = 4.0E-4
CPA CONN,10,2,10,       DEV = STUBE,    TLEN = 0.50
CPA CONN,20,10,20,      DEV = STUBE
CPA CONN,30,20,30,      DEV = STUBE
CPA CONN,40,30,40,      DEV = STUBE
CPA CONN,50,40,50,      DEV = STUBE
CPA CONN,60,50,3,       DEV = STUBE,    TLEN = 0.50
C
C SET HEAT TRANSFER TIES IN CONDENSER
C NOTE CENTER PATHS ARE SHARED AMONG TIES
C
CT HTNC,10,10,RAD.10,   10,1.0, 20,0.5
CT HTNC,20,20,RAD.20,   20,0.5, 30,0.5
CT HTNC,30,30,RAD.30,   30,0.5, 40,0.5
CT HTNC,40,40,RAD.40,   40,0.5, 50,0.5
CT HTNC,50,50,RAD.50,   50,0.5, 60,1.0
C
C THE PREVIOUS 20 OR SO LINES WERE REPLACED WITH:
C
M HX,1,C,10,10,10,RAD.10,2,3
           NSEG = 5,            PA = STUBE,          LU = JUNC
           NINC = 10,           LUINC = 10,          PAINC = 10,TIINC = 10
           TLENT = 5.0,DHS = Diam,FR = 4.0E-4
           XL = 0.4,            XLINC = -0.1,UPF = 0.5
C
C NOTE THAT INITIAL CONDITIONS ARE DIFFERENT BECAUSE OF XLINC,
C AND THAT AN HX MACRO ADDS SPATIAL ACCELERATION FACTORS AUTOMATICALLY
C
C FINISH LOOP TO PUMP, THEN ADD ACCUMULATOR
C
LU DEF, TL = 270.                          $ RESET DEFAULT TEMP. TO 270 K
C THE FOLLOWING TANK CONTAINS THE SECOND HALF OF THE
C VOLUME OF THE CONDENSER.
C
LU TANK,3, VOL = 2.5*Diam^2*0.25*pi
PA CONN,200,3,200, DEV = MFRSET            $ SIMPLEST PUMP MODEL
C                                          NOTE THE SMFR EQUALS THE DEF. FR.
C
LU JUNC,200                                $ JUNC TO MATE PUMP TO EVAP. TUBE
           QL        = 15.0                $ 15 WATTS PUMP HEATING
PA CONN,100,3,100              $ FLOW IS POSITIVE INTO ACCUM
           FR        = 0.0,    DEV         = STUBE
           TLEN      = 0.1,    DH          = Diam
LU PLEN,100, LSTAT   = STAG    $ ACCUMUL.
```

3.11 Capillary Models and Phase Suction Options

Both capillary models (CAPIL connectors and CAPPMP macros) and phase suction options require additional attention from the user. Because capillary models internally use the phase suction options, these two topics are discussed together.

Twinned tubes and STUBE connectors also manipulate the phase suction options. The primary path is assigned STAT=DLS when two phases are present, and the secondary path is assigned STAT=DVS. While some of the following discussion is therefore relevant to twinned paths, separate sections describe their behavior and usage.

3.11.1 Phase Suction Options

Almost any path can be made to extract only liquid or only vapor (or gas) from an upstream two-phase lump *if that phase is available in sufficient quantity.* This is a useful tool for simulating liquid-vapor separators, capillary devices, and stratified tanks. For example, the vapor barrier properties of a filter or small diameter passage may be simulated with a liquid suction option. (Liquid suction options are internally set and reset by the CAPIL and CAPPMP capillary models.) Also, a vent above the liquid surface of a stratified two-phase tank may be simulated with a vapor suction option.

Phase suction options may be set in the FLOW DATA block using the STAT flag,* and may be reset during execution using the CHGSUC routine.† These options are specified by the following character flags:

NORM . . . suction of both phases applied (default)
LS. . . . . Extracts liquid from upstream end (liquid suction)
VS . . . . . Extracts vapor from upstream end (vapor suction)
RLS . . . . . Extracts liquid from *downstream* end (reverse liquid suction)
RVS . . . . . Extracts vapor from *downstream* end (reverse vapor suction)
DLS . . . . . Extracts liquid from both end (double liquid suction)
DVS . . . . . Extracts vapor from both ends (double vapor suction)
LRV . . . . . Liquid-only from the defined upstream, vapor-only from the downstream
VRL . . . . . Vapor-only from the defined upstream, liquid-only from the downstream

”Upstream” and “downstream” are based on the defined positive flow rate direction. The designation does not change with flow rate reversals. Note that the RLS and RVS options merely save the user from having to reverse a path, use the LS or VS options, and deal with negative flow rates.

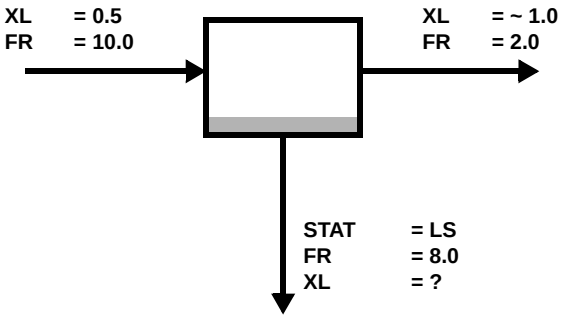
Phase suction action can only occur when the upstream lump is in a two-phase state. When the upstream lump is single-phase, the path will resume normal operation (i.e., phase suction action will be ignored). Phase suction action alone cannot prevent all flow through a path if the upstream single-phase lump contains the “wrong” phase. ***Phase suction is a request, not a demand.***

* Path end locations, if used, may also affect phase suction. See Section 3.13.4.
† Phase suction options are also affected by ports (Section 3.13.4) and flat front options (Section 3.27).

Note that phase suction options can affect the behavior of a path even if the path is flowing in the opposite direction from the suction action. The path will "see" only the appropriate phase for calculations which depend on downstream properties as well as upstream. Such calculations can include body forces, frictional resistance, and spatial accelerations. Thus, phase suction options may be used on downstream ends of paths merely to affect the properties used.

The quality "seen" by a phase suction path is seldom that of the upstream lump. The PTHTAB routine therefore lists "XL UPSTRM" as the quality used by that path. This effective quality may also be retrieved using the XTRACT subroutine (Section 7.11.1.10).

The behavior of an upstream lump can be very different depending on whether it is a junction or a tank. (Recall that tanks are treated like junctions in STEADY or "FASTIC.") To understand this difference, assume a liquid-vapor separator has been modeled as a tank with one incoming two-phase path and two exiting paths as shown at the right. One of the exiting paths extracts only liquid when that phase is present.



Suppose that, for whatever reasons, the all-liquid leg tends to operate at a higher flow rate than the vapor leg. If the incoming fluid quality does not exactly correspond to the outgoing phase distribution, then *no steady-state solution exists.* The liquid path will extract all available liquid, driving the quality in the tank to 1.0. At that time, with no available liquid, the liquid path will start to extract vapor (as would a submerged pipe penetration or a deprimed capillary barrier) until liquid is again present. As a result, the quality in the tank and therefore the quality of the vapor exit path hovers near 1.0, while the fluid in the liquid suction path alternates between all vapor and all liquid. A STDSTL run will not converge because there is no time-independent state. (As an aside, note that real fluid systems often lack a time-independent state due to a variety of causes.)

If the separator were modeled with a junction (or were analyzed in STEADY), the situation would be different. FLUINT will attempt to produce *time-averaged* results if time-independent results are impossible. In the above example, the program will converge in STEADY with a tank quality of nearly 1.0 (perhaps 0.9999). The liquid path quality will be 0.375 to balance mass and energy. This does not necessarily mean that this path will carry vapor 37.5% of the time, nor that the behavior of the subsystem downstream of that path will represent an average behavior between all-liquid and all-vapor states. However, mass and energy will balance and all phases will be distributed as much as possible in accordance with the phase suction status. Phase suction thus takes on a more liberal meaning. Using liquid suction on a path means that the path will extract *as much liquid as possible* up to a limit of 100% liquid.

The goal of the above methodology is to provide good initial conditions for STDSTL or for transient runs. Because of the lack of a steady-state solution, the results produced by STEADY *might not* correspond to any physical state. The situation is analogous to orbital average fluxes and temperatures, which are useful as sizing information and as initial conditions for transients, but which will probably never really occur.

The user must be therefore cautioned regarding STEADY solutions and phase suction from junctions in general, but there are several strong reasons to *not* avoid them. First, *the solution reached by STEADY is usually a true time-independent state.* Suction path qualities between 0.0 and 1.0 are indications of a time-averaged state. Second and most important, analytic tests with complex capillary systems have shown that the FLUINT solutions are indeed very close to the average behavior of the oscillating system, and are much easier to study than the true oscillating solution.

As a result of active phase suction options, the quality of a junction or STEADY tank is not necessarily equal to a quality resulting from a perfect mixing of incoming flows. If this were true, the quality of the fluid inside the aforementioned liquid-vapor separator would be 0.5. Actually, the time-averaged quality will be nearly 1.0 because of the tendency of the liquid path to extract all available liquid. This shifting is taken into account by the code, and has important ramifications on liquid inventory and heat transfer calculations. Suppose the tank in the above example were in a single-constituent submodel, that it were tied to a cooler wall node, and that the UA value is calculated for film condensation. The dryer the tank, the higher the UA coefficient and the heat rate out of the tank will become, causing more condensation and lower quality. These opposing forces will reach an equilibrium quality lower than that of an adiabatic tank, and/or will allow more liquid to flow out of the liquid suction path. Such considerations are taken into account by the code, and should be remembered by the user when manipulating UA values for HTU or HTUS ties.

Constant enthalpy in a two-phase single-constituent state implies nearly constant quality. When phase suction options are active out of a two-phase STEADY tank or junction whose enthalpy has been frozen by a call to HTRLMP (Section 7.11.1.4), out-flowing suction paths will have qualities of 1.0 or 0.0 since the desired phase is assumed to be available. The lump QDOT will be calculated to balance total energy flow through the heater lump, ignoring QL and other heat rate inputs. This is consistent with the quality shifting as mentioned above; using phase suction options, *phases may appear to move from inlet to outlet without ever mixing*. As with any lump, if the quality in the heater lump is either 0.0 or 1.0, all exit paths will carry that quality.

3.11.2 Multiple-constituent Notes

Although phase suction and CAPIL connectors may be used in multiple-constituent two-phase submodels, the CAPPMP macro cannot be used unless phase change is possible in such submodels. In other words, the CAPPMP macro can only be used if one of the constituents in the mixture is condensible/volatile. A preprocessor error will occur if a purely nonvolatile and noncondensible mixture is specified.

Species-specific suction options (Section 3.19.4) do not function from junctions (nor from tanks within the STEADY (“FASTIC”) solution routine, since these are treated temporarily as junctions). This places limits on the use of CAPPMP macros in mixtures: they cannot condense from junctions containing noncondensible gases, nor evaporate from junctions containing nonvolatile liquids.

The surface tensions of all liquid constituents must be supplied in order to use CAPIL connectors or CAPPMP macros. This is strongly recommended in any case since this data is required also to perform heat transfer calculations, exercise flow regime mapping options, and enable the modeling of slip flow.

The remainder of this section is only applicable to flows without phase change if one disregards heat addition or rejection, since such concepts have no meaning in CAPIL connectors.

3.11.3 Capillary Primed vs. Deprimed Decision

Because of the need to manage liquid position in low gravity, capillary devices are common in spacecraft systems. In FLUINT, available capillary models* include CAPIL connectors and CAPPMP macros. These models require special attention because (1) they internally use the phase suction options, and are therefore subject to the previous discussion, and (2) they can exist in one of two very different states: *primed* or *deprimed*. FLUINT's definitions of these terms, detailed in the next paragraph, are not necessarily intuitive, having more to do with whether or not a steady meniscus exists than whether the wick is dry or wet.

A capillary device such as a wick, groove, slot, or tubule is primed if it is full of liquid and is adjacent to vapor whose pressure is greater than or equal to the pressure of the liquid but less than or equal to that supported by the capillary radius of the device. (The capillary pressure limit is equal to $2 \bullet \sigma / RC$, where σ is the fluid surface tension and RC is the effective two-dimensional capillary radius of the liquid meniscus in the capillary structure.) Otherwise, it is deprimed. For the purposes of the following discussion, "deprimed" means that the flow rate becomes a function of the resistance and the current pressure gradient. (For the advanced user, the distinction between primed and deprimed may be summarized concisely: the connector GK is zero if primed, and positive if deprimed.) If heat is added to a primed device, it may be able to evaporate the liquid into the higher pressure vapor space, thereby becoming an *evaporator-pump*. On the other hand, *a flooded capillary device is considered deprimed in FLUINT even if it can still block vapor flow* (e.g., heat inputs are low and the pressure gradient is less than the capillary limit).

Unfortunately, capillary devices can be easily deprimed (more so analytically than in practice). First, if the pressure differential exceeds the capillary limit,† any fluid in the higher pressure end will be able to flow back into the lower pressure side. This dries out the capillary structure, which can then only be rewet if the flow is reversed or if sufficient condensation occurs. Second, if the liquid pressure is greater than that of the vapor, liquid will blow through into the vapor. Third, if too much liquid appears in the "vapor" end, it causes the device to deprime and the liquid will flow back to the lower pressure end until it is exhausted, at which time the device might reprime. Fourth, a heated device (i.e., a CAPPMP) cannot stay primed if its own internal resistance causes an excessive pressure drop from the liquid lump to the capillary interface. This *induced pressure drop* occurs within the capillary structure itself, and is a function of the heating rate and therefore the mass flow rate. Once deprimed, such devices usually do not reprime on their own. Figure 3-24 depicts the process used by FLUINT to determine whether or not a device is currently primed.

* Other capillary modeling elements include WICK ifaces, and USER fties with auxiliary wet wick modeling routines WETWICK, CYLWICK, and PLAWICK.

† For stability slight hysteresis (about 4%) is used in comparing pressure differences to capillary limits. Also, some smoothing (running averaging) is used to make sure the trend to deprime or reprime is clearly evident and not a result of numerical jitters.

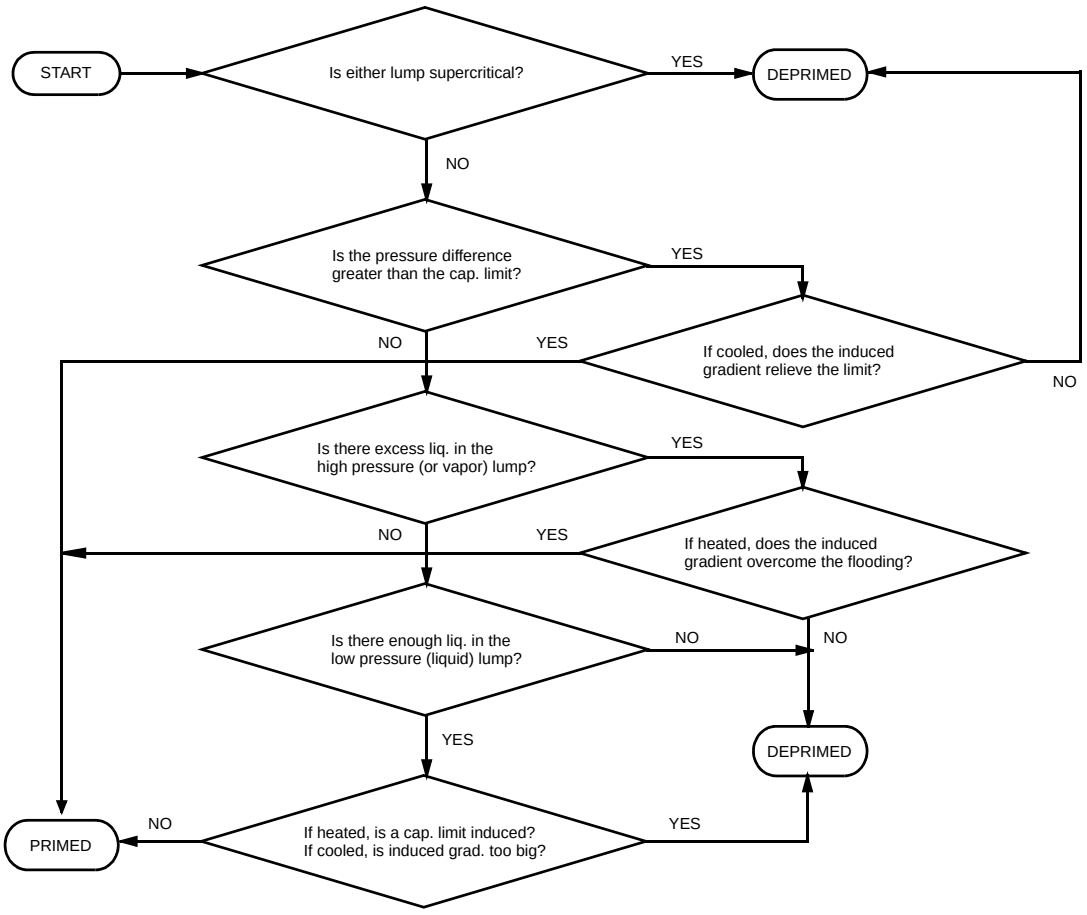


Figure 3-24 Algorithm for Determining Prime/Deprime Status

In a system with N capillary devices, there can be up to 2^N possible and valid steady-state solutions corresponding to whether or not each device is deprimed. Although *valid*, the solutions with deprimed devices are usually not *desired*. If the totally primed case were possible and all other considerations equal, the probability of arriving at that case would only be about $100/2^N$ percent. Unfortunately, other considerations are not equal. Because of the analytic fragility of the primed state and the typically rough nature of steady-state calculations, FLUINT would almost always find the case where most if not all of the capillary devices are deprimed.

Instead of posing the question: “Can these devices stay primed under these conditions?,” the user must instead *assume that they do stay primed* and then see if a valid solution is possible given that assumption. This is the purpose of the SPRIME (“stay primed”) routine described in Section 7.11.1.7. SPRIME can be called in FLOGIC 0 during steady-states to force a CAPIL or CAPPMP model to stay primed. SPRIME will make the minimum adjustments necessary in the states of the two endpoint lumps, and will report the changes necessary to keep the device primed. Adjustments made at the start of a steady-state run can be ignored. If no adjustments are made at the end of the run and the model converges, then the device *can* stay primed under the current conditions. If the model does not converge after a reasonable number of iterations *and SPRIME keeps repeating the*

same adjustments or makes increasingly greater adjustments, then the device cannot stay primed. The repeated adjustments made by SPRIME should provide a clue as to why the device didn't stay primed, and a new run without that particular SPRIME call should reveal the true deprimed steady-state solution.

The user should also take precautions to assure that capillary devices do not artificially deprime during a transient run. SPRIME can also be used for this purpose, but other methods may be more appropriate.

First, the capillary radius (RC) may be nonpositive, which signals the assumption of an ideal capillary structure that can provide or sustain an infinite pressure head. This usage can only prevent certain causes of deprime.

Second, a WICK iface (Section 3.8.3.4) can be added between the adjacent tanks to take into account the fact that the liquid control volume is far from rigid because of its proximity to a vapor control volume. The assumption of a rigid container can easily lead to pressure surges that can deprime the device. *A WICK IFACE SHOULD BE CONSIDERED MANDATORY FOR ALL CAPILLARY DEVICES MODELED USING TANKS.*

Third, the primed/deprimed decision can be affected for CAPIL connectors and CAPPMP macros by helping the program decide: how dry is dry? The remainder of this section discusses this feature.

FLUINT assumes perfect mixing and therefore even distribution of liquid within a two-phase lump. To be consistent with this assumption, any liquid in the higher pressure side (or designated vapor side) of a capillary device could deprime the device by being absorbed back into the lower pressure side (or designated liquid side). If heat is being added to a CAPPMP when it deprimes, the flow rate will drop and the junction might overheat.

Most real capillary devices may be fully primed and operational at outlet qualities less than 1.0; the liquid in the outlet is either boiled off or swept out of the outlet. Remember that a quality of 0.5 can easily correspond to a void fraction greater than 0.99; the liquid would hardly be visible and may not be a factor in the primed/deprimed decision.

Two factors are available to assist in the dryness decision in CAPIL connectors and CAPPMP macros: XVH and XVL. These factors are simply qualities above which a lump can be considered to be mostly vapor, at least in terms of a primed vs. deprimed decision.* (These factors are also applied to the liquid side (lower pressure lump) to determine whether or not it is wet enough.) A lump with quality greater than XVH is allowed to fill with liquid (for whatever reason) until the quality reaches XVL, which may cause deprime. A lump whose quality is less than XVL must reach XVH before a reprime can occur. In other words, XVH should be greater than XVL to provide some stabilizing hysteresis, but they may be equal. The default values are XVH=0.99 and XVL=0.95.

* If one end of the capillary device has been designated as a port (end location specified, Section 3.13.4), then the depth relative to the liquid/vapor surface will dominate the dryness decision in the primed/deprimed calculation.

For example, if XVH=1.0 and XVL=0.99, then the higher pressure lump would have to be initially dry (XL=1.0) to permit priming, but may drift as low as XL=0.99 before causing a deprime. Once the quality is below 0.99, the lump would have to be completely dry to allow a reprime to occur. As a rule of thumb, the user should decide which quality corresponds to “mostly dry”, and then set XVH and XVL to bracket that value by a few hundredths. If a quality of 0.8 is chosen, then XVH=0.82 and XVL=0.78 would suffice, while XVH=0.85 and XVL=0.75 would have more hysteresis if a prime/deprime oscillation is encountered. Setting both XVH and XVL to 0.8 is legal but may be susceptible to oscillations. A lower value of XVH and XVL means the device is more likely to become or stay primed. However, too low a value can produce misleading results. Recall that the dryness factors apply to the lower pressure (liquid) end as well. *A lump with a quality above XVH is considered “dry” no matter which side of the device it is on.* This means that the program may decide upon a deprimed solution on the basis of insufficient liquid. Values less than 0.5 should be used with caution, and values less than 0.1 are not recommended.

As noted above, a low value of XVH and XVL means the device is more likely to become or stay primed. When used in conjunction with RC less than or equal to zero (i.e., infinite capillary head), a low value of XVH and XVL (say 0.5) is usually a superior alternative to an SPRIME call. In both cases, the user must assume a primed solution in STEADY or STDSTL and then verify that such a solution is possible. Of course, both methods may be used together.

SPRIME may perturb a solution unnecessarily, whereas the use of zero RC and small dryness factors merely eliminates or avoids most deprime mechanisms. As noted above, continuous SPRIME adjustments are a sign of lack of a valid primed solution. Inability to prime when RC, XVH, and XVL are returned to realistic values after convergence is similarly an indication of no valid primed solution.

There are other uses for the dryness factors XVH and XVL. To illustrate, recall that FLUINT assumes one-dimensional paths. A real evaporator-pump may be partially primed, pumping in some locations, and deprimed in others. To allow for pumping while at the same time allowing liquid to flow back into the lower pressure inlet, a CAPPMP could be placed in parallel with a CAPIL connector. The XVH and XVL values of the CAPPMP would be low, signaling a tolerance to liquid in the outlet, while the XVH and XVL values of the parallel CAPIL connector would be high (say 1.0 and 0.99), allowing the device to deprime and pass liquid back to the inlet. The situation is analogous to countercurrent flow modeling in a vertical two-phase line, where again the solution is to use parallel (twinned) paths to allow liquid to fall and vapor to rise.

The above discussion touches on the occasional inadequacy of the one-dimensional and perfect mixing assumptions when applied in combination with capillary modeling. As an expansion of this topic, the user should note the difficulties that can arise when using untwinned (singlet) tanks on either end of a capillary device. Consider one tank containing subcooled liquid, and another adjacent to it containing superheated vapor at a slightly higher pressure. These two tanks are separated by a CAPIL or CAPPMP. Slight pressure perturbations might raise the liquid tank pressure higher than the vapor tank pressure. Due to the perfect mixing assumption, addition of a little cold liquid into the vapor tank would cause that tank to *cold shock* (e.g., collapse to liquid). Given the same scenario, if the pressure differential exceeds the capillary limit, warm vapor would begin to flow into the

liquid side. Again because of perfect mixing, the liquid tank pressure would rise instead of displacing liquid, and that rise would reprime the device, causing a perpetual oscillation between primed and deprimed states.

WICK ifaces (Section 3.8.3.4) avoid or alleviate many of these problems by allowing the interface to “flex” instead of allowing liquid or vapor to move prematurely from one tank to another.

Twinning endpoint tanks (see Section 3.25) avoid the assumption of perfect mixing. However, the user must be careful to attach to the vapor twin on one end and/or the liquid twin on the other end.

Detailed training notes for modeling capillary devices are available from CRTech. Refer to the contact information in the preface of this manual.

3.12 Symmetry and Duplication Options

Duplication options allow the user to identify parts of the fluid submodel that are identical and analytically repetitive. In doing so, considerable computational savings can be realized. In most real fluid systems, there are numerous examples of parallel sections that can be considered identical for analytic purposes, but whose heat transfer and pressure drop characteristics cannot be properly modeled with a single passage having larger “effective” dimensions. Examples include tube bundles or finned sections in a heat exchanger, capillary evaporator segments in a coldplate, and grooves in a heat pipe. Identical sections also can occur as a result of redundancy or symmetry in the system itself, such as parallel subsystems. FLUINT has a general mechanism for identifying such repetitions and significantly reducing both the input effort and the run times. In fact, it is strongly recommended all sections that can be *assumed* to be identical should be modeled as such to simplify the input and to speed the execution. Otherwise, the focus of the analysis may inadvertently shift toward tracking interactions between parallel passages.

Duplication options are designed to be very general to take advantage of types of symmetry and redundancy that cannot be foreseen by the authors of the program. Thus, while these options may appear to be simple, users will find it well worth their time to study them thoroughly to both maximize their usage of them and to prevent errors, which can be very difficult to detect. Duplication options are also common sources of many advanced modeling tricks.

Duplication options may be applied to paths and/or ties and/or fties, as described below. Iface duplication factors operate analogously but are described elsewhere (Section 3.8.5).

3.12.1 How Path Duplication Factors Work

Identical subnetworks and other model symmetries are specified using *duplication factors* that can be attributed to any path. If there are ten identical paths between two lumps, a single path is described and given a duplication factor of ten. Internally, FLUINT stores and analyzes only one path, but the effect of this path on the two end lumps is magnified tenfold, as illustrated in Figure

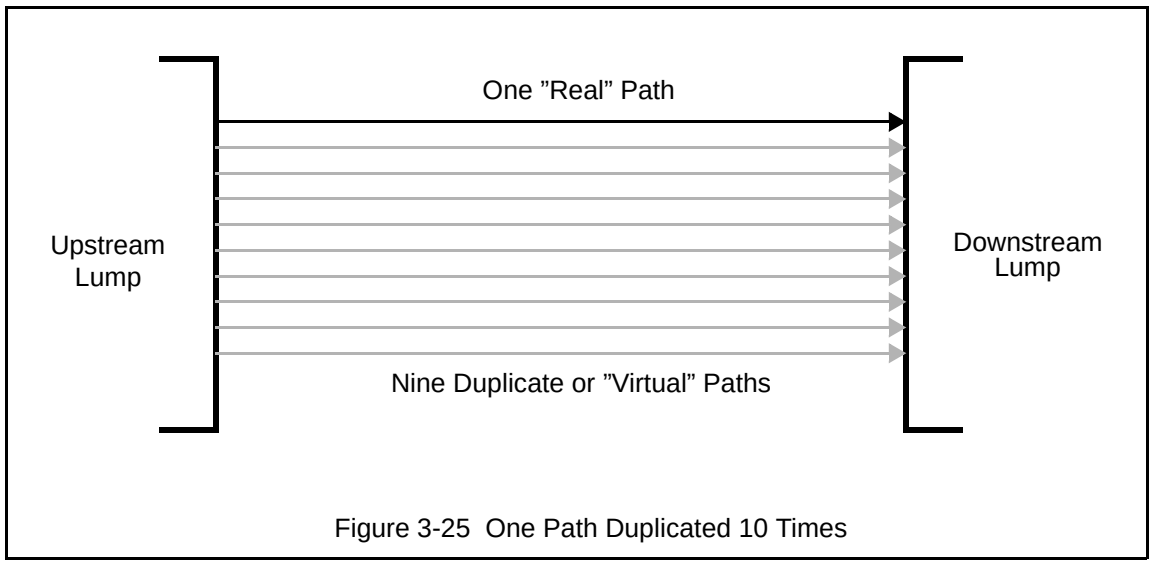


Figure 3-25 One Path Duplicated 10 Times

3-25. For example, if there were a bundle of 1000 microtubules between two control volumes, the user would describe two tanks and one path (perhaps a tube or a STUBE or CAPIL connector) with a duplication factor of 1000. The default value for all duplication factors is 1.0.

Actually, there are *two* duplication factors for every path—one for the *defined* upstream end and one for the *defined* downstream end (called DUPI and DUPJ, respectively). This allows considerable flexibility in describing unusual types of symmetry and redundancy, but using two factors instead of one may seem awkward at first. The key to understanding duplication factors is this: *the defined upstream lump “sees” a path DUPI times, while the defined downstream lump “sees” the same path DUPJ times;* DUPI and DUPJ need not be equal.

In order to comply with the configuration shown in Figure 3-25, *both* DUPI *and* DUPJ would be specified as 10.0 (input options are designed to set both factors using a single keyword ’DUP’). In many fluid systems, however, the identical sections cannot be modeled using a single duplicated path. Such a case is illustrated in Figure 3-26, where the identical portions can be an arbitrarily complex subnetwork *and can even have duplicated subsections within that subnetwork.* This example demonstrates the utility of using two duplication factors per path.

There are no restrictions on the level of “nested” duplication, and very few restrictions on the value of any duplication factor. In fact, a subnetwork (or a single path, which is simply a special case of the term *subnetwork*) does not even have to flow back into the system at all, as shown in Figure 3-27. In such a system, the duplicated or *virtual* subnetworks exist only from the point of view of the upstream lump. Although the examples in this section cover most cases of symmetry in fluid systems, the user is welcome to experiment with more complex levels of duplication. Note that the FLOMAP and PTHTAB routines print the current values of duplication factors along with other path information to help debug such complicated symmetry.

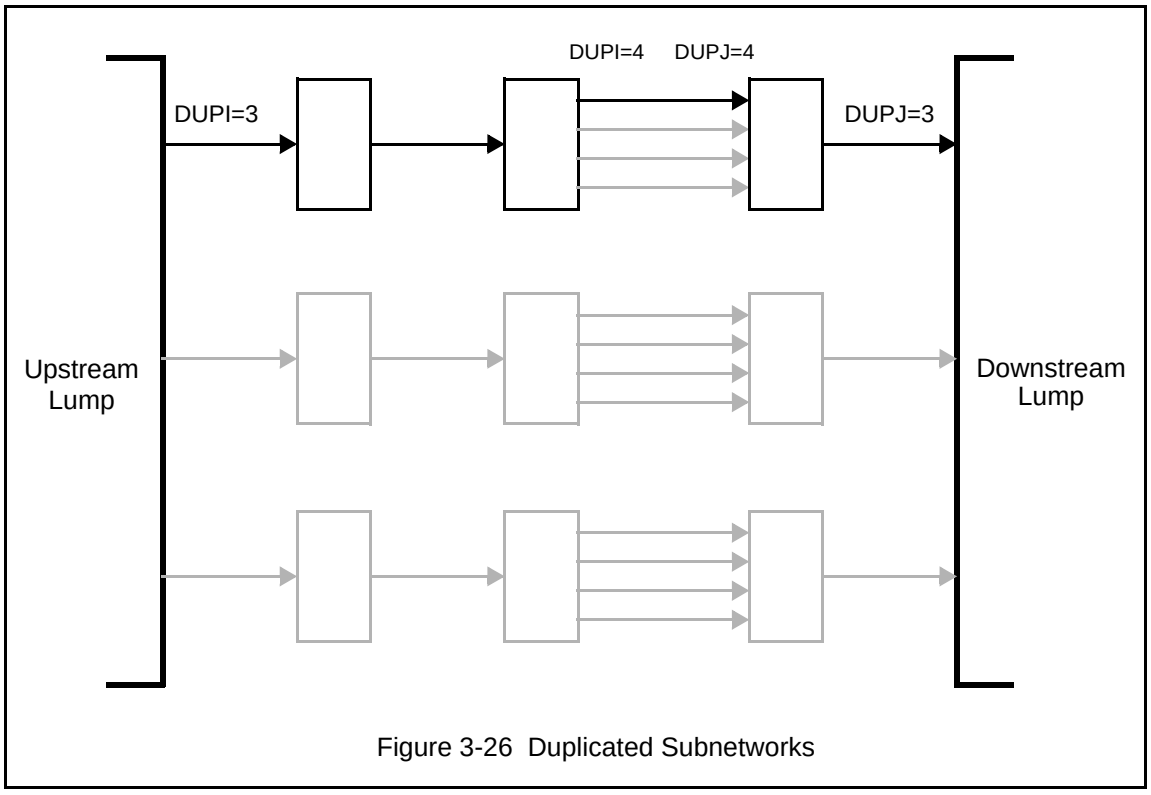


Figure 3-26 Duplicated Subnetworks

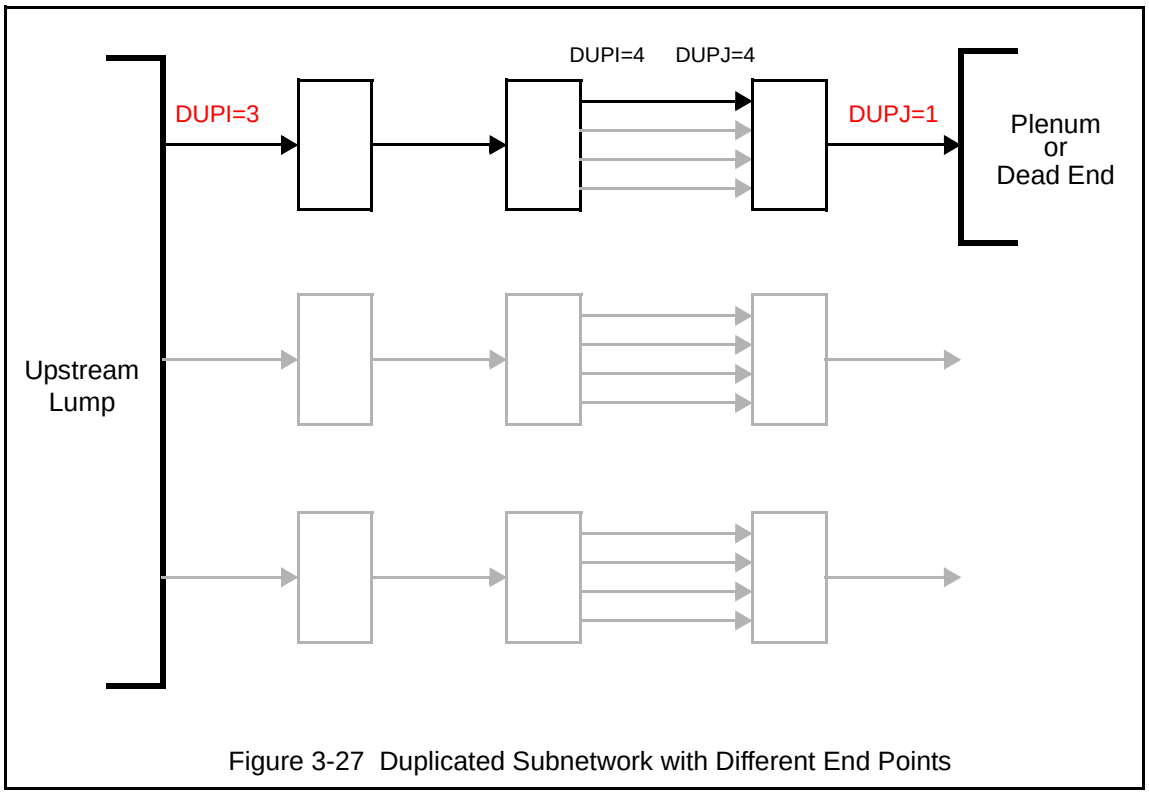


Figure 3-27 Duplicated Subnetwork with Different End Points

The user must take care to assure that if a duplicated section flows back into the main network, the total upstream and downstream duplication factors balance each other. Otherwise the virtual subnetworks will act as unintentional sources or sinks of mass and energy, which may result in illegally high or low pressures. Although not its primary purpose, the SUMDFLO routine (Section 7.11.1.10) can be used to check for an unbalanced network of path duplication factors.

For discussion purposes, the upstream and downstream lumps that start and end a duplicated section are called *manifolds*. This name is meant to stress an underlying assumption—virtual subnetworks share the same boundary conditions since they all flow out of and/or into the same lump. Thus, the FLUINT model shown in Figure 3-28a cannot make use of duplication factors unless all

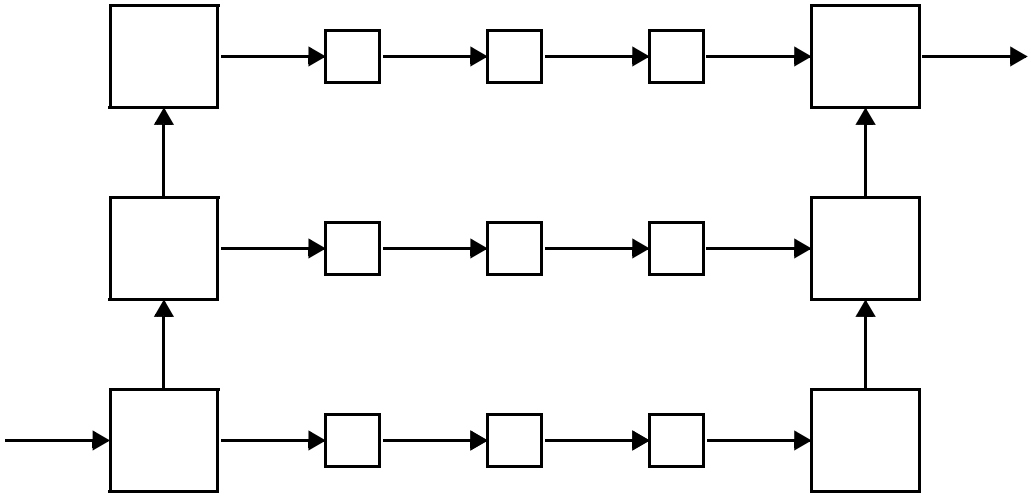


Figure 3-28(a) Subnetworks That Cannot Use Duplication Factors

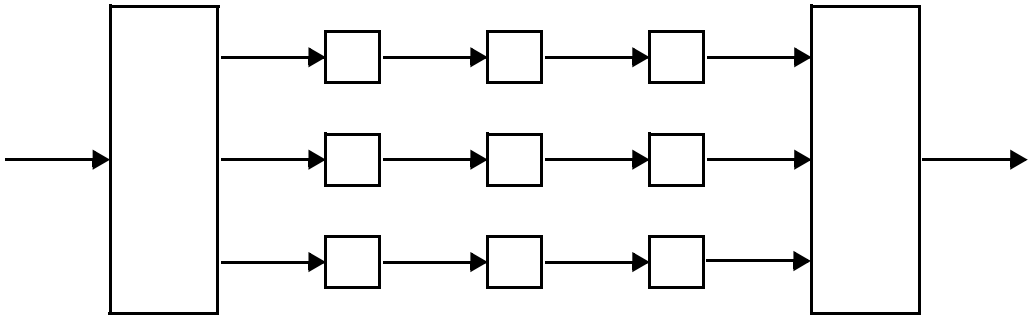


Figure 3-28(b) Above Model After Duplication-Enabling Assumption

upstream lumps can be combined into a single lump, and all downstream lumps can be similarly modeled as one lump. This assumption (Figure 3-28b) is usually equivalent to assuming a negligible

pressure gradient in the manifold, which is an excellent assumption since that is the purpose of using a manifold in a real system. The user is strongly advised to make such assumptions if possible because of the tremendous savings in time for both the computer and the user.

With respect to twinned paths, note that each twin shares the same definitions of DUPI and DUPJ, just as they share only one definition of DH, TLEN, etc. Refer only to the duplication factors of the primary twin when making changes in logic blocks.

In order to demonstrate some of the concepts discussed here, some FLOW DATA input examples are presented here. Refer to earlier sections for specific input formats. Note the use of 'DUP' to set both 'DUPI' and 'DUPJ', although no such short cut exists in logic block references. The default values are always 1.0.

EXAMPLE 1—In the following example, two paths are generated between lumps 10 and 20. The first is a tube that is duplicated 300 times between these lumps. The second is an STUBE connector that appears once from the point of view of lump 10 but twice from the point of view of lump 20:

```
PA TUBE,100,10,20,TLEN= 1.0                    $ EXAMPLE 1a
        DUP     = 300.0 ,       FR      = 22.0
C
PA CONN,200,10,20,DUPJ= 2.0                    $ EXAMPLE 1b
        DEV     = STUBE,DH      = 0.5/12.0
```

EXAMPLE 2—In this example a center-discretized LINE with 10 segments is duplicated 300 times between manifold lumps 100 and 200. In a second example, an upstream-discretized LINE command is used along with additional subblocks necessary to accomplish exactly the same result as the first example:

```
M LINE,1,C,1,1,100,200, NSEG=10                $ EXAMPLE 2a
        TLENT   = 10.0,         DHS     = 0.75/12.0
        DUP     = 300.0
        PA      = STUBE,LU      = JUNC
C
M LINE,1,U,1,2,10, NSEG=9                       $ EXAMPLE 2b
        TLENT   = 9.0,          DHS     = 0.75/12.0
        PA      = STUBE,LU      = JUNC
PA DEF, DH      = 0.75/12.0
                TLEN    = 0.5
PA CONN,1,100,1,  DUPI=300.0,DEV=STUBE
PA CONN,11,10,200,DUPJ=300.0,DEV=STUBE
LU JUNC,10
```

3.12.2 How Tie and Ftie Duplication Factors Work

Heat transfer ties and fties also use duplication factors. Nearly all of the foregoing discussion applies to tie and ftie duplication factors directly or by analogy, and hence will not be repeated here except to clarify concepts.

Tie duplication factors are DUPL and DUPN, which also default to 1.0. *The lump "sees" a tie DUPL times, while the node "sees" the same tie DUPN times;* DUPL and DUPN do not have to be the same value. Ftie factors are named DUPC and DUPD, where the c^{th} lump is the first input, and the d^{th} lump is the second input (defining the direction of positive QF: from c to d).

To avoid confusion in macrocommands, there is no input option for ties nor fties analogous to the path "DUP" keyword. Mapping routines print duplication factors and their effects, but unfortunately the TIETAB routine does not print these factors due to page width limitations.

One of the principal uses of tie duplication factors is to contain virtual networks created by DUPI and DUPJ within the same fluid submodel, without affecting the thermal submodel. For example, if a heat exchanger has 10 identical parallel passages, then path duplication factors of 10.0 would apply at the beginning and end of the passages. However, if the corresponding thermal nodes modeled the whole heat exchanger (instead of one tenth of it, i.e., one identical passage), then the DUPN factors could be set to 10.0 such that the thermal model "saw" the effects of all ten fluid sections. Other methods of dealing with symmetry in thermal models are presented below.

As another example of tie duplication factors, suppose a detailed model of a pipe and fin were generated, and the user wishes to exploit the fact that the centerline of the pipe corresponds to the plane of symmetry—the fins on either side are identical, and therefore only half of a pipe cross-section is needed for thermal boundary conditions. In this case, the fluid submodel should correspond to the entire pipe for realistic pressure drop calculations, but each tie to the thermal "half wall" can use a DUPL factor of 2.0 such that each lump in the pipe model "sees" two identical nodes (one on each side).

The primary purpose of ftie duplication factors is to match the DUPI and DUPJ factors for the paths with which they might be parallel. When using the FTIE=AXIAL command when generating a tube or STUBE connector, for example, the DUPC and DUPD of the ftie is automatically made identical to the corresponding DUPI or DUPJ of the path. Otherwise, for individually input fties it is the user's responsibility to make sure ftie duplication factors, listed in the FTTAB routine, are correct.

3.12.3 Referencing Duplication Factors in Logic Blocks

DUPI, DUPJ, DUPL, DUPN, DUPC, and DUPD may be used as variables within logic blocks in the same manner as FR, UA and other path or tie or ftie descriptors (e.g., [fsmn.]DUPIn). The user should *not* reset these values in FLOGIC 1. Also, note that "DUP" is only used as a *path* input aid and does not correspond to any path descriptor—in logic blocks, both DUPI and DUPJ must be

reset if duplication factors for both ends of the path are to be changed. For twinned paths and fties and ties, refer only to the duplication factors of the primary twin since there is only one definition for each pair.

3.12.4 Important Impacts of Duplication Factors

HTN, HTNC, and HTNS Ties—A path that is duplicated with respect to a neighboring lump is duplicated in every sense—including heat transfer characteristics. When a HTN, HTNC, or HTNS tie is made between a node and a lump, one (HTN, HTNS) or two (HTNC) paths are named to convey the characteristic dimensions, including heat transfer area. Thus, a duplication factor of 10.0 on the upstream end of a tube or STUBE connector (or to a pair of twins) will increase the heat transfer area attributed to the upstream lump by a factor of 10.0, and hence will increase the heat rate of that lump by the same factor. This is very useful in characterizing heat exchangers with many small passages such as those created by fins: an increased heat transfer area is available while the effects of the small diameter on the flow rate and the heat transfer coefficient are preserved.

There is one subtlety regarding virtual paths involved in HTN and HTNC ties. There are currently no restrictions on the location of a path involved in such a tie—unlike HTNS ties, it does not have to be adjacent to the lump through which heat is transferred. Because of this freedom the user should remember that *magnification of the heat rate of a tie will only occur if the lump and path involved in the tie are adjacent.*

For HTNS ties, another potential problem exists if flow rates reverse and unequal (asymmetric) path duplication factors are used (i.e., DUPI ≠ DUPJ). Since these duplication factors affect the amount of heat transfer area ‘seen’ by the lump, the value of UA may change by the ratio DUPI/DUPJ for HTNS ties if flows reverse and the “STAY” command has not been issued. A warning is produced at the start of the processor if it detects the potential for such a condition given the *initial* values of DUPI and DUPJ and the absence of the “STAY” command, but no further actions or checks are performed thereafter.

Thermal Submodels—There are no corresponding duplication factors in thermal submodels. As described below, *imbalances in the energy flow of a thermal submodel will result if it is tied to a duplicated fluid subnetwork but is not isolated or tie duplication factors are not used.* (Imbalances are allowed and in fact are common in fluid submodels, but can slow steady-state convergence and can cause confusion.) As an alternative to duplication factors, dual one-way conductors (either linear or radiation) can be used to emulate thermal duplication factors. Thus, the thermal model of a fluid container can be made to mimic the way that redundancies and symmetries are exploited in the fluid model.

To demonstrate this technique, Figure 3-29 shows how to make one node appear duplicated three times with respect to a second node, while the second node would seem to appear only once with respect to the first node. This is analogous to the DUPI=3.0 part of the major subnetwork in Figure 3-26, and could be used to model the pipe wall across this “duplication discontinuity.”

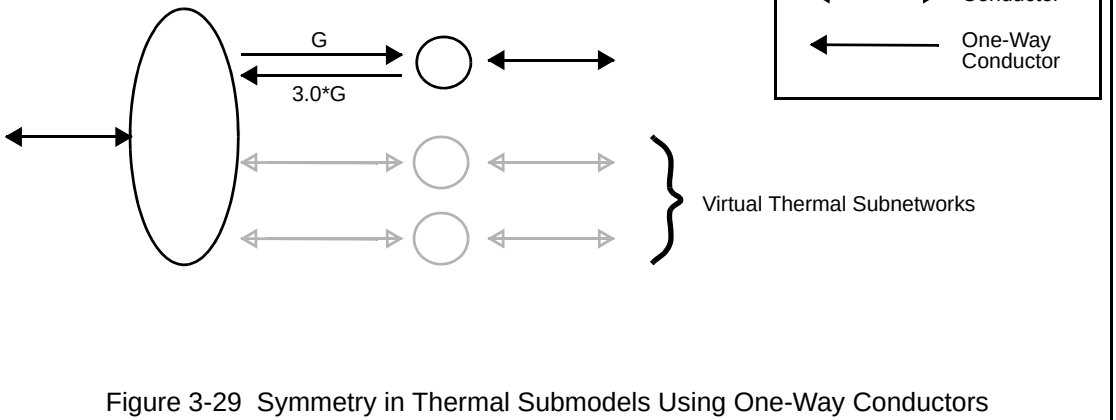


Figure 3-29 Symmetry in Thermal Submodels Using One-Way Conductors

In fact, the SINDA/FLUINT thermal user may find such modeling techniques can significantly reduce the size of purely thermal models. The thermal user should also pay attention to the next subsection dealing with modeling tricks, since many of these tricks also apply by analogy to thermal-only models using dual one-way conductors. Refer to Section 2.14 for a discussion of one-way conductors, and refer to Sample Problems A, E, and G for a demonstration of such modeling practices.

Energy Balances—The use of dual one-way conductors and virtual, diabatic fluid subnetworks may affect the overall steady-state energy balance check: the submodel can be considered converged even though the system-level energy flows do not balance. The imbalance is created by the fact that duplicated sections can appear to the full network as sources or sinks of mass and energy. (In mathematical parlance, the coefficient matrix is asymmetric.) This fact is recognized in both thermal and fluid energy balance checks, and may result in a message such as “ENERGY STABLE BUT NOT BALANCED.”

This message should not be construed as a warning that the results are erroneous or inaccurate. Rather, it is intended to warn the user to make sure the final answers are consistent with the use of the duplication factors and/or one-way conductors. However, in thermal submodels another cause of this message exists which *does* indicate inaccuracy: large conductors and small temperature differences can cause steady state solutions to stall, and this lack of progress may be mistaken for asymmetric networks. If such is the case, the user should either reduce the convergence criteria and continue, or correct the large conductors if they are in error, or eliminate the large conductors by combining adjacent nodes, or use the matrix methods (nonzero MATMET, which is the default).

When in doubt, consider using the lump-by-lump energy checks (negative REBALF) and node-by-node energy checks (EBALNA). While they are usually less preferable than system-level checks, these element-level checks are not sensitive to one-way conductors or duplication factors.

Checking for a Balanced Network—Path duplication factors need not "balance" in open or quasi-open systems (see for example Sample Problem A), but they should balance within closed systems such that energy and mass are not created (see for example Sample Problems E and G). One method of checking the network is to use the SUMDFLO routine (Section 7.11.1.10), which errs off if it encounters an unbalanced network.

3.12.5 Modeling Tricks Using Zero and Noninteger Duplication Factors

In most models, duplication factors will be integral (although they are REAL in Fortran). This section describes the implications of using zero and noninteger duplication factors. Negative factors should be avoided.

Zero Duplication Factors—A zero duplication factor may be used to make a path or tie or ftie or iface "disappear" from the point of view of an adjacent lump or node. From the point of view of a path with zero duplication factor, the relevant lump will still appear, but changes in the path will not affect the lump. Note that this does not reduce execution time since *the "zeroed" subnetwork does not itself disappear, merely its effects on the rest of the network*. Zero path factors may be used to simulate a valve closing next to the lump without causing undue pressure/flow rate changes in the zeroed subnetwork (through which flow continues), or to dynamically add, delete, or exchange subnetworks. Zero tie factors may be used to shut off heat transfer through that tie,* perhaps to turn on another parallel or nearby tie. Using 0.0 on one end of a path or tie and 1.0 on the other end is analogous to a one-way conductor in thermal models. Clearly, zero duplication factors are a powerful tool and will certainly provide a mechanism for other modeling tricks.

Note that tanks and junctions must have at least one active path adjacent to them.

Noninteger Duplication Factors—Fractional factors (such as 0.1 and 2.5) can also be useful in certain modeling instances. To a lump, a fractional path has the normal diameter but carries DUP times the flow rate and has DUP times the heat transfer area (where DUP is either DUPI or DUPJ). One possible use of such factors is to divide a fluid component in half or some other fraction. For example, five parallel pipes between two 1.0 ft^2 manifolds could be conceivably be modeled as 2.5 parallel pipes between two 0.5 ft^2 manifolds.

Fractional duplication factors may be used to replace integral factors. For example, a path with DUPI=6.0 and DUPJ=1.0 may be replaced with a path that has DUPI=1.0 and DUPJ=1.0/6.0. The only difference would be that the first path carries one sixth of the flow rate of the second path (i.e., appears six times rather than once). This trick is useful when upstream- or downstream-discretized duct macros (opt= U or opt = D) are duplicated. In those cases, the duct starts or ends with a lump that appears duplicated with respect to the rest of the network. A fractional value on a path adjacent to such lumps can be used to combine or split the flow appropriately without introducing an unnecessary path. When using the Solver (Section 5) to select the number of parallel passages, noninteger path duplication factors can be used as design variables, then rounded to the nearest sensible value.

* If both DUPL and DUPN are zero for a tie, the tie heat transfer calculations for that tie are not performed: the QTIE and UA values are not updated. This action is intended to eliminate property range warnings (e.g., "TEMPERATURE TOO LOW") when ties are disabled. Path calculations, on the other hand, are always performed.

Fractional tie duplication factors are useful because they allow dynamic variations to HTN and HTNC ties—they can be used as multiplication factors to augment or degrade heat transfer calculations, perhaps to add fouling or to parametrically investigate sensitivity to heat transfer coefficients. Because area fractions for these ties cannot be changed during the processor execution (i.e., can only be set in FLOW DATA), tie duplication factors provide an alternative for dynamically varying the heat transfer area on any tie. When using the Solver (Section 5) to correlate heat transfer to test data, noninteger tie duplication factors can be used as design variables. (UAM is perhaps a better choice, nonetheless.)

3.13 Gravity and Acceleration Body Forces

The user may impose body forces due to gravity or accelerations on all active fluid submodels. This option is exercised by attributing an elevation to any lump in the network and an acceleration (such as “g”) in that direction. More generally, the location of any lump may be given in three Cartesian coordinates, and the acceleration may be described as three components of a vector. While one dimension is sufficient for ground-based systems, three dimensions are available to allow simulations of launch and orbital maneuvering using the same model.

Flow regime mapping (IPDC=6) and slip flow modeling with twin paths are both affected by the direction and magnitude of body force terms, as described in later sections.

The LMXTAB output routine is useful for tabulating lump coordinate locations and listing the current components of the acceleration vector.

Note that each HTP tie has its own “local” gravity (ACCM, perhaps affected by distance from a spin axis) and location (DEPTH relative to a liquid surface). Those terms affect boiling suppression and help twinned tanks determine the liquid level at the wall (Section 3.6.3.3). They are intentionally independent from the global options described in this section.

3.13.1 Definitions

The body force can be thought of as a superimposed pressure drop or gain on each path, similar to the HC factor that the user may apply to tubes and STUBE connectors. This additional pressure force is a function of the effective path density and the dot product of the acceleration vector and the *path vector*, where the path vector is the vector pointing from the defined upstream lump to the defined downstream lump:

$$\Delta P_a = \rho_{eff} \left(a_x (x_i - x_j) + a_y (y_i - y_j) + a_z (z_i - z_j) \right) F$$

where:

ΔP_a . . . . . Superimposed acceleration pressure force, i to j direction
ρ_{eff} . . . . . Effective path density
a_x, a_y, a_z . . . . . Acceleration vector components
x_i, y_i, z_i . . . . . Coordinates of defined upstream (i^{th}) lump
x_j, y_j, z_j . . . . . Coordinates of defined downstream (j^{th}) lump
F . . . . . Unit conversion factor for English units

The coordinate locations of each lump are input by the user as CX, CY, and CZ. These variables have units of length (ft or m), and are defaulted to zero—no elevation differences between lumps. Note that these coordinate locations are used for body force calculations only. There is no relation

between lump coordinates and tube or STUBE connector lengths. For flexible modeling, paths may be longer or shorter than the distance indicated by endpoint coordinate locations. Furthermore, as evidenced by the default values, any number of lumps may occupy the same location in three-space.

This section describes the program actions when the path end location is the same as the lump location. See Section 3.13.4 for an expanded, alternate treatment for automating analysis of large* two-phase tanks such as dewars (cryostats), fuel tanks, and two-phase reservoirs.

A lump is defined in this manual as having a single thermodynamic state. Actually, that definition should be generalized to include *a single (though not necessarily constant) coordinate location.* Thus, to simulate the body force felt by a piping penetration submerged in a vertically stratified tank, the coordinate location of that tank might be input as that of its surface, with the elevation of the penetration point being specified according to its depth. If the surface level changes, the user must update the coordinate location accordingly. Similarly, specifying more than one distinct elevation or coordinate position will require more than one lump, even if the thermodynamic state (e.g., temperature and pressure) is the same. Sample Problem B uses three separate plena with identical states for just such a purpose.

The components of the acceleration vector, ACCELX, ACCELY, and ACCELZ, are *global* control constants like ABSZRO and therefore apply to all submodels. To facilitate modeling, *the ACCEL vector is taken to be in the opposite direction of the force on the fluid*, consistent with the above equation. Therefore, if the CZ array represented the *elevation* of lumps from some arbitrary reference point, then use ACCELZ = +g to simulate gravity. For body forces due to acceleration, remember that an acceleration in one direction yields a net force on the fluid in the opposite direction; the ACCELX/Y/Z vector should be input as the direction in which the fluid container is accelerating.

Acceleration vector components have units of either m/s^2 or ft/hr^2. To be consistent with the conventions dictated in Appendix D, ***the English units for acceleration are ft/hr^2 and not the more familiar ft/s^2***. In these units, the acceleration of gravity is equal to $32.174*3600^2 = 4.17E8$ ft/hr^2. In SI units, the acceleration of gravity is 9.81 m/s^2. The built-in constants “grav” and “gravsi” contain these respective values for convenience.

Recall that no thermodynamic state can be directly attributed to a path. The effective density of each path is calculated on the basis of the densities of the lumps on either end of the path, and includes the effects of any phase suction options. For homogeneous tubes, the factor UPF decides the weighted fraction for this density as it does for the frictional pressure drop correlations. In *all* connectors, the effective density is always the average of the lump densities—any other weighted average can cause numerical instabilities for two-phase flow. This means that UPF is ignored in homogeneous STUBE connector calculations for body forces, constituting the only deviation in meaning between tube and STUBE factors. If twin paths are used and if they are not currently homogeneous, then the density is reasonably constant for each phase. However, for stability, the uphill void fraction is used when calculating body force terms on each phase.

* “Large” in this case can be taken to mean that the volume is significant compared to the flow area of the path such that the position of the liquid/vapor surface is important to the path’s behavior.

The use of average density for homogeneous STUBEs can have other modeling implications. To illustrate these implications, assume that an STUBE carrying liquid from an upstream lump flows into a plena containing gas or two-phase fluid. Even though the STUBE "carries" liquid, the effective density is somewhat less than pure liquid due to the averaging of the densities. Thus, the body force on that STUBE is less than might be expected. To overcome such cases where the user wishes to use only the in-flowing state, consider using path end locations ("ports", see Section 3.13.4).

Caution: If a higher density or liquid-containing lump is placed above a lower density or gas-containing lump (relative to the ACCEL vector), flow rate oscillations and reversals are likely. In a steady state, such situations can lead to nonconvergence. In transients, flow reversals can cause artificial energy errors and exaggerated mixing between the lumps. Consider Ports (Section 3.13.4), tubes (Section 3.4.1), and flat-front ducts (Section 3.27) as enhancing or alternative modeling methods.

3.13.2 Referencing Body Force Options in Logic Blocks

CX, CY, and CZ may be used as variables within logic blocks in the same manner as TL and other lump descriptors (e.g., [fsmn.]CXn).

The coordinate location of tanks that are used within a FloCAD Compartment is adjusted automatically.

ACCELX, ACCELY, and ACCELZ may be used as variables within logic blocks in the same manner as ABSZRO and other global control constants (e.g., simply ACCELX).

The user should *not* reset these values in FLOGIC 1 or in VARIABLES blocks.

3.13.3 Examples

This section contains simple examples of body forces due to both gravity and general accelerations. Refer to Section 3.3 and Section 4.3.3 for input formats. To summarize the rules previously described, either use a positive value of gravity and elevations for lump coordinates, or input ACCELX/Y/Z *as the direction in the (x,y,z) frame in which the whole system is accelerating.*

3.13.3.1 Gravity

In the following example, a vertical line is built whose positive flow direction is upwards. The line is 10 meters long, and is broken into 10 segments built using junctions and STUBE connectors. Note that lump 1 (not shown) has an elevation of 0.0:

```
HEADER CONSTANTS DATA, GLOBAL
        ACCELZ  = 9.81
        .
        .
        .
HEADER FLOW DATA ...
        .
        .
        .
M LINE,1,D,10,10,1
        DHS=0.01,TLENT=10.0,   NSEG=10
        LU=JUNC,PA=STUBE,  CZ=1.0,CZINC=1.0
        .
        .
        .
```

3.13.3.2 General Acceleration

In the following example, a loop is built in the shape of a perfect square with sides equal to 10 feet (see Figure 3-30). The first leg extends 10 feet from the origin in the X direction, and is modeled by a single tube. The second leg bisects the positive Y and Z axes, and is modeled with a single STUBE connector. The third leg returns to the Y-Z plane, and is modeled with a center-discretized LINE using 10 segments. The last leg returns to the origin, and is modeled with a MFRSET pump (recall that there is no relationship between path length and the difference between lump coordinates).

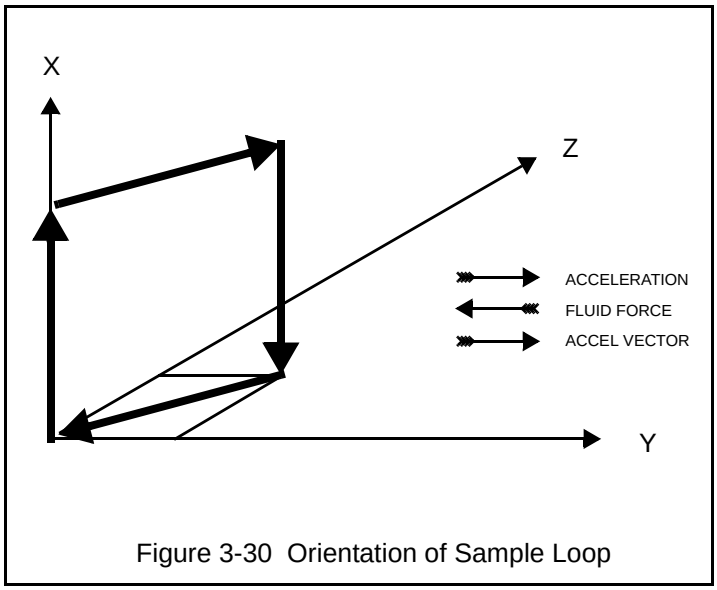


Figure 3-30 Orientation of Sample Loop

Initially, the spacecraft containing the loop is not accelerating. At time zero, the spacecraft fires a rocket and accelerates at 0.01g in the Y direction for 1 minute. Note that a vehicle acceleration in the Y direction results in a force on the fluid in the negative Y direction. ACCELY is the vector representing the vehicle acceleration in the lump coordinate system. A partial list of inputs is provided below:

```
HEADER CONSTANTS DATA, GLOBAL
        ACCELY  = 0.0
        .
        .
        .
HEADER OPERATIONS DATA
        .
        .
        .
        CALL STEADY
        ACCELY  = 0.01*32.2^3600.0            $ 0.01G
        TIMEN   = 1.0/60.0                    $ ONE MIN
        CALL TRANSIENT

HEADER FLOW DATA ...
        .
        .
        .
LU JUNC,1              $ CORNER AT ORIGIN (DEFAULT)
PA TUBE,1,1,2
LU DEF,CX=10.0
LU JUNC,2              $ CORNER (10,0,0)
PA CONN,2,2,3,DEV=STUBE,TLEN=10.0
LU DEF,  CZ= 10.0/1.414
        CY      = 10.0/1.414
LU JUNC,3                              $ CORNER (10,7.07,7.07)
M LINE,1,C,30,30,3,4 DHS=0.01, TLENT=10.0, NSEG=10
        LU=JUNC,PA=STUBE,
        CX=9.5, CXINC=-1.0             $ NOTE INIT. CX
LU JUNC,4,CX=0.0                       $ CORNER (0,7.07,7.07)
PA CONN,4,4,1,DEV=MFRSET,SMFR=100.0
        .
        .
        .
```

3.13.4 Path End Locations ("Ports")

By default, each path is assumed to start at the coordinate location of the defined upstream (i^{th}) lump (as defined by its CX, CY, and CZ positions), and terminate at the coordinate location of the defined downstream (j^{th}) lump. These locations are used for body force terms and flow regime determination, as described above and in Section 3.16.

However, when modeling large two-phase vessels (e.g., a fuel tank, dewar, boiler, or liquid reservoir) in a gravity field or with vehicle accelerations, the lump at the end of the path is often no longer tightly associated with the path: its void fraction or density may no longer be representative of what is flowing through the path. In such cases, the changes associated with one or both ends of the path being either submerged (wet) or exposed to ullage (dry) need to be taken into account.

For a large two-phase vessel modeled as a single (homogeneous) or twinned (phasic nonequilibrium) tank, a single CX/CY/CZ location is still expected to characterize this vessel, even though it obviously cannot be represented by a single point in three-space. To make sure that body force and other calculations are correct, the location of the tank needs to be chosen to be any point on the liquid/vapor surface, which is where the pressure of that tank can be thought of as being defined.

Since a SINDA/FLUINT model does not include details of the shape of the tank wall or any structures contained within it, the above statement implies a requirement on the user to update (or provide a self-updating defining expression) for the CX/CY/CZ location of the lump that represents a point on the (presumed flat) liquid/vapor surface.*

For example, in a vertical cylindrical tank with a flat bottom surface, the CZ (elevation) of the liquid/vapor surface might be defined as:

```
CZ = (1-AL#this)*VOL#this/A_sect + CZ_bottom
```

for a homogeneous tank, where *A_sect* is the cross sectional area and *CZ_bottom* is the elevation of the bottom of the tank. For a twinned tank,† use the following to remove "1-AL" from the equation (unless the secondary volume is zero, in which case the tank is currently in mixed mode):

```
CZ = ((VOL#twin==0)?1-AL#this:1)*VOL#this/A_sect + CZ_bottom
```

This twin formula assumes that if there is void within the liquid side, it has raised the liquid/vapor level.

Of course, the above example is intentionally simplistic. For a more realistically complex tank, including one that is tilted or is horizontal and has complex end caps, the expressions for the height of the liquid can become more complex. See TankCalc60 in Section 7 for help in such cases. FloCAD Compartments are strongly recommended for vessels that are more complex, and/or which tilt during a transient simulation. See the Thermal Desktop User's Manual for more details on Compartments.

* Tanks within a flat-front model (Section 3.27) are an exception. The program automatically calculates the liquid/vapor front within the active segment. Tanks that are part of a FloCAD Compartment are another exception: that program sets up logic which updates the coordinate location of such lumps automatically.

† See Section 3.25.

To be consistent with this usage, if the vessel is 100% full of liquid, the "location" of the tank used to model this vessel should be placed at the top of the vessel. Similarly, if the vessel is completely empty, the CX/CY/CZ location should be at the bottommost point.*

For a two-phase vessel, it can make a tremendous difference to the prediction of an attached path's behavior to know whether it is flowing into the vessel (or pulling fluid from the vessel) at a point that is either above or below the liquid/vapor surface. Path end locations may therefore be overridden using the values CXI, CYI, and CZI (for the i^{th} end) and/or CXJ, CYJ, and CZJ (for the j^{th} end). Using these coordinates, the user is defining the location of the "port" on the wall of the two-phase vessel.

As with lump locations, when working with gravity, only a single axis needs to be used. Commonly, this is the Z axis where ACCELZ has been set to the acceleration of gravity (*grav* or *gravsi*). In this case, the "coordinate location of the path's end" degenerates to using either just CZI or just CZJ, depending on the direction of positive flow rate. CZ of the lump is the *current* elevation of the liquid/vapor surface, and CZI (or CZJ) is the (usually constant) elevation of the centerline of the path at the point in connects to the tank.

For example, consider a vessel whose bottom is at an elevation of 30 meters, and whose top is at 60 meters. Twenty meters above the bottom of the vessel (at an elevation of 50 meters), the centerline of a 4 cm diameter tube or "pipe penetration" flows into the vessel (at least when FR is positive). The CZJ for this tube is fixed at 50.0 meters (CZJ=50.0), and an expression for the CZ of the tank would need to be developed to vary the elevation of the liquid surface as a function of the liquid volume contained within it, varying in the range $30.0 < CZ < 60.0$.

When a path end position has been defined (meaning when *any* of the above six end location parameters have been input, rather than defaulted), and if the current depth is nonzero,† FLUINT invokes a variety of automatic (internal) simulation manipulations, as described below.

First, the phase suction options will be automatically manipulated‡ as a function of whether the path end is above or below the liquid surface. The depth of each path end is calculated by the program and reported by the BFI and BFJ output parameters, with negative "depth" signifying height above the liquid/vapor surface. If the untwinned path** is connected to an active (split mode) twinned tank, the end of the path will be automatically moved to the tank representing the correct phase.

---

* With this definition, path port locations may also be applied to single-phase tanks: no attempt is made to restrict their usage to two-phase tanks.

† Depth is calculated as BFI for the i^{th} end, and BFJ for the j^{th} end, as explained below. Note that the UPF update is performed even if BFI/BFJ is zero (perhaps because the ACCEL vector is zero). However, the phase suction changes may not be effected if BFI or BFJ is exactly zero due to hysteresis as explained below. Consider setting the initial phase suction appropriately, instead of relying entirely on the automatic operation of ports.

‡ Capillary devices (CAPIL connectors, CAPPMP macros) are an exception, though the port location relative to the liquid/vapor surface is taken into account in the wet/dry primed/deprimed decisions.

** End positions on active (slipping) twinned paths are ignored. When the end location is ignored, the endpoint tank is connected to the twinned path as if that tank represented the end condition on that path.

To avoid numerical instabilities, a hysteresis is applied to this switch.* If a submerged port is gradually exposed to vapor or gas ("ullage"), the switch from liquid to vapor/gas will not happen until the path's flow area† is completely dry. If the port begins again to submerge later, the switch back to liquid will not happen until the flow area is completely submerged.

To continue the above example, the 4cm pipe would be submerged for a liquid surface elevation above 50.02 meters (50+0.5*DH), and exposed to ullage below 49.98 meters (50-0.5*DH). In between these two values, no change is made: whatever phase the pipe was previously exposed to, it would continue to "see."

Second, the UPF of a tube or STUBE connector may be changed to 0.5 or 1.0 as necessary to take into account the fact that the tank state is not representative of fluid within the path. For example, if a path is currently flowing into a tank and the exhaust end is an active port (i.e., path location distinct from lump location *even if the depth is zero*), the UPF will be set to 1.0 such that the currently downstream tank state is not used in frictional pressure drop calculations. (A flow reversal in this example might result in the UPF being reset to the original default of UPF=0.5.)

Third, if a fluid entrance into a path is submerged, and if the tank is stagnant (LSTAT=STAG, which should normally be true for a large tank whose thermodynamic state is not a function of nearby path velocities), then the expansion calculations applied to choking (Section 3.18.1.3) may be modified slightly. Such modifications take into account the fact that the upstream pressure is slightly higher than that of the upstream tank. They also take into account the fact that the in-flowing enthalpy is slightly lower than that of the tank's liquid phase, due to loss of potential energy. These are both normally very minor corrections. In fact, they often do not make any difference in results.

Fourth, and perhaps most importantly, the body force calculations described in Section 3.13.1 are expanded to take into account the difference between what is flowing into and through the path, versus the phase that the path "sees" at its exhaust into a larger two-phase vessel.

For example, if a path containing gas or two-phase flow is flowing into a tank *below* the liquid/vapor plane ("submerged exit"), a larger head term will be applied at the exit since the density of only the liquid phase is applicable. The lower density in the path is only applicable at the entrance and/or length of the path itself, but does not affect the local pressure felt by the exit of the path.

Similarly, if a path that is flowing liquid or two-phase fluid flows into a tank *above* the liquid/vapor plane ("exposed exit"), there is no falling column of high density fluid "pulling" on this path. Instead, the liquid is cascading into that exit tank, and the head term at the exit can be almost disregarded, as if the body force terminated at the path exit and resumed after the liquid hits the surface.

---

* For CAPILs and CAPPMPs, phase suction options don't change, and no hysteresis can be applied to the primed/deprimed decision if the CAPIL or CAPPMP is attached to a plenum or homogeneous (untwinned or mixed mode) tank. Twinned tanks are strongly recommended with ports on CAPIL or CAPPMP paths.

† Since diameter is not required for most connector devices, the maximum of 0.5*DH and the square root of AF/π (or AFI/π or AFJ/π) is compared to BFI or BFJ. This means that if AF has not been defined for some path (e.g., an MFRSET or VFRSET connector), no hysteresis can be applied and the liquid/vapor designation could conceivably switch every iteration or time step. For ORIFICEs, AORI is used instead of AF to determine this hysteresis band, as described in Section 3.13.4.1.

These port calculations are disabled or ignored (perhaps temporarily) in certain circumstances, as noted below:

1. An active twinned tube (or STUBE connector) has been attached to a lump using an end location. In this case, the end location is temporarily disabled until the path returns to the homogeneous mode. Disabling the port means that the endpoint lump is assumed to represent the fluid flowing into or out of the actively slipping tube or STUBE pair.

   Note that flat-front paths (Section 3.27) are twinned, but are not active (i.e., they stay homogeneous) unless DUCT_MODE has been used to activate slip flow away from the front.

2. A flat-front relationship exists (Section 3.27) between a tube and its tank, such as the tank on the defined downstream (j^{th}) end of an FF=FILL tube, or the tank on the defined upstream (i^{th}) end of an FF=PURGE tube. Such a path cannot have ports defined on the paired tank.

   Note that this does not preclude a flat-front model of PURGE model from exhausting *into* a port, nor a flat-front FILL model starting *from* a submerged port. Nor does it preclude other paths from having "side" ports on a tank that is part of a flat front pair. See Section 3.27.5.4 for more details.

3. The path end locations have not originally been defined in FLOW DATA. **If the end of a path has been originally defaulted to the location of the end-point lump, future manipulations of the paths CXI/CYI/CZI or CXJ/CYJ/CZJ in logic will be ignored.**

This last item deserves added attention. If port calculations are to be applied in a model using *only* user logic (e.g., OPERATIONS or FLOGIC 0), the user should be sure to set at least one of the coordinates of the relevant path end to *any* value (even if zero, since it will be overwritten in logic). Otherwise, the program is not able to detect what is changed within user logic, and by default assumes port calculations to be inactive.

Otherwise, once the program has been "notified" that a port is potentially active by the presence of an input end point location in FLOW DATA (e.g., "CZI=..." for the i^{th} end, or "CXJ = ..." for the j^{th} end), then the lump locations and path end locations can be adjusted within logic as needed.

For example, consider a flexible hose within a tank. The hose has a weight on the end, such that this end of the path is submerged even if the tank is inverted. Login in FLOGIC 0 that handles such a case might appear as follows, assuming that the i^{th} end of path 104 can toggle between an elevation of 10 and 0 in the Z direction:

```
if(ACCELZ .gt. 0.0)then
        CZI104 = 0.0 $ move inlet to the current bottom
else
        CZI104 = 10.0 $ the bottom is on the other end!
endif
```

This is the gist of the caution in the third item above: This logic will be ignored unless the user *also* specified CZI=0.0* in FLOW DATA for path 104, signaling that the i^{th} end of path 104 will require port calculations. Otherwise the program would continually reset CZI to be the current CZ of the lump on the defined upstream end of that path, and no port calculations will ever be performed.

Even if "CZI=CZ#up" is set for the path in FLOW DATA, such that there is no difference in elevations, this is still a signal to the program: port calculations will now become active at the defined inlet. Of course, most port calculations will have no effect in this specific case (of equal elevations), excepting the setting of UPF=1.0 and hysteresis in the phase suction and tank twin selection.

This caution about FLOW DATA inputs being a signal can be disregarded if the path end locations have been defined via expressions, but not if they have been defined only using Sinaps or FloCAD Network Element Logic (which is a form of user logic, not an expression).

The PORTTAB output routine is useful for tabulating path end locations relative to the liquid/vapor position defined by the endpoint lump.

NOTE: A FloCAD (page xliv) *Port* defines path end locations (e.g, CXI/CYI/CZI) relative to a lump (including to a tank representing a Compartment). Compartments automatically calculate liquid/vapor plane positions in complicated vessel geometries, and assign wet/dry fractions and other modifications to associated heat transfer ties. See the Thermal Desktop User's Manual for more details on Compartments and Ports.

3.13.4.1 Cautions and Guidance: Ports

Junctions, Plena, and Steady States—*Ports are primarily intended to be used with two-phase tanks within transient solutions.* Using ports with junctions (and tanks during the STEADY solution) is not illegal but can easily be problematic. Use with plena is legal, but the quality XL (or void fraction AL) *and* the location of such a plenum should be chosen carefully.

When a path end location is specified, it is assumed that the coordinate location of the lump represents the liquid/vapor plane. Since the quality and void fraction in a junction (or tank during STEADY solutions) can change instantaneously, the liquid/vapor plane should move instantaneously as well. This conflicts with the restriction that CX, CY, and CZ must be assumed to be constant during each solution interval (time step or STEADY iteration) along with phase suction settings.

It is unlikely that a model will be developed that intentionally uses a port on a junction. However, it *is* easy to accidentally "forget" that a STEADY solution treats tanks as junctions, and few transients begin without a STEADY solution to prepare consistent initial conditions. Refer to the utilities HLDLMP and HLDTANKS in Section 7 for temporarily treating tanks as plena during steady state solutions.

* They could of course specify any value here, such as CZI=3.1416. Or they could specify CXI or CYI, even if ACCELX=ACCELY=0.0. Any of these actions signal to the program that the i^{th} end of the path is not defaulted to be the same as the lump's location, and therefore port calculations should be performed.

"PORT TEMPORARILY DISABLED"—If a one end of a path is positioned below (relative to the current ACCEL vector) a lump that contains only gas (XL=AL=1.0), the port on that end will be ignored until either the relative coordinates change or the lump gains some liquid (XL<1.0, AL<1.0). Otherwise "submerged when there is no liquid" does not make sense.

Similarly, "exposed when there is no gas or vapor" does not make sense: a port will be disabled if it is above a lump that contains only liquid (XL=AL=0.0).

This warning message usually means that the path end location has been located incorrectly, or that the logic or expression that adjusts the lump coordinate location is defective. It can also occur if a port has been placed on a plenum whose thermodynamic state and location were not carefully specified.

Flow Reversals—Use of a port on a path signals that the density within the path is distinct from that of the downstream lump. This could mean that the body force on the path will be completely different if the flow were to reverse. A solution may not exist: forward flow causes a retarding force that reverses flow, and reversed flow causes a restoring force that returns to forward flow. In such situations, the flow rate will oscillate excessively, causing nonconvergence, energy errors, and artificially large mixing between the endpoint lumps.

These issues are compounded by paths with low flow resistance between two-phase tanks. Pressure solution accuracy in such tanks is not perfect. While the computational noise in such a solution is normally invisible, but could become dominant if the flow resistance of an attached path is very small. For example, if the solution during a time step carries a normally acceptable error of 1 Pa (~1.5e-4 psi), a huge flow conductance could make such errors problematic.

LOSS devices (LOSS, LOSS2, CTLVLV) are particularly sensitive to such flow oscillations because the their flow resistance becomes zero at zero flow. Tubes, STUBEs, and ORIFICEs should be considered as alternatives. However, such paths are not immune to oscillations, they are just less sensitive.

Often it will be necessary to add artificial resistance to such a path: adding length (TLEN) to a tube or STUBE or reducing its diameter (DH), reducing the hole size (AORI) of an ORIFICE, etc. Such changes often do not affect pressures and other important results substantially, and are therefore justifiable.

If artificial resistance is added, the user should be make sure that vibrations and tilting* do not realistically cause some stirring between adjacent compartments. If such conditions are present but not modeled explicitly, perhaps a single tank (or twinned tank pair) should have been used to represent both bays, eliminating the problematic low-resistance flow paths in between.

* For example, turbulence and banking in an airplane fuel tank application.

Full Tanks with Ports on Top, Empty Tanks with Ports on Bottom—When a twinned tank transitions to a homogeneous tank at either very low qualities or very high void fractions, the resulting mixing of phases can cause a sudden liquid level and pressure change (Section 3.25). Similarly, when a bubble first forms in a liquid-filled tank or when the last bubble disappears, when a droplet first appears or the last droplet disappears, pressure changes can be significant even without the use of phase suction on exhaust paths (which is implied by use of ports).

Ports are often applied to the very top or bottom of a vessel. Such ports can contribute to the above step functions, and they are susceptible to them as well.

If a port is placed on the bottom of a vessel that is draining, for example, it will continue to extract the very last drop of liquid and then suddenly jump to 100% gas or vapor (which may cause a flow reversal as noted above). If liquid is first injected into a dry vessel through this bottom port, it will continue to "see" the gas/vapor phase until the it is submerged by more than its effective radius, per the hysteresis that has been described previously.

Analogous issues exist at the top of a nearly full vessel. In fact, if the vessel is modeled as a very compliant (COMP>0) tank, pressure changes can cause volume changes that can in turn disrupt logic or expressions that use the tank's volume to predict the coordinate location of the liquid/vapor plane. For example, a port near the top of a tank can become "submerged" due to volume/pressure growth even if some void still exists.

To avoid this issues and the corresponding analytical distractions, consider placing ports just above the bottommost point or just below the top-most point. When starting a transient with a full vessel, make sure that ports on the top are initialized (phase suction or twinned tank choice) to "see" the correct phase, and consider starting the transient with a small amount of gas or vapor already present. When filling the vessel, consider stopping the transient just shy of 100% full. Analogous measures can be taken for nearly empty vessel: start with a little liquid if filling, and end before the last liquid disappears when draining. Consider also flat-front modeling of a duct (Section 3.27) if the liquid/vapor front enters the tank from that duct, or if the liquid/vapor front flows into that duct.

ORIFICEs as Holes between Vessels—When modeling a vent or bleed hole between adjacent compartments, an ORIFICE device is a good choice. The AORI can be chosen to be the actual hole size, the ELLD can be chosen to represent the thickness of the wall, and a large AF (say 10*AORI) can be chosen as the path flow area such that the built-in correlation can be used to predict CDIS and other parameters based on a severe "contraction" to the hole.

One caution regarding such usage is that the ORIFICE correlations are based on fully turbulent flow, which implies a small hole or strong pressure differential between the lumps. K-factors similarly assume fully turbulent flow, and any underrepresentation of the flow resistance can result in flow oscillations as noted above.

An extra caution is applicable when such orifices are used as ports on the walls between two-phase tanks. Recall that the hysteresis of the phase suction (or twinned tank connection) is based on the "radius" of the path. For most paths, this hysteresis radius is calculated internally based on the AF of the path, assuming a circular shape. For ORIFICEs, to support this "hole modeled as orifice" usage, this hysteresis radius is instead calculated based on AORI assuming a circular aperture (R= SQRT(AORI/π)).

3.14 Variable Volume and Compliant Tanks

As the default condition, the walls of control volumes (tanks) are assumed to be fixed and infinitely rigid. Such an assumption is usually adequate for most analyses, but prohibits the modeling of many systems and devices. Fortunately, the user can override this assumption using either or both of two methods:

1) the control volume may expand or contract as a function of time, or

2) the control volume may stretch or shrink in response to pressure changes.

The ability to alter the size of the control volume allows the user to better simulate such diverse devices as diastolic pumps, pistons, and accumulators and such diverse phenomena as water hammer, acoustics, gas inclusions, and separated phases.

The volume of a tank can also vary if it is contained within a subvolume, as defined by any active ifaces that are connected to it. Furthermore, adjusting the COMP and VDOT of a tank indirectly affects other tanks within the same subvolume. Refer to Section 3.8 for more information on ifaces and subvolumes, and in particular to Section 3.8.6.1 which describe the affects of using both ifaces and COMP or VDOT.

3.14.1 Definitions

Tank walls may expand or contract over time if the *volume expansion rate* is nonzero. This growth rate, designated VDOT, is defined as:

$$\Psi = \frac{dV}{dt}$$

where:

Ψ . . . . . Volume expansion rate (VDOT)
V . . . . . Tank volume (VOL)
t. . . . . Time

The units of VDOT are volume/time; VDOT is positive for expansion and negative for contraction. The default is VDOT=0.0. Note that tank volumes may not be negative; a continuous negative VDOT will result in a diminishing time step as the volume approaches zero.

Tank walls may also be given a *compliance*, meaning they will stretch elastically if the pressure rises, and will shrink if the pressure drops. Tank wall compliance, designated COMP, is defined as:

$$C = \frac{1}{V}\frac{dV}{dP}$$

where:

C . . . . . Compliance (COMP)
P. . . . . Tank pressure (PL)

Compliance has units of inverse pressure (1/psi or 1/Pa, depending on the master model unit system). Compliance is the inverse of stiffness or volumetric spring rate; the higher the compliance, the softer the tank wall. A zero compliance (the default) corresponds to an infinitely stiff tank wall, which is the default assumption. Negative compliances are legal and have even found some modeling uses, but they should be used with caution since they are destabilizing terms.

A tank with both nonzero COMP and nonzero VDOT will grow or shrink with both time and pressure changes:

$$\Delta V = \Psi \cdot \Delta t + C \cdot V \cdot \Delta P$$

A stiff-walled tank (small COMP) will be dominated by the VDOT term, growing or shrinking with time relatively independent of pressure changes. Conversely, a soft-walled tank (large COMP) may stay the same size while the pressure builds or drops according to VDOT (i.e., the tank wall may stretch to accommodate a negative VDOT). As an aid in thinking about the interactions between the two options and the modeling choices between them, VDOT terms may be thought of as a forcing function or as an action, while COMP terms may be thought of as a response function or as a reaction. For this reason, VDOT terms should be used carefully, while positive COMP terms are more innocuous and usually help smooth the system response.

Thermodynamically, a growing control volume performs work on its environment, whereas a shrinking volume absorbs work energy from its environment. For example, the temperature of a gas will rise as it is compressed, and will drop as it is expanded. This phenomena is correctly simulated in FLUINT whether the volume change is due to time or pressure or both.

3.14.2 Referencing COMP and VDOT in Logic Blocks

COMP and VDOT may be used as variables within logic blocks in the same manner as TL and other lump descriptors (e.g., [fsmn.]COMPn). The user should *not* reset these values in FLOGIC 1. Also, since tanks are treated like junctions in STEADY, both of these factors are ignored in STEADY with the exception that tank volumes are updated at the end of STEADY for nonzero COMP to reflect overall pressure changes. Therefore, the user should temporarily set COMP to zero before calling STEADY if he or she does not wish the volume to be affected by the call.

Several simulation routines assume control of the VDOT and COMP terms, including NCGBUB. When using those routines, the user should avoid altering these values.

3.14.3 Modeling with Volume Rates

Volume expansion rates can be used to simulate pistons, diastolic pumps or compressors (such as the mammalian heart), and accumulators that are squeezed or expanded. Because the system response can be very sensitive to changes in VDOT values, the following rules should be obeyed:

1) Avoid perpetually negative VDOT values—tank volumes must be positive.
2) Make sure the fluid has some where to come from or go to, such as a plenum, a compliant tank, or the compressibility of the fluid itself.
3) Do not change the value suddenly or extremely. Pressure spikes and notches will result when VDOT is changed on a liquid-filled tank; the adjacent flow rates must suddenly change to accommodate the new mass balance. Sinusoidal fluctuations in the volume can be absorbed, saw tooth fluctuations (discontinuous VDOT over time) will probably cause problems. In both cases, constraining the time step to a fraction of the period should be considered mandatory.

3.14.4 Modeling with Compliances

Compliances can be used to model flexible lines and nonrigid containers, liquid compressibility, noncondensible gases, and bladder accumulators. Adding even a small compliance to all tanks will result in smoother hydraulic responses and therefore less problems with user logic changes and real transient effects. *Small* compliances should be considered mandatory for hard-filled capillary tanks, hard-filled tanks next to control valves, two-phase tanks that can become hard-filled, and tanks with sudden VDOT or QL/QDOT changes.

3.14.4.1 Pipe Wall Flexibility

The spring rate inherent in the wall material of pipe or vessel can be modeled with a compliance. While flex hoses and rubber lines are obviously flexible, even metal pipes have some flexibility. For thin-walled pipes with small deflections that are within the elastic range, the compliance is constant:

$$\text{COMP} = \frac{D}{E\,t}$$

where:

D . . . . . Pipe diameter; the inner diameter (DH) is adequate
E . . . . . Young's Modulus or stiffness of wall material (units of pressure)
t . . . . . Pipe wall thickness (same units as D).

3.14.4.2 Gas Compressibility

While the compressibility of gaseous working fluids is intrinsically included, the effects of noncondensible gas bubbles in liquids and gas bladder accumulators adjacent to liquids can be simulated using compliances. (Of course, multiple-constituent mixtures of gases and liquids might

also be an appropriate modeling choice, perhaps using ifaces to describe the liquid/vapor interface.) Assuming a perfect gas that is in thermodynamic equilibrium with the working fluid, the compliance is:

$$COMP = \frac{V_g}{P\ V_t}$$

where:

V_g . . . . . Gas volume
P. . . . . Tank and gas pressure, *in absolute units* (PL)
V_t . . . . . Tank volume (VOL)

Note that the fluid network does not include the gas itself, only its effect on the softness of the tank wall.

For small gas bubbles, the ratio V_g/V_t approaches the void fraction. For example, to add a small bubble of gas (say void fraction of 0.1%) in every tank to smooth the system response*, set COMP to 0.001 divided by a characteristic system pressure (say 14.7 psia) in an LU DEF subblock (see Section 3.3.4):

```
LU DEF, COMP= 0.001/14.7   $ ENGLISH UNITS (1/psia)
```

To model a gas bladder accumulator with fixed total volume VOLACC, specify a tank of volume VOL (such that the volume of the gas in the accumulator is then VOLACC-VOL) and add the following logic to FLOGIC 0:

```
COMPn   = (VOLACC-VOLn)/((PLn-PATMOS)*VOLn)
```

where n is the tank identifier. Additional logic may be necessary to ensure that VOL is less than VOLACC, or that a minimum compliance is maintained. The above formula assumes the bladder is very compliant compared to the gas; complex compliances are discussed below.

For very large gas bubbles, fast transients, or otherwise relatively slow heat transfer between the gas and the working fluid, the assumption of thermal equilibrium (constant temperature) may be replaced with one of constant entropy (i.e., a fast but reversible compression or expansion of the gas), resulting in a somewhat smaller compliance:

$$COMP = \frac{V_g}{\gamma\ P\ V_t}$$

where:

γ . . . . . Ratio of specific heats (C_p/C_v) for gas in bubble

Similarly, the compressibility of adjacent vapor (for hard-filled tanks next to a capillary device) or any other fluid may be simulated without assuming a perfect gas:

* In many systems, tiny gas bubbles are purposely introduced to avoid pressure surges.

$$COMP = \frac{V_g \zeta}{\rho_g V_t}$$

where:

ρ_g . . . . . Density of the adjacent gas or fluid

ζ . . . . . The absolute value of the partial derivative of density with respect to pressure at constant temperature (or constant entropy for quick changes) for the adjacent gas or fluid (may be found using property routines for standard library fluids)

To model an adjacent vapor volume where heat and mass transfer between the liquid and vapor phases must be included, refer to twinned tanks (Section 3.25).

3.14.4.3 Liquid Compressibility

FLUINT normally assumes an incompressible liquid state, although 6000 series FPROP blocks can instead use a fully compressible liquid (see COMPLIQ option in Section 3.21.7). For other working fluids, the compressibility of the liquid can be modeled if needed using compliances as follows:

$$COMP = \frac{1}{K_B} = \frac{1}{F \rho a^2}$$

where:

K_B . . . . . Adiabatic bulk modulus of liquid (units of pressure)

F . . . . . unit conversion factor: 1.0 for standard SI units, 1.665E-11 for standard ENG units (ft, hr, lb_m, psi)

ρ . . . . . liquid density

a . . . . . Speed of sound in liquid (use VSOSF routine unless using 9000 series fluids without defining compliance)

NOTE: FLUINT does not specifically track liquid (or vapor) compression or expansion waves. The solution method used specifically avoids resolution of such “fast transients” to allow larger time steps. However, when the time step is restricted suitably, and when both the compliance of the liquid and its container are included (see Section 3.14.4.4 below), FLUINT *can* be used for water hammer and induced load studies. Excellent comparisons have been made with codes that are specifically intended to simulate water hammer. FLUINT predictions have also closely matched test data for situations that included flashing (column separation). Refer to Section 7.11.9.6 for more details on modeling of liquid compressibility (using the COMPLQ routine) and pressure waves.

3.14.4.4 Adding Compliances

If there is more than one reason for a tank to be compliant, a total effective compliance can be easily calculated. Thus, all of the above phenomena (tank wall flexibility, liquid or gas compressibility) can be summed together and simulated simultaneously.

The formulas differ according to whether the compliances are in series or parallel. A gas bubble *inside* a rubber-walled tank is an example of parallel compliances, since there is a volume change associated with *each* cause. Any number of parallel compliances can be added algebraically to yield a single effective compliance:

$$COMP_{eff} = COMP_1 + COMP_2 + COMP_3 + \cdots \quad \text{(PARALLEL)}$$

As an example, for water hammer analyses the compressibility of the liquid can be added to the compressibility of the pipe. Or, if the system wave speed is known, the compliance can be calculated directly using the formula presented above in Section 3.14.4.3.

A gas bubble *behind* the wall of a rubber-walled tank (e.g., a gas-bladder accumulator) is an example of series compliances, since there is *one total* volume change associated with *both* causes. The inverse of any number of series compliances can be added algebraically to yield the inverse of a single effective compliance:

$$COMP_{eff}^{-1} = COMP_1^{-1} + COMP_2^{-1} + COMP_3^{-1} + \cdots \quad \text{(SERIES)}$$

Combined series and parallel compliances can be calculated using a spring rate or electrical resistance analogy (such as is used in SINDA!). For example, assume there is gas both behind and inside a bladder. The compliance of the bladder is added to the compliance of the gas behind the bladder using the series formula, and then this effective compliance is added to the compliance of the gas inside the tank using the parallel formula.

When modeling spring-like devices such as metal bellows, it is important to note that compliances are analogous to the "kx" term in the spring equation $F = k(x-x_o) = kx - kx_o$. The term driving the spring to the return position or *rest state*, kx_o, is *not* included in the compliance term. This driving term must be added separately via VDOT values, or perhaps even the HC of adjacent tubes or STUBE connectors. Alone, compliances can only be used to model such spring-like devices if the rest state corresponds to the initial condition. See also the SPRING iface in Section 3.8.3.3 for a more complete treatment of a bellows accumulator and similar devices.

3.14.5 Compliances as Smoothness and Safety Measures

Strictly, compliances should be used to represent real phenomena, such as gas inclusions and elastic walls, and *all* pipes and containers exhibit some compliance even if this is limited to liquid compressibility and the flexibility of a metal pipe. (Refer to the simulation routine COMPLQ in Section 7.11.9.6.) Unfortunately, severe transient events (such as rapid changes in pumps, valves, VDOT, and QL or QDOT values) coupled with the assumed incompressibility of liquid can result in unrealistically large pressure spikes and notches in hard-filled portions of the network. Therefore, adding extra "artificial" compliance to tanks will make the program much more rugged. Even errors in program use (such as situations where the fluid has nowhere to flow) are easier to detect, and coarse changes made in user logic blocks are more tolerable. For these reasons, *small* compliances

should be considered mandatory for hard-filled capillary tanks, hard-filled tanks next to control valves, two-phase tanks that can become hard-filled, and tanks with sudden VDOT or QL or QDOT changes.

However, note that an unrealistically large compliance can give misleading results by allowing the tank to stretch like a balloon (the simulation would be correct, but the model would be inappropriate). Compliances are inherently small numbers; typical values might be 1.0E-4 in English units, and 1.0E-8 in SI units.

3.15 Spatial Accelerations and the AC Factor

Each path in FLUINT has one characteristic flow rate. Therefore, each tube or connector has one characteristic velocity. In many cases, however, the “real” velocity on one end of a path is not equal to that on the other end, even at steady-state. This can be caused by an inlet from a stagnant source, flow area changes (e.g., sudden contractions, diffusers, nozzles), density changes (due to heating or cooling or movement of gas and liquid), side passages, or by other modeling assumptions, as will be shown below.

There is a gain or drop in static pressure associated with such a velocity gradient: the difference in dynamic head. This pressure gradient is caused by the *spatial acceleration* of the fluid: a driving force is required to accelerate or decelerate the fluid within the path. Spatial accelerations may also be thought of as differences in momentum flux (mass flow rate times the velocity vector) for a momentum control volume formed by the path.

This is a reversible effect, meaning that *if the flow rate reverses, a drop becomes a gain and vice versa.* The AC (Area Change or ACceleration) parameter is used to model such *recoverable* losses. Other terms (FC, FPOW, FK) cover the effects of *irrecoverable* losses such as friction. Irrecoverable losses reduce total pressure, while recoverable losses increase or decrease static pressure.

There are three ways in which AC factors, which default to zero, can be set:

1) For individual (not duct macro) paths, provide AFI and AFJ inlet and outlet flow areas (whether or not they are equal). For a REDUCER or EXPANDER, DHI and DHJ may be used to indirectly set AFI and AFJ.

   AF will be maintained as the average flow rate, and AC will be calculated according to the formulas presented in this section, taking into account varying areas and/or densities. To disable such calculations during processor execution, set AFI and AFJ to zero or to a negative value (to which they default).

2) LINE and HX duct macros (Section 3.9.2) automatically calculate the AC factor (amongst other terms) for the paths they generate, and are the recommended approach for including such terms in lines. These calculations are able to take into account side passages, transient flow rate gradients, etc. This is also true for “extended duct macros” that may be created automatically by Sinaps and FloCAD (Section 3.9.2.2).

3) If neither of the above two options are used, the AC factor defaults to an initial value of zero for individual tubes and STUBE connectors, and is available to the user for modeling area and/or density changes, as described next.

The TB2TAB routine (Section 7.11.8) is useful for tabulating AC factors, the parameters that determine their values, and their net effect on the path in terms of effective pressure force. *Note that user-specified values of AC might be overwritten each solution step as needed to preserve solution stability.* Internally, FLUINT uses a more implicit formulation: $\Delta P_a = AC*FR^2 + b*FR + c$, where b and c are internal variables (vs. user parameters such as AC). TB2TAB lists the total pressure force

due to spatial accelerations ("DP ACCEL"), of which AC*FR^2 might only be a fraction (perhaps even zero). Therefore, TB2TAB and TUBTAB list "ACeff" which is the spatial acceleration divided by the flow rate squared: $AC_{eff} = \Delta P_a/FR^2$. DPTAB and DPTAB2 also list the spatial acceleration term in units of pressure difference.

3.15.1 Background: Modeling Changes in Flow Area

By default, all lump pressures in SINDA/FLUINT represent the local (static) state, or that pressure which would be felt by an observer traveling along with the flow. Although not tracked in SINDA/FLUINT, stagnation (total) pressure is also a useful concept. It represents the hypothetical state that would exist if the flow at any point were decelerated adiabatically to zero velocity. (To designate any lump as "stagnant" or having negligible velocity, use LSTAT=STAG, in which case its temperature and pressure will be total values. This choice affects the outflowing paths, however, and so should not be used indiscriminately. See also the TOTALTP query routine.)

When an area change occurs in a piping system, the total pressure drops due to *irrecoverable* losses. Irrecoverable losses (frictional dissipation etc.) remain a loss when the flow reverses. Thus, it doesn't matter whether the flow area grows (expander or diffuser) or shrinks (reducer), the *total* pressure will decrease.

The *static* pressure (i.e., the lump PL value) also decreases for flow through a reducer (Figure 3-31). Not only has an irrecoverable loss occurred, but a *recoverable* one as well: the fluid has been accelerated from a low velocity to a higher velocity state, requiring a pressure gradient to balance the momentum flux gradient. Such spatial accelerations also occur in evaporators, and in outlet manifolds (see Section 3.15.3).

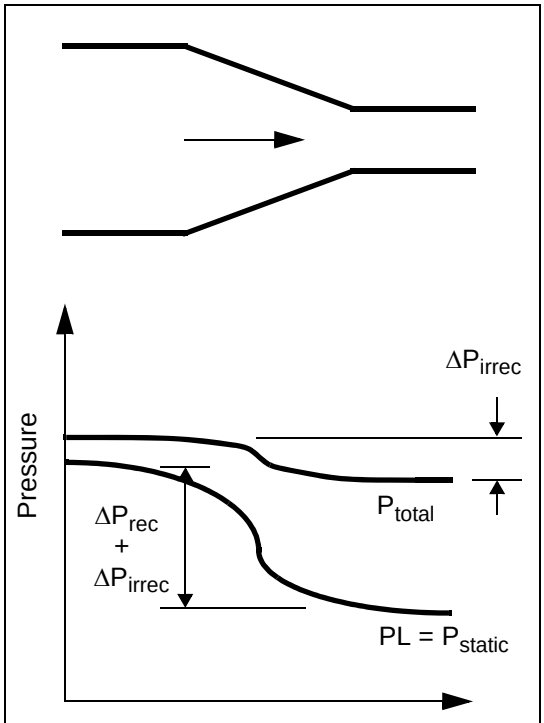


Figure 3-31 Pressures in a Reducer

A recoverable loss becomes a gain (at least in theory) if the flow reverses. Thus, even though total pressure drops in an expander, the static pressure *might* increase because the flow has been decelerated from a high velocity state to a lower velocity state (Case #1, Figure 3-32). Such spatial decelerations also occur in condensers, and in inlet manifolds (see Section 3.15.3).

If the recoverable "gain" associated with velocity decrease is greater than the irrecoverable losses (associated not only with friction but with low pressure regions caused by detachment of the boundary layer), then the expander is said to re-

cover pressure since the static pressure is higher at the outlet than the inlet. (An expander specifically designed to recover static pressure is often called a *diffuser*.) Otherwise, static pressure drops along with total pressure, although not by as large a difference (Case #2, Figure 3-32).

Actually achieving pressure recovery is difficult for turbulent flows, typically requiring a full-angle α on the order of 6 to 12 degrees, with an optimum near 10 degrees. At larger angles, detachment occurs and flow separation losses dominate. At smaller angles, wall friction dominates since the required length of the diffuser becomes so large.

To model recoverable losses, the AC factor of a constant-area tube or STUBE connector can be used. The formulas for the calculation of the AC factor are presented later in this section (Section 3.15.2.1 and Section 3.15.2.3). Note that these formula are often approximations.

Instead of calculating AC, the user may simply enter AFI and AFJ, the flow areas at the defined inlet and outlet of the segment (see Section 3.15.2.1). The program will then automatically update the AC (and AF) terms according to current conditions.

For REDUCER and EXPANDER connectors, the flow area is always different from inlet to outlet, so the AC factor is always calculated. The irrecoverable losses are (excepting MODEC=0) automatically calculated as well.

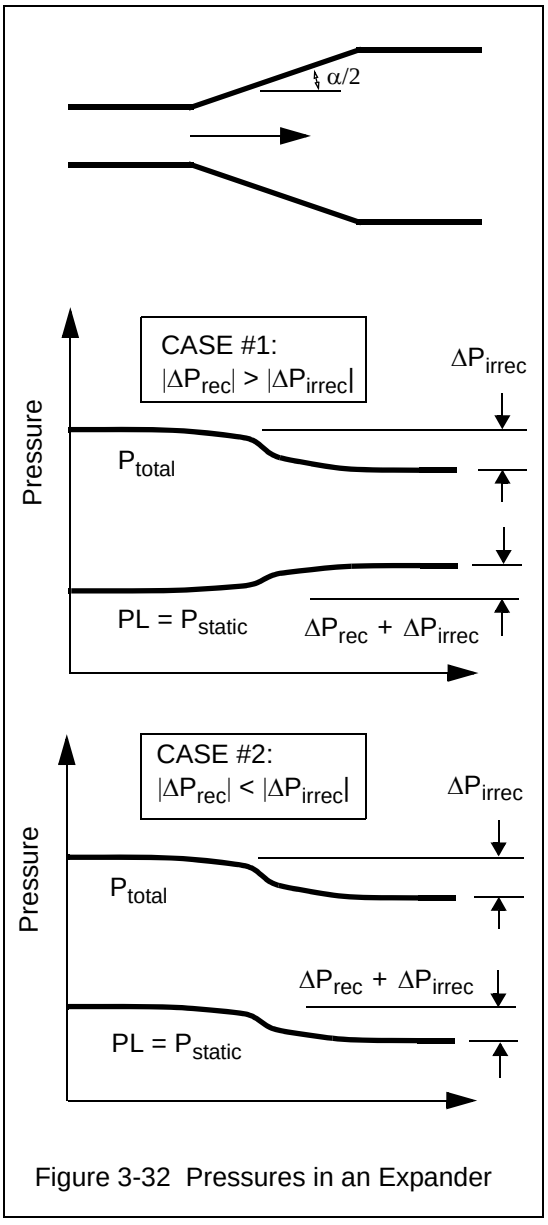


Figure 3-32 Pressures in an Expander

3.15.2 Modeling with the AC Factor

The AC factor is only optionally input for a constant area tube or STUBE connector. *Spatial acceleration is automatically calculated for a tube or STUBE connector with AFI/AFJ inputs, or for an EXPANDER or REDUCER, or for any path that is part of a duct macro.* Nonetheless, even if it is not the user's intent to specify the AC factor themselves, it is helpful to know the basis of those automatic calculations as well as their uncertainties.

AC factors are based on *defined* inlet and outlet conditions, and do not change if the flow rate reverses during a solution step. As will be seen, between solution steps it is a good idea to update a user-input AC factor if flow has reversed if the densities have changed.

The AC factor is otherwise constant because a positive value causes a static pressure gain in the positive flow rate direction and a loss in the reverse direction. (By comparison, FC and FK factors always cause a total pressure loss in any direction—they are *ir*reversible.)

The following subsections list formulas useful in calculating the AC factor for causes such as area and/or density changes. Users are cautioned that the potential for instabilities is high in instantaneous models (e.g., models with STUBE connectors and junctions) when density terms are present, since the AC factor is assumed to remain constant over the interval even though the density in junctions can change instantaneously.

3.15.2.1 Area Changes

For area changes in incompressible flows (or at least in cases where density changes are small), AC is only a function of fluid density and the *defined*[*] inlet and outlet flow areas A_i and A_j, respectively. If the area change is relatively small (A_i and A_j are nearly equal):

$$AC = \frac{2(A_j - A_i)}{\rho A_i A_j (A_i + A_j)} \qquad \text{(SLIGHT AREA CHANGE, CONSTANT DENSITY)}$$

Even though the user might calculate AC given two flow areas A_i and A_j, only one value of AF actually exists in FLUINT. This flow area, AF, is used to describe the tube or STUBE for the purposes of friction calculations (i.e., FC and FPOW) and for heat transfer calculations (if the path is used in an HTN, HTNC, or HTNS tie). The AF for the path should therefore be chosen to represent an average value, with the hydraulic diameter calculated to match (assuming a circular cross section):

$$AF_{avg} = \frac{1}{2}(A_i + A_j) \qquad \text{(AVERAGE FLOW AREA)}$$

$$DH_{eff} = \sqrt{\frac{4}{\pi} AF_{avg}} \qquad \text{(EFF. HYD. DIAMETER)}$$

If AFI and AFJ are used in a tube or STUBE connector, the AF relationship is applied automatically. Both the AF and DH formula are always applied for a REDUCER or EXPANDER connector as well: AF and DH become output values.

Using such an area averaging scheme for incompressible flows causes the momentum equation to correspond to Bernoulli's equation for slight changes in flow area.[†] Any desired differences between the "heat transfer geometry" and the above "momentum geometry" can be accommodated via the tie area fractions or tie UAM factors, or by distinguishing DEFF from DH (they are otherwise equal).

---

* In other words, flow rates are defined to be positive if they flow from the i^{th} lump to the j^{th} lump. The defined up- and downstream ends represent a sign convention only, neither those ends nor the definition of the AC factor changes if flow rates reverse.

† *Only in incompressible and frictionless flows can the AC for area changes be known exactly.* Otherwise, a detailed profile of flow area vs. axial distance would be needed along with a few other simplifying assumptions, such as a constant pressure gradient (e.g., linear axial variation in pressure). Some such assumptions have already been made in the formulas presented in this section.

When the above formula is used for AF, the prior equation for AC becomes:

$$AC = \frac{1}{\rho AF_{avg}}\left(\frac{1}{A_i} - \frac{1}{A_j}\right)$$ (SLIGHT AREA CHANGE, CONSTANT DENSITY)

If flow area changes are greater (A_i and A_j significantly differ), then the above formulas underestimate the acceleration factor. A better estimation is to use $AF_{eff}=2A_i*A_j/(A_j+A_j)$ resulting in the following:

$$AC = \frac{1}{2\rho}\left(\frac{1}{A_i^2} - \frac{1}{A_j^2}\right)$$ (LARGE AREA CHANGE, CONSTANT DENSITY)

However, since the flow area AF is used for other purposes too, FLUINT uses a simple average for AF and applies a correction factor to AC to achieve the same results as the above equation.

The above relationships are available in "prepackaged" form by using REDUCERs or EXPANDERs or by using the AFI and AFJ variables for tubes and STUBEs. When these flow area values are set, AF is set to the average value, and AC is calculated according to the above formula (at least if density happens to be constant--see Section 3.15.2.3).

The tube or STUBE connector can be assumed to be frictionless by specifying IPDC=0 and FC=0.0 as long as AC and FK are never both zero. This assumption implies an ideal reducer or diffuser.* In that case, the AF and DH are only relevant for HTN, HTNC, and HTNS ties, although the AF of a tube is still used in calculating its inertia.

Instabilities can result in tubes and STUBE connectors with AFI<AFJ (or AFI>AFJ if the flow reverses) if irrecoverable loss factors (Section 3.15.2.4) have been excluded or underestimated. Therefore, if flow area changes are detected along with a zero K-factor (FK), the code will issue a warning and calculate the K-factor automatically using an incompressible sudden expansion† or sudden contraction loss. This mechanism can be exploited to have the program calculate such losses purposely. However, use of a REDUCER or EXPANDER connector is strongly recommended instead.

The internal throat area, AFTH, is independent of these calculations for a tube or STUBE connector. (In fact, the only current use for the AFTH factor is in the choking calculations.) In other words, no assumption is made regarding *where* within the path the throat exists. If AFI and AFJ are used and AFTH has not been input, however, AFTH is initialized to be the smaller of AFI and AFJ.

For a REDUCER or EXPANDER, AFTH is normally calculated automatically.

* A frictionless diffuser is mathematically unstable since there is no limit to the pressure it can recover. This is also true of a diffuser or expander for which the irrecoverable losses have been underestimated, even if they are not zero.

† This loss may be insufficient to regain stability: it may need to be augmented via a user-supplied F-factor (FK).

3.15.2.2 Density Changes

AC may also be used to represent spatial accelerations in homogeneous paths due to density changes between the inlet and the outlet. Such spatial accelerations are significant in evaporating or condensing two-phase ducts. These terms represent the pressure loss needed to accelerate a slow moving liquid into a fast moving vapor, or the recovery of pressure from condensing high speed vapor. The formula is:

$$AC = \frac{(\rho_j - \rho_i)}{\rho_i \rho_j AF^2}$$

(DENSITY/PHASE CHANGE, CONSTANT FLOW AREA, CONSTANT MASS FLOW RATE)

where ρ_i and ρ_j are the densities at the *defined* inlet and outlet, respectively. When phase change is negligible, density gradients may exist even without velocity gradients:

$$AC = \frac{(\rho_i - \rho_j)}{(\rho_i AF)^2}$$

(DENSITY GRADIENT, CONSTANT FLOW AREA, CONSTANT VELOCITY)

Note that simple formulas are not possible in intermediate cases such as unsteady flows or partial phase changes. *In nearly all cases, AC cannot be calculated exactly, and instead must be estimated.*

If the user elects to calculate spatial acceleration factors when the cause involves density changes, extreme caution should be used because of the potential for serious numerical problems. The above formula does not take into account transient lags in velocity in ducts modeled with tanks, nor does it include the effects of side passages. Both effects are automatically included in duct macros, as described below.

The above relationships are available in "prepackaged" form for a tube or STUBE connector by setting the AFI and AFJ variables equal to the flow area AF. When AFI and AFJ are defined as positive (and in this case, equal) values, AC is calculated according to the above formula. When AC is calculated by SINDA/FLUINT, the downstream density (e.g., ρ_j for positive FR) may be not be the same as the density of the downstream lump, since that lump's state may not be representative of the fluid flowing through the tube or STUBE connector.* In this case, the downstream density is instead calculated based on an isenthalpic expansion of the fluid entering the path. This outlet density is roughly that which would result at an unheated, singly connected junction at the outlet of the path.

As a reminder, density changes and other effects are included automatically for REDUCER and EXPANDER connectors, as well as *any* path within a duct macro. These formula are mostly presented for explanatory purposes, since the need for a user to apply them to a stand-alone (nonmacro) constant-area tube or STUBE connector is a rarity.

* For example, if the outlet is a plenum, or a tank that is filling or emptying in a transient, or a junction that is heated or that is receiving flow from other lumps, then the state of that lump is no longer useful for the calculation of velocity gradients within the current tube or STUBE connector.

3.15.2.3 Combined Area and Density Changes

When both area changes and density changes occur, the AC factor for slight area changes can be *estimated* as:

$$AC = \frac{2(\rho_j A_j - \rho_i A_i)}{\rho_i A_i \rho_j A_j (A_i + A_j)} \qquad \text{(SLIGHT AREA CHANGE WITH PHASE CHANGE)}$$

Once again, an average of A_i and A_j should be used for the path AF, and all prior comments and cautions regarding modeling of area and density changes apply. If AF is an average of A_i and A_j, then the above equation reduces to:

$$AC = \frac{1}{AF_{avg}}\left(\frac{1}{\rho_i A_i} - \frac{1}{\rho_j A_j}\right) \qquad \text{(SLIGHT AREA CHANGE, WITH PHASE CHANGE)}$$

If flow area changes are large, an *approximate* AC factor is:

$$AC = \frac{1}{2}\left(\frac{1}{\rho_i A_i} - \frac{1}{\rho_j A_j}\right)\left(\frac{1}{A_i} + \frac{1}{A_j}\right) \qquad \text{(LARGE AREA CHANGE, WITH PHASE CHANGE)}$$

The above relationships are available in “prepackaged” form for tube and STUBE connectors by using the AFI and AFJ variables. When these values are set, AF is set to the average value, and AC is calculated according to the above formula. When AC is calculated by SINDA/FLUINT, the downstream density (e.g., ρ_j for positive FR) may be not be the same as the density of the downstream lump, since that lump’s state may not be representative of the fluid flowing through the tube or STUBE connector. This point was described in the previous subsection.

As a reminder, area and density changes are included automatically for REDUCER and EXPANDER connectors, as well as *any* path within a duct macro. These formula are mostly presented for explanatory purposes, since the need for a user to apply them to a stand-alone (nonmacro) constant-area tube or STUBE connector is a rarity.

3.15.2.4 Irrecoverable Losses Associated with Area Changes

When modeling area changes, it is important to remember that AC determines only the *recoverable* “losses:” changes in static pressure, not total pressure. If the flow area reverses, such a loss will become a gain (or vice versa). FK and FC factors, on the other hand, describe irrecoverable losses: pressure will drop as a result of these forces whether flow continues in the current direction or reverses. (See Section 3.15.1 for a background discussion.)

The AC calculations performed by FLUINT do not take into account the irrecoverable nature of flow area changes. The user must supply such factors either as FC (with IPDC=0), or more commonly as an FK factor. These irrecoverable losses are calculated automatically as the FKI factor in an EXPANDER or the FKJ factor in a REDUCER.

Although such losses are comparatively minor when flow areas are gradually reduced (i.e., converging sections), they can be very high in expanding or diverging sections. *Instabilities can easily result if the user models an expanding section without accounting for such losses.*

When adding such losses in the form of FK factors, the user is cautioned that FK factors are applied in FLUINT on the basis of average flow area (AF), while the formulas presented in most texts and papers are usually based on the inlet or outlet flow area (AFI or AFJ). Thus, the FK factor must often be multiplied by either $(AF/AFI)^2$ or $(AF/AFJ)^2$.

The FK factor is an output for a REDUCER or EXPANDER, and is defined at the average area AF. The FKI factor applies at the defined inlet (AFI), and the FKJ factor applies at the defined outlet (AFJ). Therefore, for a REDUCER or EXPANDER:

$$FK = \left(\frac{AF}{AFI}\right)^2 FKI + \left(\frac{AF}{AFJ}\right)^2 FKJ \qquad \text{(EFFECTIVE K-FACTOR AT AVERAGE FLOW AREA AF)}$$

If FK factors are being calculated within a tube or STUBE, refer to the formula for FKI for an EXPANDER (Section 3.5.4), or for FKJ for a REDUCER (Section 3.5.3). Note that the above formula would have to be used to convert the head basis to the average flow area AF in order to input the FK for a tube or STUBE connector. Using an EXPANDER or REDUCER in series with a constant-area tube or STUBE is strongly recommended instead, if applicable.

If AFI and AFJ have been input for tubes or STUBE connectors but irrecoverable losses have been neglected, FLUINT will issue a warning ("FK was 0 ...") and perform a *one-time* calculation of FK based on *current* FR, AFI, and AFJ values presuming an incompressible sudden expansion* or sudden contraction. The user can use this as a default mechanism. However, use of REDUCERs or EXPANDER connectors are strongly recommended instead to avoid both instabilities and inaccuracies.

3.15.2.5 Inlets and Outlets (Entrances and Exhausts)

When fluid enters into a passage from a large vessel (where the fluid is essentially stagnant), or when it exhausts to such a condition, both accelerational and irrecoverable losses occur.

At an inlet, an FK of between 0.05 and 0.8 should be applied, depending on whether the inlet is smooth, sharp, or reentrant. *In addition*, since FLUINT pressures are static by default, an FK of 1.0 should be added (*for default LSTAT=NORM upstream lumps only*) to include the fact that the upstream fluid must be accelerated from stagnation, such that the total FK will be on the order of 1.05 to 1.8.

Just because a plenum or large tank is used to model the upstream state does not imply to the code that the upstream velocity is zero. It is meaningless to assign a velocity to a lump: lump pressures are by default static, not stagnation. Nonetheless, an assumption of negligible velocity *can* be asserted using the LSTAT=STAG option for the upstream (source) lump, which means the pressure is taken as stagnation. If this election is made, then an internal acceleration equivalent to a K

* This loss may be insufficient to regain stability: it may need to be augmented via a user-supplied F-factor (FK).

factor of 1.0 is automatically applied to all suitable paths* flowing out of the stagnant lump, and this acceleration should *not* be added to the FK factor (i.e., the FK should be between 0.05 and 0.8). This "hidden" acceleration happens automatically when LSTAT=STAG, taking into account flow reversals. ***Therefore, LSTAT=STAG is the preferred means of modeling an entrance or exhaust to a stagnant state.***

If such a pressure loss causes a large density change, an AC factor of $1/(2\rho AF^2)$ can be used to replace one unit of K-factor loss (in which case LSTAT should be NORM for the upstream lump so as not to account for the acceleration twice). In other words, the inlet can be treated as a sudden contraction from a large flow area. If the AC factor is used instead of the FK factor to model the inlet acceleration, the irrecoverable losses must still be modeled with a FK factor of between 0.05 and 0.8.

At an outlet, it is customary to apply an FK (K-factor) of unity in incompressible flows since the dynamic head of the fluid is lost. However, this treatment is not compatible with FLUINT, again because of its default use of static pressures. ***It is not necessary and is in fact incorrect to add additional losses to the exit***† *if they have been added to the inlet already. If the source of the upstream fluid is an LSTAT=STAG lump, the acceleration loss as already been applied at the inlet and should not be added to the outlet.*

Since this is a common point of confusion, it will be rephrased for emphasis. In the common "total pressure approach" for incompressible flows, a K-factor of unity is added to the **exit**. This is to be contrasted with the more generalized "static pressure approach" used by FLUINT, in which exit losses are absent but **inlet** accelerations must be included.

If flow reverses, an inlet can become an outlet and vice versa, perhaps causing some of the modeling decisions and inputs to be revisited. *This is handled automatically using the LSTAT=STAG option*, and if this option is not employed then FK or AC factors must be modified using logic or expressions to cover the case of flow rate reversal.

Many other phenomena apply at sharp or reentrant inlets, including fluidic constrictions (vena contracta, which affect choking) and boundary layer development. Those topics are outside the scope of this subsection, which focused on acceleration forces.

3.15.2.6 Radial Blowing (Injection) and Suction (Extraction)

Spatial acceleration may also result from radial blowing or suction, causing an axial velocity gradient. In this case, the difference in momentum flux is due to differences in the direction and magnitudes of the in-flowing and out-flowing momentum vectors. One example occurs in the modeling of heat pipe type channels. In such channels, evaporation at the wall injects mass radially into the flow, forcing an acceleration, while condensation extracts mass radially, decelerating the flow. In either case, the flow rate at one end of the channel is zero. If the injection or extraction is uniform,

* "Suitable paths" means tubes, STUBE connectors, and LOSS-family connectors (LOSS, LOSS2, CHKVLV, CTLVLV, UPRVLV, DPRVLV, and ORIFICE) but excludes NULL, CAPIL, PUMP, VFRSET, and MFRSET connectors.

† In contrast, it *is* reasonable and common practice to add a "sacrificial" LOSS element to an exhaust expected to be a choke point, as described in Section 3.18. However, such a LOSS element uses an intentionally negligible loss (say, FK=0.01).

the flow rate along the length of the channel is linear, as shown in Figure 3-33.

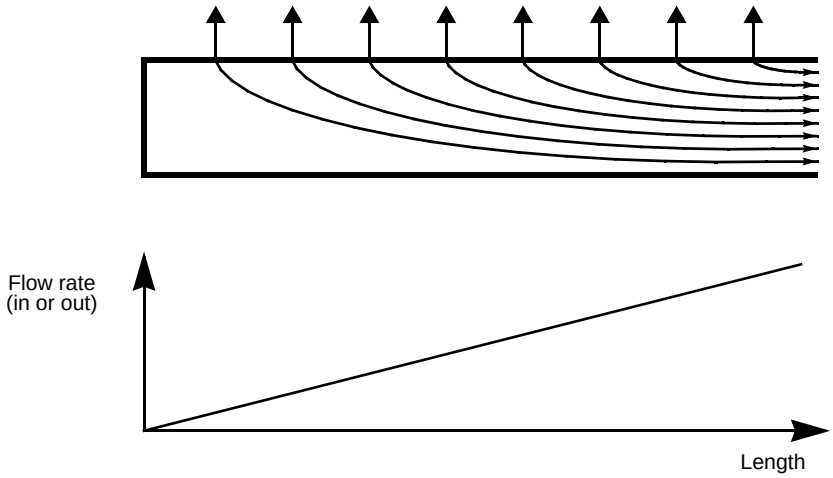


Figure 3-33 A Passage with Uniform Radial Injection/Extraction

To model the frictional characteristics of such as passage, a common “half-length” assumption is used: the full flow rate is assumed to act over half of the actual length. To include the acceleration effects (the fact that the fluid being injected or extracted radially does not contribute to the axial momentum flux):

$$AC = \frac{1}{\rho A F^2} \qquad \text{(SUCTION, EXTRACTION)}$$

$$AC = \frac{-1}{\rho A F^2} \qquad \text{(BLOWING, INJECTION)}$$

Notice that the sign is determined by whether the flow rates are positive for suction or for blowing, but do not change if the flow reverses (i.e., blowing becomes suction). In other words, if the flow rate is zero at the defined upstream end, the sign of the AC factor is negative since only the downstream flux term remains, and vice versa.

The above modeling method assumes one lump and one path are used to represent the whole segment. If it is desired to break the segment up into smaller pieces, the equations become more complicated. For such cases, the use of a duct macro (Section 3.9.2) is recommended.

3.15.3 Automatic Acceleration Calculations in Duct Macros

The AC factor is automatically calculated in macros generating duct models (LINE and HX, introduced in Section 3.9.2), including extended duct macros generated by Sinaps and FloCAD (Section 3.9.2.2).

(In Sinaps and FloCAD, REDUCER and EXPANDER connectors may also be included within a duct macro. The rules described in this section apply to them as well, unless otherwise noted. Note that REDUCER and EXPANDER connectors cannot be twinned, unlike tubes and STUBE connectors.)

These acceleration calculations are needed to model area changes (excluding corresponding loss factors), density changes, and flow rate gradients (perhaps due to transient compression/expansion or due to side passages), but not for inlets and outlets. Otherwise, the AC factor on a tube or STUBE that is not owned by a macro is left as zero by default. The user generally forfeits the ability to independently specify the AC factor when using a duct macro, although the AC factor for tubes may be modified in FLOGIC 1 in all solution routines except STEADY (aka, "FASTIC"). For twinned tubes and STUBE connectors, the FG coefficient follows the same rules.

In homogeneous duct macros, calculations based on (but not identical to) the above formulas are automatically invoked. Other factors such as mass flow rate gradients due to compressibility and side flow passages are also included. Furthermore, a more detailed formulation of the acceleration term is included for increased stability. Instead of the acceleration term being equal to $AC*FR^2$, for duct macros linear and constant terms are added as well: $\Delta P_{accel} = AC*FR^2 + b*FR + c$, where b and c are internal variables (vs. user-accessible parameters such as AC). The equivalent pressure force term ("DP ACCEL") can be viewed as printed by the TB2TAB output routine (Section 7.11.8).

If the paths are twinned, then the basis for the internal algorithm is derived from the area change formula presented in Section 3.15.2.1, since each twin has nearly constant density but occupies a varying portion of the flow passage. Furthermore, if paths are twinned the related FG coefficient is calculated, accounting for the small momentum transfer associated with phase change. Therefore, *use of duct macros is strongly recommended to represent evaporating or condensing sections and other compressible flow lines with large density changes.*

On the other hand, note that ***additional turning losses associated with tees or wyes are not automatically included*** *when modeling a manifold using a duct macro, much less any time paths are joined in FLUINT.* Neglecting these factors can result in overestimation of pressure losses (in an outlet manifold) and pressure recovery (in an inlet manifold). FLUINT uses a one-dimensional momentum equation that neglects the fact that flow entering or exiting a manifold does not do so perpendicularly, even if the branch tube is perpendicular to the manifold. Consider using wye/tee (diverging/merging) K-factor correlations (YTKALI, YTKALG, YTKONV in Section 7.2) in addition to and perhaps instead of duct macros when modeling manifolds that cannot be assumed to be ideal.

AFI and AFJ can be used to model flow area change effects in addition to the other causes of velocity gradients. When defining an HX or LINE macro in FLOW DATA, it is important to remember that these parameters are taken to mean the inlet and outlet flow areas of the whole macro. These values are then used as end-points for a linear interpolation of the internal AFIs and AFJs used on each path. Thereafter, the user may change the AFI and AFJ values of individual paths, perhaps via the MACONE re-initialization subroutine (Section 7.11.1.9). Once again, loss factors should be included (Section 3.15.2.4). Also, recall that an FK defined in a FLOW DATA macro block is applied to each generated path.

An AC factor with the same sign as the flow rate FR means that the fluid is decelerating, which can actually result in a pressure gain instead of a drop. While this is physically realistic, the assumption of instantaneous reaction in connectors, combined with decelerations, can cause the flow rate to fluctuate wildly and reverse suddenly. Such behavior is often seen in rapidly condensing duct macros built with STUBEs. To avoid such instabilities, the user should add more reality to the model in the form of head losses on the high-velocity vapor side. For example, FK=0.5 might be placed on the inlet path of a condensing duct to represent entrance losses in the manifold or header. If tubes are used, then the time step may be limited by stability for analogous reasons. If so, then the time step status line will indicate a "STABILITY LIMIT."

In duct macros where the evaporation or condensation rate is severe, the fluid might boil or condense within a distance shorter than the discretized segment length. In such cases, the program will have difficulty converging or will require small time steps because of the sharp gradients. These gradients are particularly troublesome in condensers due to the spatial deceleration, but can be troublesome in any case because they affect heat transfer calculations and frictional calculations as well. Either more spatial resolution is required, or the user might approximate the heat transfer process using a heater junction (HTRLMP, Section 7.11.1.4). Refer to Sample Problem E (specifically, the steady-state initial conditions for the abbreviated model) for examples.

End Effects—The reason that a duct macro automatically takes into account spatial acceleration terms, whereas a single (nonmacro) tube or STUBE connector does not (unless AFI and AFJ are input), is because a path in the middle of a duct macro "knows" that the paths immediately up and downstream of it are part of an aggregate momentum control volume: it can trust that the flow rates in the adjacent paths can be used for gradient calculations. It also "knows" that the lumps on either end of it are contained within this extended momentum control volume, and it can therefore take into account the transient storage and release of mass in those lumps and the effects of side-flow passages if present.

This is not true for the first and last paths in a duct macro, independent of the discretization scheme. For many reasons, the states of lumps and paths outside of a duct macro cannot be used for gradient calculations, even if there is only one path downstream of the macro and it has the same flow area as does the last macro path. Therefore, while inlet (but not outlet) densities can be taken into account in the calculation of ΔP_{accel} as can AFI/AFJ changes, zero density gradient is otherwise assumed at the exits of a duct macro (as depicted in Figure 3-34 for the representative but fictitious outlet of a duct macro with constant flow area and only density changes). In other words, the program essentially assumes $\rho_j=\rho_i$ for the exit path (normally the last path in a duct macro, assuming positive flow rate). This effect places a limit on the usefulness of duct macros containing very few lumps, since there is a truncation effect that is less pronounced for macros built with finer resolution.

Furthermore, the effects of fluid compressibility in transients and side flow passages are taken into account for macro paths whose endpoint lumps are also contained within the macro, but those additional velocity modification terms are neglected at the endpoints of path (whether the inlet or the outlet) if those endpoints are *not* also contained within the macro.

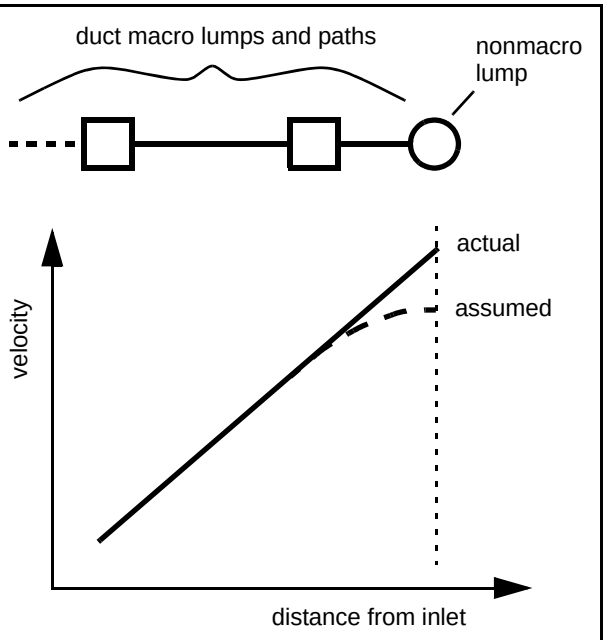


Figure 3-34 Actual vs. Assumed Velocity at the Exit of a Duct Macro

Thus, breaking a single duct macro in two introduces an extra inlet and exit and the break point: the carry-over of momentum from one macro to another is therefore lost:* different answers may result. Refer to Section 3.9.2.2 for means of avoiding this loss of accuracy.

The reasoning for this "limitation" is in part to preserve numerical stability, and in part because such treatment is consistent with the definition of a duct macro: a self-contained aggregate momentum relationship which cannot take into account gradients outside of itself. If such end gradients are important, then the macro should be extended to include the external paths as well.†

* Kinetic energy is carried through, but not momentum flux gradients (the acceleration term).

† Sinaps and FloCAD automatically attempts to collect lumps and paths in series into as few duct macros as possible, as explained in Section 3.9.2.2. ("Pipes" are always treated as duct macros.) This treatment can be invoked or disrupted either intentionally or unintentionally. Unintentional disruptions of duct aggregation can happen by changing tank/junction or tube/STUBE distinctions, or by adding side flow passages: even if they have zero flow, they might prevent Sinaps or FloCAD from deciding which route is the primary/continuous flow path.

3.16 Flow Regime Mapping and Related Pressure Drop

IPDC, the frictional pressure drop correlation identifier for tubes and STUBE connectors, is 6 by default.* This invokes automatic calculation of FC and FPOW for single phase flow based on Moody chart friction factors. In the two-phase regime, if the default of IPDC = 6 is used, a best-guess two-phase flow regime will be predicted and the pressure gradient will be estimated on the basis of that regime. These correlations are described in Appendix A.

If UPF is not 0.0 or 1.0, each tube or STUBE connector will be described by distinct flow regimes at both ends. The PTHTAB output routine prints the current flow regimes at each end of the tube or STUBE connector if IPDC=6 and the flow is two-phase. If the flow is single-phase, the Reynolds number is printed instead. These values may also be inspected (but not changed) in logic or expressions as the path parameters REY (Reynolds number) and MREG (integer regime identifier per Figure 3-35). For example, “REY22” is the Reynolds number for path 22 in the current submodel, assuming that path has a flow area AF.

If IPDC is negative, the user is forcing the selection of the regime, in which case MREG=|IPDC|.

The special case of IPDC = 6 (or negative IPDC) with two-phase flow is the subject of this section. This section describes the simplified flow regimes used internally, the rationale for choosing between them, and the implications of that choice on the frictional pressure drop. While IPDC=6 (or negative IPDC) is optional for single homogeneous paths, it is *essentially* a requirement for slip flow (twinned) paths, since otherwise the input requirements (factors such as FD, FG, etc.) can quickly overwhelm anyone but a two-phase expert. Hence, in many ways this discussion is a prelude to the next section (Section 3.17) detailing slip flow modeling.

Note that flow regime mapping *is* required for modeling ducts composed of twinned tanks (Section 3.25) and twinned tubes, and is essential for dissolution/evolution modeling in such ducts as well. The code *must* know what regime it is in to perform such advanced calculations.

The principal reference for this section is the Design Manual for Microgravity Two-Phase Flow and Heat Transfer (AL-TR-89-027), October 1989, by C.J. Crowley and M.G. Izenson. Note that the methods used within FLUINT deviate somewhat from those described in that reference, mostly in (1) the addition of regime hysteresis, (2) the extension of these methods to unsteady and diabatic flows, (3) the preclusion of stratified flow if the liquid can reach the top of the pipe by surface tension alone, (4) different methods for predicting pressure drop in all regimes except stratified, and (5) the inclusion of terms for curved and rotating passages. With respect to slip flow modeling, the differences are even greater.

* If IPDC is set to zero, then the user assumes responsibility for calculating and updating FC and FPOW, which are described in Section 3.4.1.2. Furthermore, if the paths are twinned, the user assumes responsibility for calculating and updating FD as well.

3.16.1 Flow Regime Descriptions and Distinctions

Four generalized (simplified) regimes are recognized, as illustrated in Figure 3-35: bubbly, slug, annular, and stratified.* The first two are considered "dispersed," and the latter two "separated." The distinction between regimes is based (1) on the liquid and vapor mass fluxes, (2) on the void fraction, (3) on the hydraulic diameter of the line--assumed nearly circular, (4) on the magnitude of a body force (or acceleration) vector and its orientation with respect to the duct, (5) on fluid properties such as densities, viscosities, and surface tension, and (6) in the event no clear determination can be made, on previous flow regimes (i.e., regime boundaries exhibit hysteresis).

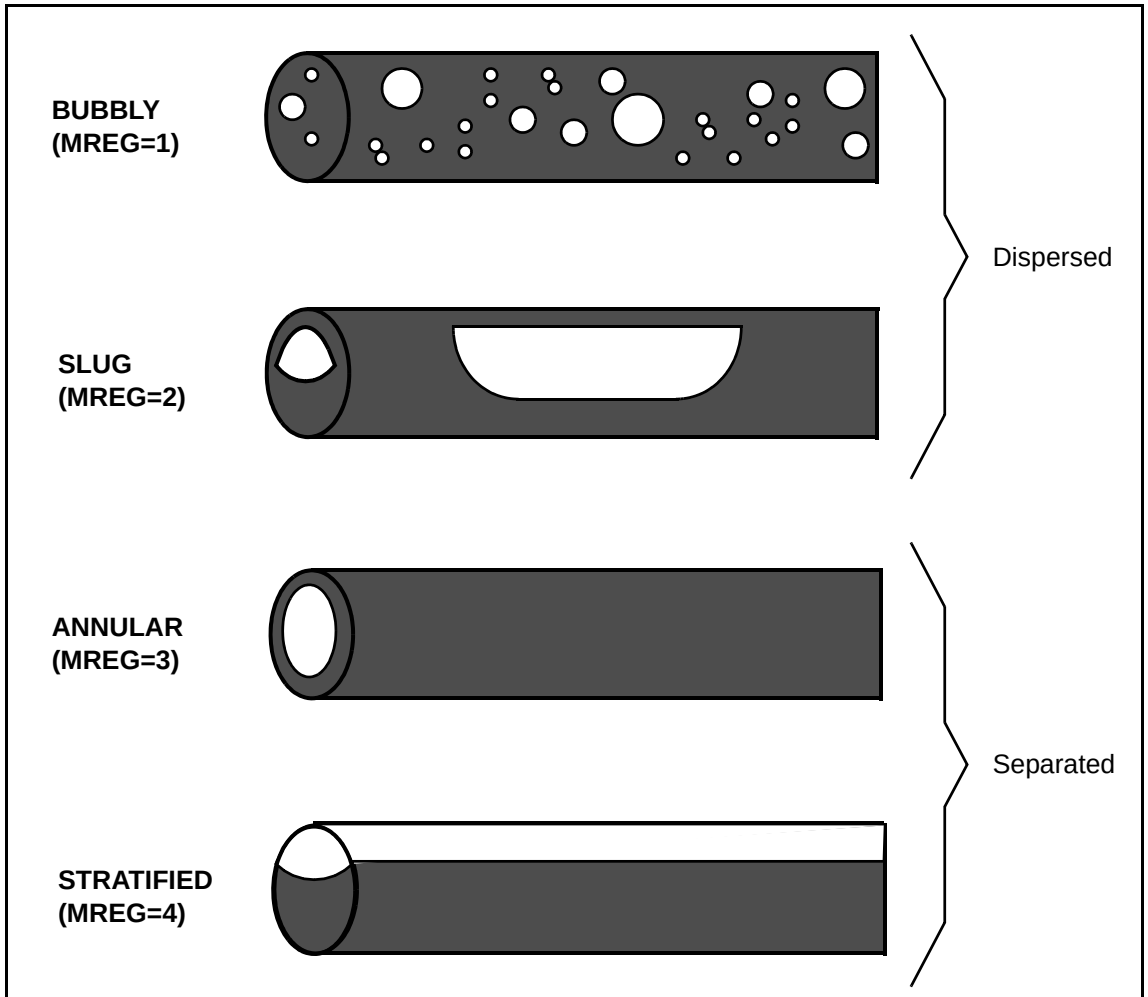


Figure 3-35 Simplified Two-phase Flow Regimes

* Note that misty annular and mist flow regimes are not currently available: no regime includes entrained droplets. Neither is the "inverted annular" regime, which may occur due to film boiling. Note also that a degenerate "regime," axially stratified, is used for flat-front two-phase modeling (Section 3.27).

Bubbly flow occurs at the extremes of low gravity (or other accelerations and body forces), high liquid mass fluxes compared to the vapor flux, and low void fractions (less than about 0.46), and is characterized by small vapor bubbles entrained in liquid. If the bubbles coalesce due to increased accelerations, decreased liquid mass flux, or increased void fraction, then the slug flow regime will appear.

The slug flow regime is typified by large bubbles that nearly span the diameter of the tube, but which are axially separated from each other by liquid. Both the slug and bubbly flow regimes are characterized by relatively little slip flow, approaching true homogeneous flow. In both cases, predicted pressure drops are based on the McAdam's formulation for homogeneous flow (which is invoked when IPDC=1). These two regimes are therefore identical for single homogeneous paths, but they behave differently if slip flow is modeled (see Section 3.17).

The annular regime may result if the void fraction continues to grow (above about 0.76), or if the liquid flows downhill, or if there is high enough vapor flux to sustain the uphill flow of liquid. This regime is characterized by a continuous vapor core surrounded and 'lubricated' by a continuous liquid annulus. In most two-phase systems, annular is by far the most common regime. When the regime is determined to be annular, the Lockhart-Martinelli correlation (also invoked when IPDC=2) is used.

The stratified regime, characterized by liquid pooling in the bottom of the tube, results if either the vapor mass flux or the liquid fraction is low enough, or the gravity high enough (and the flow is not vertically upward). The stratified regime cannot exist in microgravity or in the absence of other accelerations (sharp curvature, spin-induced centripetal accelerations, etc.). The methods used to predict pressure gradients involve calculating the height of liquid and the fractions of each phase in contact with the wall, *assuming a circular cross section* (per the method of Taitel and Dukler, 1976). Unfortunately, this model is highly sensitive to void fraction, and because the stratified regime typically exhibits the greatest degree of slip, the error in a homogeneous approximation to void fraction can be significant. In other words, *pressure drops in the stratified regime are suspect if the default homogeneous options are used: slip flow should be considered.* Typically, a homogeneous assumption results in overestimation of pressure drop for stratified flow, whereas if slip flow is modeled, the predicted pressure drop is usually lower than that of all other regimes for the same flow quality.

To allow the code to predict one of the above regimes, use IPDC=6. To instead force the selection of a regime, use negative IPDC values as follows:

-1. . . . . bubbly
-2. . . . . slug
-3. . . . . annular
-4. . . . . stratified
-5. . . . . axially stratified (only if user-specified for flat-front two-phase modeling, as described in Section 3.27 ... never predicted using IPDC=6)

The current upstream regime for each path can be inspected (but not set) using the MREG integer variable. For example, if hplate.mreg33 is equal to 3, this means the upstream regime (and usually but not always the downstream regime as well) of path 33 in submodel *hplate* is annular.

3.16.2 Regime Uncertainties: Simplification and Hysteresis

The four possible flow regimes are considered simplified because they represent the top level classifications for families of more detailed regimes. For example, the annular regime is usually subdivided into regimes such as “wispy annular,” “misty annular,” and “wavy annular.” However, the exact distinction between these subdivisions quickly becomes subjective, and the differences between mechanistic descriptions become negligible.*

Even allowing for such simplifications, the term ‘best-guess’ was not used lightly in the opening discussion. The user should not have the impression that flow regime prediction is an exact science. Effects of heating, cooling, upstream conditions, and past history can all affect the local flow regime, and as noted above, the literature (which, as usual, is largely devoted to water) recognizes many more than the four regimes used in FLUINT. This regime predicting capability simply attempts to discern the major break-points in behavior on the basis of mechanistic principles.

For example, bubbly flow does not usually exist when the void fraction exceeds some critical limit. Beyond this critical void fraction, the bubbles would exceed the maximum packing and coalesce, and the flow transitions to the slug regime. Values for this critical limit can range from 0.4 to 0.52, and even this range should not be treated as “gospel.”

Because of these uncertainties and because of the substantial differences in behavior between regimes, FLUINT employs substantial hysteresis in regime prediction for stability: *when in doubt, the last regime is favored over an alternative regime*. In effect, *all* documented uncertainties are treated hysteresis. While there is some physical basis for flow regime hysteresis, in reality not all uncertainty in regime determination can be attributed to hysteresis.

When uncertainties were not quantified, hysteresis was created somewhat artificially. For example, the critical transition criterion for slug to annular is given in the aforementioned Design Manual as a void fraction of 0.76. (Actually, this is a bit high compared to other references, again underscoring the inherent uncertainty in the flow regime determination game.) In FLUINT, annular cannot exist when the void fraction is below 0.9*0.76, and slug cannot exist above 1.1*0.76.

From a user’s perspective, this hysteresis means that different results may be achieved when starting from different initial conditions. While such behavior is certainly not new, it is perhaps more exaggerated with IPDC=6 than with other current causes of hysteresis. Different (but hopefully equally correct) answers are especially likely where flow distributions between parallel legs are calculated, and where the flow rates are based on pump curve matching, etc.

* Perhaps the greatest error in these simplified regimes is the lack of an entrained liquid phase within the vapor spaces of separated regimes. Another source of error in the simplification of the flow regimes is the lack of bubbles within the liquid spaces of slug, annular, and stratified regimes.

3.16.3 Interactions with the ACCEL vector, Curved Tubes, and Spinning Flow Passages

The body force or vehicle acceleration vector ACCEL affects not only axial pressure forces but also flow regime determination as well when IPDC=6. Since ACCEL vector components (ACCELX, ACCELY, and ACCELZ) all default to zero, *flow regime predictions are based on microgravity conditions by default.* Any nonzero ACCEL components might have an effect on the predicted flow regime.

Such interactions are based both on the magnitude of the ACCEL vector and on its orientation with respect to the path vector. The latter is defined as the vector pointing from the coordinates of the upstream lump to the downstream lump. Recall that the path vector length does not have to match the input length, TLEN, of the tube or STUBE connector.

NOTE: When the path vector has zero length, the ACCEL vector is assumed to be perpendicular to it at all points. This is important since all lump coordinates (CX, CY, CZ) default to zero (coincident at the origin), and therefore all path vectors are zero also by default. Thus, *the user can quickly assess the impacts of gravity on a horizontal system without having to input any lump coordinates at all* since any input ACCEL vector will be treated as perpendicular to *all* tubes and STUBE connectors if no lump coordinates have been defined.

Irrespective of orientation, the magnitude of the ACCEL vector can affect the bubbly/slug transition, the annular/slug transition, as well as the stratified/other transition. If the magnitude of the body force is high enough, the bubbly regime will disappear in favor of the slug regime since bubbles will coalesce into slugs due to buoyancy forces. Stratified flow cannot exist at all unless the magnitude of the ACCEL vector is great enough.

Orientation with respect to the body force vector can be important, too. Separated flows (stratified and annular) usually cannot exist if the flow is rising relative to the body force (e.g., vertical upflow) since the liquid falls back on itself and the slug regime results. This *flooding* is less likely to occur at high qualities where the vapor shear prevents it.

Accelerational effects also occur in curved or coiled tubes (tubes for which CURV is sufficiently small) when the flow rate is sufficiently high, and for spinning paths (nonzero ROTR, see Section 3.26). For example, sufficient centripetal acceleration can cause two-phase flows to stratify even in the absence of translational accelerations or gravity. In the case of spinning paths, the perpendicular force can only be roughly estimated, but usually the flow-wise force is dominant and/or two-phase applications of spinning passages are scarce.

3.16.4 Multiple-constituent Two-phase Heat Transfer

Flow regime information is not only used for pressure drop calculations and slip flow simulations, it can also be used to better estimate heat transfer coefficients in HTN, HTNS, and HTNC ties when two-phase multiple-constituent flows are present. Use of the default IPDC=6 option is therefore recommended for such flows.

3.17 Slip Flow Modeling Using Twinned Paths

By using a single path to model a passage in which two-phase fluid is flowing, the user is implicitly assuming that the flow is homogeneous, meaning that the velocities of each phase are equal. In fact, for most purposes homogeneous fluid flow is modeled as a single effective 'phase' having properties intermediate between pure liquid and pure vapor phases.

The assumptions of "one lump, one thermodynamic state" and "one path, one characteristic flow rate" are fundamental to FLUINT's lumped parameter network style. Actually, the latter statement should be revised to read "one path, one characteristic *velocity.*" Other than that lexical change, slip flow modeling does not violate those assumptions. Just as the modeling of nonequilibrium (imperfectly mixed) two-phase control volumes with twinned tanks (Section 3.25) requires two control volumes, one for liquid and one for vapor, the modeling of unequal phase velocities requires two paths. In fact, *twinned paths are required when tanks have been twinned within duct macros*, though twinned paths can also be used in combination with perfectly mixed (untwinned) tanks.

Before describing how slip flow can be modeled using FLUINT, some background information will be provided for those users unfamiliar with two-phase modeling. Flow regime mapping methods described in Section 3.16 are also a prerequisite for this section.

3.17.1 An Introduction to Slip Flow

In all *single* (untwinned) FLUINT paths, a homogeneous assumption is used, meaning that the vapor velocities and the liquid velocities are assumed equal: there is zero relative velocity or slip. With the homogeneous approximation, two-phase flow is modeled as the flow of a mixture of both phases: one momentum equation describes the entire path. This assumption is usually adequate and is both simple to implement and fast to execute. Because of this assumption, there is no difference between flow quality and thermodynamic quality. Thermodynamic quality is the fraction of vapor *mass* within a segment divided by the total *mass* in that segment: $M_v/(M_v+M_l)$. Flow quality is the ratio of vapor *mass flow rate* through a segment divided by the total *mass flow rate* through that segment: $FR_v/(FR_v+FR_l)$.

In reality, vapor usually moves faster than liquid, and sometimes even in opposite directions. A slip flow formulation takes this into account, using one momentum equation per phase (see Section 3.4.1.1). Unlike homogeneous flow, with slip flow the thermodynamic quality is no longer the same as the flow quality. Conservation of mass dictates that flow quality must be the same (eventually) whether a homogeneous or slip flow formulation is used. However, the thermodynamic quality is no longer constrained by the homogeneous assumption: it becomes the new degree of freedom necessary to accommodate a new momentum equation. In other words, the thermodynamic quality and its manifestations, such as the density and void fraction, will vary as needed to balance the flow forces. Because vapor generally travels faster than liquid, *the predicted void fraction will be smaller with slip flow than with homogeneous flow at the same flow quality.* In other words, more liquid will reside in the line, and the thermodynamic quality will be smaller than the flow quality. In FLUINT parlance, if twinned paths are employed then the lump XL will no longer be an indication of the flow quality even at steady state. Figure 3-36 graphically illustrates this point for the stratified regime.

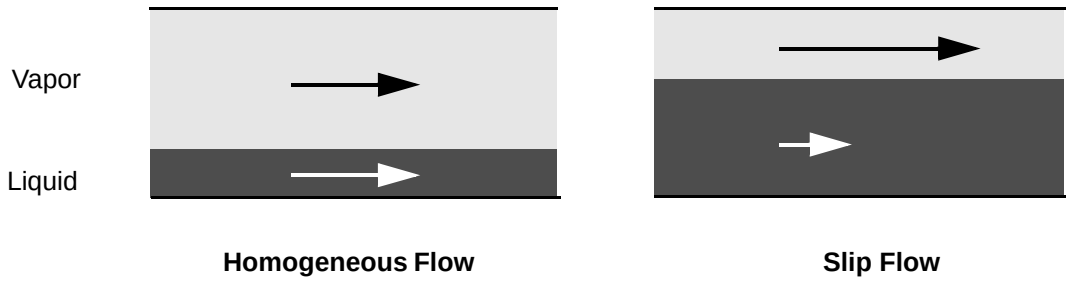


Figure 3-36 Homogeneous vs Slip Flow in Stratified Regime at Same Flow Quality

Because most pressure drop and heat transfer correlations are based on flow quality, slip flow and homogeneous formulations predict almost the same steady state as long as flow is cocurrent (see Sample Problem G); the local homogeneous assumption does not affect the overall pressure drop and heat transfer rates. The major difference is the proportion of liquid and vapor in lines. For example, in annular flow a slip formulation predicts typically three to four times as much liquid will reside in a pipe compared to a homogeneous prediction. Of course, this amount is small to begin with, and so quoting a factor of three to four might be misleading.

In transients, the differences can be more dramatic, especially for separated flow regimes where vapor can shift quickly and liquid lags behind. As a specific example, a SINDA/FLUINT model was developed to predict the time it takes to clear a small tube of volatile liquid by heating it, noting that much more liquid is displaced by generated vapor than is actually evaporated. The default homogeneous assumption resulted in a prediction of 8 seconds to clear the line, whereas allowing slip flow in the same model nearly doubled the duration of the liquid purge event. Since annular flow was quickly established, slip flow allowed the vapor to escape the tube without displacing as much liquid in the process.

This extra degree of modeling power does not come without its price. In addition to greater solution expense, a new layer of uncertainties is revealed. New parameters must be estimated, including (1) the frictional drag between phases, (2) the degree of sharing of inertia, also called added mass and virtual mass, (3) the apportionment of wall friction to each phase, and (4) the momentum transfer associated with phase change. In FLUINT terminology, these factors are called FD, AM, FC/FPOW, and FG, respectively. By default, FLUINT will estimate these factors automatically, which requires knowing the flow regime. Hence, the default flow regime mapping options (IPDC=6) should be used when specifying slip flow. Alternately, like almost all other SINDA/FLUINT options, knowledgeable users can calculate their own coefficients, or users can more simply select the regime themselves using negative values of IPDC.

3.17.2 Twinned Paths: Concept and Usage

To model slip flow in a pipe, the user creates a *pair* of twinned tubes or twinned STUBE connectors that will together model *one* flow passage. Nominally, one will carry liquid and the other vapor. They start and end at the same two lumps, sharing one definition of positive flow rate direction, diameter, flow area, wall roughness, duplication factor, etc. Although they are intimately interrelated, they are distinct paths that require distinct user identifiers.

To maintain commonality with single homogeneous paths, one of the twin paths is considered *primary* and the other *secondary.* When carrying two-phase flow, the primary path carries the liquid phase and the secondary carries the vapor phase. (This distinction is arbitrary, implying no favoritism to one phase or the other.) In FLOW DATA, the secondary path is named after the input keyword "TWIN = ," the presence of which causes two paths to be generated simultaneously.

When only one phase or the other is present in the line, only the primary path will be used even if the remaining phase is vapor. The secondary path is largely ignored until the absent phase reappears. The former mode, where both twins are active, is called the *slip flow mode*, and the latter mode, where only the primary path remains, is called the *homogeneous mode.* Refer to subsection 3.17.3 for a more detailed discussion of the mode transitions. Refer also to the NOSLIP and GOSLIP routines in Section 7.11.1.3, which give the user explicit control over the above modes.

3.17.2.1 Input

Only tubes and STUBE connectors may be twinned; slip flow modeling logic is only available for twinned tubes and STUBE connectors.

Section 3.4.1.2 describes the tube and STUBE governing equations, along with the definitions and descriptions of the relevant parameters. The rules governing the calculation of FD, FG, and AM are similar to those governing FC and AC. FD is automatically calculated if IPDC=6 (the default value), otherwise FD defaults to zero. FG, like AC, is only calculated automatically if the twinned paths were generated as part of a duct macro. Otherwise, FG defaults to zero and may be user specified. AM is always calculated for tubes. Figure 3-3 details the sequence of updating these parameters and the appropriate logic block to use for modifying them.

The default automatic calculations of FD, FG, and AM meet a special requirement: with the exception of the dissipative FG term, when the two equations for both phases are summed, both the FD and AM terms sum to zero. The FG terms would also sum to zero if an average interface velocity were used instead of the dissipative formulation used in FLUINT. Also, the total pressure forces sum to PL_i-PL_j since the fraction of that force on each phase is simply the volumetric fraction.

Use the default of IPDC=6 (or negative IPDC) for twinned paths, otherwise FD will not be calculated automatically and defaults to zero. Zero FD may cause instabilities because it is nonphysical. If IPDC is never defaulted in a PA DEF subblock nor otherwise stated explicitly, then the IPDC for all tubes and STUBE connectors will default to 6.

Note that *twinned paths cannot use the phase suction options*. Any such inputs (e.g., "STAT=...") will be accepted but ignored.

All HTN, HTNS, HTUS, and HTNC ties (including those generated by macros) must refer only to the ID of the primary tube or STUBE of a twinned pair. Involvement of the secondary path is implied, but explicit reference to it in a tie is not allowed. This treatment is compatible with the homogeneous mode, in which the secondary twin is largely ignored.

3.17.2.2 Twinned Paths with Twinned Tanks

When a tank has been twinned to model phasic nonequilibrium (Section 3.25), then a single (untwinned) path can be attached to one or the other of those tanks, thereby only seeing the phase associated with that tank. However, it is more common for a pair of twinned tubes or STUBE connectors to be connected to a pair of twinned tanks.

In that circumstance, the liquid path connects to the liquid tank, and the vapor path connects to the vapor tank. The paths cannot be made homogeneous (using the NOSLIP routine) while the endpoint lumps are twinned and in the separated mode. Furthermore, the iface between the pair of twinned tanks must be a FLAT iface to keep the pressures equal, such that the pressure drop across the liquid path is always the same as the pressure drop across the vapor path.

If the pair of tanks enters the homogeneous (combined) mode, then the secondary path will be automatically moved to the remaining primary tank, but the pair of paths may still remain in the slip flow mode.

See also Section 3.25 as well as NOSLIP, GOSLIP, GOTWIN, and NOTWIN in Section 7.11.1.3.

3.17.2.3 Output

The PTHTAB output routine will print flow regimes or Reynolds numbers for tubes and STUBE connectors. (The REY and MREG parameters refer to each path separately.) When a pair of tubes or STUBE connectors is in the slip flow mode, each will be listed separately as a single phase duct, where the Reynolds number is given as if the flow existed alone in the duct (i.e., based on superficial velocity). In this mode, the primary path ID is printed with an “L” suffix, and the secondary path ID is printed with a “V” suffix. When they are in the homogeneous mode, the secondary path is omitted, and the primary path ID is printed with an “H” suffix.

Despite input values of UPF, for stability all twinned paths internally use UPF=1.0 while in the slip flow mode. This means that of the two columns available in PTHTAB for up- and downstream flow regime or Reynolds number, only the one corresponding to the current upstream lump will be used. An exception is countercurrent flow, where a weighted combination of each state is used. When the pair enters the homogeneous mode, normal usage of UPF resumes.

An output routine similar to PTHTAB is also available that treats twinned pairs as single effective paths, printing out one pair per line. This routine, TWNTAB, ignores all single paths and unlike PTHTAB, prints flow regime and effective flow quality for all twins. In TWNTAB, when a twinned pair is in the slip flow mode, the primary path ID is printed with an “L” suffix, and the secondary path ID is printed with a “V” suffix. When they are in the homogeneous mode, and the primary path ID is printed with an “H” suffix, and the secondary path ID is printed with an “X” suffix signaling that it is currently being ignored.

TWNTAB has exactly the same argument as PTHTAB: the fluid submodel name or ‘ALL’ in single quotes. The following example demonstrates recommended usage. Consider calling both PTHTAB and TWNTAB when twinned paths are used:

```
HEADER OUTPUT CALLS, BOBSEY
       CALL LMPTAB
       CALL PTHTAB
       CALL TWNTAB('BOBSEY')
```

3.17.2.4 References in Logic Blocks

Each member of a twinned pair may be referenced in any logic block at any time, whether they are currently in the slip flow mode or the homogeneous mode. However, certain parameters such as DH and TLEN are shared by each pair, meaning that the same value is used for both paths. These parameters are: DH, TLEN, FK, AF, AFTH, WRF, UPF, IPDC, DUPI, and DUPJ. While any of these values may be referenced or altered for either path, *changes to these values will be ignored for the secondary path and will be overwritten by the corresponding value for the primary path during each solution step*. In other words, simply use the primary path ID in referencing any of these parameters, and then they can be accessed and altered in any manner legal for single homogeneous paths. Rephrasing, the secondary paths will contain the same values as the primary paths, and hence can be accessed equivalently, but any changes to these values should reference the primary path.

For example, consider that tubes #34 and #1034 are paired, with #34 as the primary twin:

```
HEADER FLOW DATA, DUBLMINT ...
       ...
PA TUBE,34,3,4,TWIN = 1034 ...
       ...
HEADER FLOGIC 0, DUBLMINT
       IF(TLEN1034 .GE. 1.0)THEN $ REF TO SECONDARY TUBE OK
            TLEN34   = 0.5    $ CHANGE TO PRIMARY TUBE OK
            DUPJ1034 = 0.0 $ DOES NOTHING: OVERWRITTEN LATER
       ENDIF
```

See also the Section 7.11.1.3 describing NOSLIP and GOSLIP, auxiliary routines that allow the user to explicitly determined whether the tubes and STUBE connectors may slip, or whether they should be confined to the homogeneous mode.

Like CAPIL and CAPPMP, phase suction options are used internally to model slip flow for active twins. Therefore, any calls to CHGSUC will be later overwritten and therefore effectively ignored. The only exception to this rule is that twins that have been forced to stay homogeneous (using the NOSLIP routine, as described in Section 7.11.1.3) will then obey subsequent calls to CHGSUC. Subsequent calls to GOSLIP erase any such NOSLIP calls.

3.17.3 Twinned Paths: Behavior

This section endeavors to give the user some feel for how twinned paths behave compared to traditional (single, homogeneous) paths. This discussion includes transitions between homogeneous and slip flow modes, descriptions of how each flow regime differs, and details of new time step limits for twinned paths.

3.17.3.1 Single Phase Limits: Homogeneous Mode

As described above, twinned paths may exist in either one of two modes: (1) the slip flow mode, where both paths are active, the primary carries liquid, and the secondary carries vapor; and (2) the homogeneous mode, where the primary path acts like a normal path and the secondary path is largely ignored. While the user may purposely choose to avoid the slip flow mode (as described in Section 7.11.1), the decision to change modes and the corresponding conversion are handled automatically.

The homogeneous mode is intended to cover the extremes of single phase liquid and single phase vapor, but is actually encountered at very low and very high void fractions in two-phase flow. Hysteresis is used for stability. For example, twinned paths will transition from slip flow mode to homogeneous mode when the void fraction of the upstream lump is greater than 0.999999, but will remain in the homogeneous mode until the void fraction falls below 0.9999. At the liquid end, the limits of the hysteresis cycle are void fractions of 0.001 and 0.01. (Analogies with the perfect mixing vs. separated modes in the twinned tank capability, Section 3.25, should be evident.) If the flow rate in either path reverses and, in doing so, cannot extract the necessary phase, the homogeneous mode will result. In other extreme cases such as zero flow or 100% liquid adjacent to 100% vapor, the homogeneous mode will result.

When a pair of paths transitions into the homogeneous mode, the flow rate in both pairs is summed to produce a single flow rate, and the primary path is returned to a STAT=NORM phase suction state. The flow rate in the secondary path is set to zero, which can be used in logic to distinguish between the two modes.

When the upstream lump of a primary path enters the two-phase region sufficiently to enable slip flow modeling, the secondary path is reactivated and the flow rate in the primary path is redistributed between the two according to the current suction quality. At that time, the phase suction of the primary path is reset to STAT=DLS and the phase suction of the primary path is reset to STAT=DVS. As always in the slip mode, all shared parameters (e.g., DH, TLEN) are copied from the primary path into the corresponding locations for the secondary path.

3.17.3.2 Interactions with ACCEL and Rotating Paths

In addition to affecting the determination of the flow regime, the magnitude and direction of the ACCEL vector and as well as any spin or rotation affect twinned paths by applying a body force on each phase in proportion to its density. In homogeneous flow, the ACCEL vector and any rotation act like superimposed pressure forces whose strength is proportional to the bulk fluid density. In slip flow, each twinned path will feel a different force and react differently, since each represents a different phase.

For example, applying gravity to a vertical duct can cause vapor to rise at the same time liquid is falling: countercurrent flow might result (see Sample Problem G). This force is always applied according to the void fraction of the uphill lump for stability, independent of flow direction. Junctions are generally incompatible with perpetual countercurrent flow.

3.17.3.3 Flow Regimes, Interface Drag, and Wall Friction Distribution

When IPDC=6, one of four flow regimes will be attributed to twinned paths: two dispersed (bubbly and slug) and two separated (annular and stratified). These regimes and the pressure drops associated with them are described in Section 3.16. When paths are twinned, the flow regime also determines the interface drag and the apportionment of wall friction to each phase. This results in different behavior in each regime.

Despite the value of UPF, in the slip flow mode only the current upstream lump affects the flow regime and hence the subsequent wall friction and interface drag calculations. In other words, twinned paths behave as if UPF=1.0 in the slip flow mode, but then resume normal UPF usage when in the homogeneous mode. The only exception is countercurrent flow, where both endpoint lumps are used in a weighted average based on the flow rate of each phase.

Unlike homogeneous flow, there *is* a difference in behavior between bubbly and slug flows in twinned paths even without an acceleration gradient. One difference is in the interface drag calculation (FD). In bubbly flow, the average size of the bubbles is estimated on the basis of the slip ratio, void fraction, and surface tension, and then the drag of the bubbles through liquid is estimated. The result is a drag force that is a strong function of the slip ratio, tending to quickly ‘homogenize’ the flow and minimize slip. Because of this homogeneity, the wall friction (FC and FPOW) is attributed to both liquid and vapor phases in proportion to the friction that each phase would experience if the other were absent.

In slug flow, a correlation based on void fraction is used (Ishii and Chawla, 1979), which again estimates the drag of long bubbles in the liquid. In the limits of very high void fractions (e.g., vertical upflow where annular is prohibited due to flooding), a check is made to ensure that the drag force is at least as strong as it would be in separated flow. All wall friction is applied to the liquid phase and none to the vapor phase (which ‘feels’ the wall only by the interfacial shear).

In annular flow, the Lockhart-Martinelli correlation is generalized into an interfacial drag correlation whose limits are set by $f_{gl}/f_{gw} = 1$ on the low end (meaning a smooth interface), and a variation of Wallis’ classic $f_{gl}/f_{gw} = 1 + 75(1 - \alpha)$ corrected for variable density ratios (ρ_l/ρ_v) on the high end (meaning a very rough interface). The result of this treatment is that steady-state pressure gradients will approximately correspond to the Lockhart-Martinelli predictions that are implied by setting IPDC=2 for homogeneous paths. As with slug flow, all wall friction is applied to the liquid phase, and none to the vapor.

In the stratified regime, wall friction is apportioned according the phase velocities as well as the proportion of perimeter wetted by each phase (assuming a circular duct). The interface drag is calculated using the width of the interface (again assuming a circular duct), a Reynolds number based on the differential velocity, and a roughness estimated on the basis of the Froude number. The limits on this drag are $f_{gl}/f_{gw} = 1$ and $f_{gl}/f_{gw} = 10$.

The less the degree of slip (with homogeneous corresponding to zero slip), the greater the void fraction. Unfortunately, the greater the void fraction, the greater the tendency to be in a regime that allows greater slip. This would lead to instabilities were it not for the sizeable hysteresis employed in the flow regime determination logic. For example, as annular flow transitions to slug flow, the void fraction decreases as liquid is swept up by the faster moving vapor. However, as the void fraction diminishes, the transition back to annular becomes more likely, setting up the potential for an endless loop.

A similar problem exists at the liquid end of the homogeneous transition, where bubbly or slug flow becomes homogeneous below a void fraction of 0.001, and the subsequent elimination of slip drives the void fraction back up. Hence, a hysteresis band is used for that transition as well: homogeneous flow is allowed to persist up to a void fraction of 0.01.

3.17.3.4 Time Steps

Twinned tubes are subject to the same time step constraints as normal tubes, except that these limits are applied to each tube of the pair. Thus, the flow rate in *any* tube is kept from changing by a ratio of more than DTTUBF (or RSTUBF) during one time step.

Twinned tubes also obey an extra limit that does not apply to homogeneous tubes: the “SLIP FLOW CHANGE LIMIT.” This is the change limit placed on the *difference* in velocity between the primary and secondary tubes. In other words, this limit makes sure that the slip ratio does not change by too much (again, DTTUBF or RSTUBF) per time step. Otherwise, the terms unique to slip flow (e.g., linearized drag, wall friction distribution, etc.) would accrue excessive error per solution step. Unfortunately, this limit is encountered frequently. The user can only avoid this limit by (1) avoiding twinned tubes, or (2) increasing DTTUBF/RSTUBF (which must be less than 1.0).

While STUBE connectors cannot directly restrict the time step, they have an indirect effect because limits are placed on the flow rate changes allowed in and out of adjacent tanks. For twinned STUBE connectors flowing out of two-phase tanks, the message “FR CHG LIM IN TWIN PATH” may appear if the time step is limited in order to indirectly limit flow rate changes that would be caused by tank void fraction changes.

3.18 Sonic Limits (Choking)

The speed of sound within a restriction (valve throat, orifice opening, nozzle throat, etc.) or at a low-pressure exhaust cannot be exceeded. Once the fluid achieves that limiting velocity at the restriction, the flow rate and pressure drop characteristics of that path are determined not by its flow resistance, but by the sonic or *critical flow* limit: the flow becomes choked.

This section details the methods by which such limits are determined, and outlines the control and output parameters available to the user to regulate the detection and simulation of choked flow.

Note that the default parameters are chosen in order to proceed cautiously: the program checks everywhere for choking, and applies nonequilibrium expansions in doing so. These defaults can frequently stress models and fluid property limits, so ***knowing when and how to suspend choking calculations becomes an obligation on the user, even in models in which choking is not an issue.***

3.18.1 Background and Computational Methods

3.18.1.1 AFTH: Throat Area

For most K-factor loss elements and valves (LOSS, LOSS2, ORIFICE, CTLVLV, CHKVLV, UPRVLV, and DPRVLV connectors), the minimum flow area is much less than the flow area (AF) that is used as the basis for dynamic head and kinetic energy. For reducers and expanders, the throat area can be less than the minimum area due to the presence of a vena contracta.

For sonic limit and other critical flow rate checks, this minimum throat area, AFTH, must be input as a separate parameter if it differs from AF. (In ORIFICE, REDUCER, and EXPANDER devices, AFTH is calculated by the program based on other inputs.) In tubes and STUBE connectors with varying flow area or internal constrictions, AF represents a characteristic flow area for friction, kinetic energy, heat transfer, and inertia calculations and is normally greater than the value that should be used for choking calculations. By default, AFTH is the same value as AF (or the minimum of AFI and AFJ if those options are used for tubes or STUBE connectors).

As with AF, AFTH may be input or defaulted for any path. It represents the minimum flow area within the path, or at least the flow area that should be used for sonic limit and critical flow rate checks. When choking calculations are performed, the fluid is assumed to be isentropically expanded from an upstream state to the throat area, as described below (Section 3.18.1.3) if AFTH is less than AF or if the upstream lump employs LSTAT=STAG.

Currently, choking calculations are the only use of AFTH.

As a caution, the user might consider a value of AFTH that is even less that the actual (physical) minimum throat area for devices with abrupt reductions in flow area. For example, in the cases of sharp-edged contractions, inlets, and orifices, a *vena contracta* can form due to two-dimensional flow inertia, reducing the effective flow area to some fraction of the actual opening. While it is difficult to predict the size of the vena contracta, typical sizes are on the order of 60% to 80% of the actual opening area. In the ORIFICE device (Section 3.5.12), AFTH is therefore always less than or equal to the hole size, AORI, and is calculated automatically by that device. In REDUCER devices

(or EXPANDERs with reversed flow), the AFTH is normally calculated automatically to be less than the AFJ (or AFI for a reversed-flowing EXPANDER). In other path types, the user should account for the vena contracta by specifying an AFTH that is smaller than the physical aperture.

For modeling flow area changes, refer to REDUCERs and EXPANDERs (Section 3.5.3, Section 3.5.4) and to Section 3.15. Although modeling an area change with a tube or STUBE is not recommended, when doing so, note that the AFI and AFJ of tubes and STUBE connectors are independent of AFTH except that AFTH may be defaulted to the smaller of AFI and AFJ during initialization (but is not updated later if the user changes AFI or AFJ during execution).

3.18.1.2 Two-Phase Sound Speed

Two methods are available for *estimating* the speed of sound in a two-phase mixture: Wallis' adiabatic method or Bursik's metastable method. The desired method is signaled by the integer argument MCH: 1 for Wallis', 2 for Bursik's method.

A third method, Dyer's method (MCH = -3, Section 3.18.1.6), doesn't use an explicit calculation of sound speed, and is discussed separately. (Dyer's critical flow rate method is only applied in expanding volatile/condensible flows. Otherwise when there is no expansion or phase change, it corresponds to Bursik's method, MCH=2.)

Wallis' adiabatic method is simple and fast, but may overpredict the speed of sound by a little (typically a few percent, but potentially up to 20%). It predicts speed of sound by assuming a homogeneous mixture of two adiabatic phases, combining the phasic compressibilities like springs in series. Wallis also proposed another isothermal method which better predicts sound speeds in two-phase mixtures, but does not predict the right value in the limits of 100% vapor and is therefore not available as an option.

The second method (Bursik's, NASA TM 78810) is computationally slower, but combines the best features of Wallis' two methods: it is a better estimation at low qualities, and it predicts the correct value in the limit of 100% vapor. However, the user should note that, like almost any two-phase prediction, there is no definitive method or exact solution. In other words, either method is probably as accurate as the other, since neither take into account flow regime variations and other effects that are known to be very important.

The user should keep in mind the high degree of such uncertainties when choosing between the two methods provided for speeds of sound: if the user has no other preference or information, Wallis' method (MCH=1) might be used because it is cheaper and it tends to overestimate the sound speed, perhaps helping to balance potential underestimations of the simplified expansion process described below. However, in many tests Wallis' method predicts a slightly (5 to 10%) *smaller* critical flow rate in liquid and low-quality regions than does Bursik's method (MCH=2) because of the importance of the expansion process compared to the speed of sound estimation.

The default for MCH is -2 for many commonly used paths (including tubes, LOSS and STUBE connectors), which signals use not only of Bursik's method by its absolute value, but the negative sign also signals use of nonequilibrium expansion, as described in Section 3.18.1.5.

All choked flow calculations assume that dissolved species stay in solution during expansion, and that they have no effect on the speed of sound in the liquid phase.

3.18.1.3 Isentropic Expansion and Critical Flow Rate Estimation

For MCH = 1, 2, -1, and -2 (a common default), the velocity in the path will be directly compared with the current speed of sound at the inlet of that path. However, under certain circumstances the program will include an expansion process: a virtual thermodynamic state ("virtual" meaning not described by any lump) will exist at a throat *within* the path.*

The expansion calculation results in critical flow rates (path FRC) that are much lower than those yielded by the simple product of the flow area, the upstream speed of sound, and the upstream density. That simple formula implicitly assumes that the flow has already been expanded, accelerated, or otherwise heated up to the sonic limit in a constant area duct. Since pure, volatile liquids almost always expand into a two-phase state in the throat, the difference caused by inclusion of the expansion process can easily be two or more orders of magnitude.

The conditions for including an expansion calculation (throat state differing from the inlet state) include:

1. if the upstream lump is stagnant (LSTAT=STAG), and/or
2. if the throat area (AFTH) is smaller than the inlet flow area (AF, or perhaps AFI or AFJ).

In determining the critical throat flow rate, the upstream state is assumed to represent total or stagnation (zero velocity, or at least negligible kinetic energy) conditions if LSTAT=STAG. Otherwise, if LSTAT=NORM (the default) and AFTH<AF, then expansion is still calculated although not from stagnant conditions: some nonzero kinetic energy will enter the device. AFTH, the throat area, is used to compute the choked mass flow rate *assuming that the flow is isentropically expanded from the upstream velocity to a critical throat state*. Because AFTH defaults to AF, and because the throat velocity is frequently much larger than the characteristic head velocity for valves and similar restrictions, the user is strongly encouraged to consider an explicit input for AFTH if choking is a concern.

If the upstream lump is stagnant and if the path entrance is submerged, such as a drain line at the bottom of a two-phase tank, then slight modifications of the expansion calculation will be made, as described in Section 3.13.4.

3.18.1.4 Hysteresis in the Choking Calculation

FR (current mass flow rate) is not actually compared directly to FRC (critical mass flow rate). Instead, slight hysteresis is applied per the HCH parameter. The default for HCH is 0.02 or 2%, meaning that flow will not begin choking until FR exceeds 102% of FRC, and will not cease choking until it is below 98% of FRC. This 4% lag exists purely for numerical stability, even though there

* If this is true, the "EXP?" column of CHOKETAB will list a "Y" for "Yes." Otherwise, "N" for "No" will appear.

is some loose physical basis for such behavior in nonequilibrium two-phase throat states. For models employing gases and/or nonvolatile liquids, setting HCH=0 (no hysteresis) is perfectly acceptable in many models and will avoid artificial behaviors.

On the other hand, if a model with many choke points is experiencing convergence problems while choosing between active choke points, consider raising HCH and checking for multiple solutions via alternate starting points (initial conditions).

3.18.1.5 Nonequilibrium Throat States

The assumption of isentropic expansion from inlet to the throat is a very common assumption that is adequate for single-phase flows and other flows without phase change. Despite the fact that the speed of sound calculation methods (Section 3.18.1.2) do not themselves assume equilibrium between phases, the code does so when predicting the throat state itself (upon which the speed of sound routines operate) if MCH is positive (MCH=1 or 2). In other words, if MCH is positive, the code predicts an equilibrium throat state, and *then* predicts a sound speed through homogeneous fluid at such a state. When a volatile liquid is present at the inlet, it will normally not have time to achieve thermodynamic equilibrium as it is expanded into the throat: not as much liquid will flash to vapor as an equilibrium assumption would predict. In many cases hardly any liquid flashes within the throat itself, although it might flash just downstream of the throat. Similarly, a high pressure cold gas may not have time to form droplets as it passes through the throat: hardly any liquid forms in the throat. Even a saturated two-phase fluid entering the path will not expand along an equilibrium isentrope: the liquid will be slightly superheated and the vapor will be slightly supercooled since phase change can only occur at finite rates.

The equilibrium expansion assumption (MCH>0) underestimates the critical flow rate if a condensible/volatile fluid is upstream of the throat, and the error can be very large for zero or low quality flows of pure substances or mixtures containing a large percentage of volatile liquids. In other words, choking may be predicted in cases where it doesn't exist.

The equilibrium approach is reasonable, relatively cheap to implement, robust, and perhaps even conservative. More realistic expansion processes take into account the fact that the proportions of liquid and vapor might not change significantly during the rapid expansion. The vapor cools much faster than does liquid, resulting in a polytropic expansion whose characteristics must be estimated, perhaps with empirical correlations.* Despite the fact that these temperature differences cause the local rate of interphase heat transfer to be very large, the transit time for fluid moving from inlet to throat is very short.

In pure-substance (single-constituent) flows, if there is sufficient pressure differential across the path (e.g, orifice or nozzle) under consideration, then subcooled single phase liquid almost always expands into a low quality two-phase state at the throat. This is the basis of operation for the cavitating venturi. Otherwise, at extremely high pressures (or when using nonvolatile liquids), liquid in FLUINT does not expand at all if the liquid is incompressible: it merely speeds up at constant saturation density until it reaches (perhaps) its sonic velocity.

* R. E. Henry and H. K. Fauske, *The Two-Phase Critical Flow of One-Component Mixtures in Nozzles, Orifices, and Short Tubes.* Journal of Heat Transfer, May 1971, Pp 179-187.

For example, in cavitating venturis* a rule of thumb is often used for predicting the critical flow rate in incompressible liquids. This rule of thumb corresponds to a throat state of saturated liquid; neither significant flashing nor superheated liquid states are assumed to occur. At high degrees of inlet subcooling, this rule of thumb is in close agreement with the predictions using equilibrium expansions (MCH>0). However, at low subcooling, such equilibrium methods predict as small as $1/1000^{th}$ of the flow rate that the cavitating venturi relationship (which is invalid at zero subcooling) predicts. Furthermore, the fluid might not cavitate: superheated liquid can exist at the throat, leading to a flow rate that is perhaps twice as high as the rule of thumb would predict.

To avoid the problem of equilibrium flashing of liquid (and equilibrium fogging of vapor), the program (using the common default of MCH=-2) first *attempts* a solution that *neglects* phase change between the upstream state and the throat state.† In other words, if volatile liquid is upstream, the program assumes that there is no time for significant vaporization to have occurred, and that the liquid is in a superheated state and the vapor is in a supercooled state within the throat.

This assumption avoids the severe underestimation of critical flow rate mentioned above, but it tends to slightly overestimate critical flow rates‡ since a little phase change *does* occur. In other words, reality is between the two limits of perfect equilibrium (positive MCH) and zero phase change (negative MCH). (See MCH=-3 in Section 3.18.1.6 for an alternative approach that is especially helpful for cavitating venturis, flashing fuel or oxidizer injectors, valves, or long orifices such as are used as expanders in vapor compression cycles.)

When there is no vapor upstream, the throat state is assumed to be at saturation with slight (negligible) vaporization, matching the cavitating venturi rule-of-thumb noted above. By itself, the assumption of negligible phase change creates a discontinuity when the liquid into the inlet is exactly at saturation: such an assumption underestimates critical flow rates at low levels of subcooling (even lower estimations than an equilibrium method would provide). Therefore, this method uses the maximum of the “slight vaporization” assumption** and the default equilibrium overexpansion assumption if the upstream quality is zero and some volatile liquid exists upstream.

For the above reasons, MCH is by default negative (-2, specifically, for LOSS-type and STUBE connectors) such that nonequilibrium expansion calculations are attempted. However, the user should be cognizant of problems and limitations caused by this default assumption.

---

* L.N. Randall, *Rocket Applications of the Cavitating Venturi,* Proceedings of the American Rocket Society, Toronto, Canada, June 1951.

† As explained below, spinodal decomposition or property limits may prevent the expansion calculation from finding such a metastable throat state, in which case the code automatically reverts to an equilibrium assumption.

‡ For different reasons, MCH=-1 or MCH=-2 can also severely *underestimate* critical flows if the throat pressure (PLTH) is much less than the downstream pressure. See MCH=-3 in Section 3.18.1.6 as an alternative.

** This is why XLTH, the throat quality at critical flow, is often precisely 3.0e-6 for MCH=-1 or MCH=-2: that is an assumption, not a prediction. Currently, if a condensible substance exists and the upstream state is 100% liquid, FLUINT applies a quality of 3.0e-6 as a tolerance on nonequilibrium expansion: a minimum degree of flashing corresponding to a very small quality. This correlates well with test data for cavitating venturis that have already flashed as long as the backpressure is well below saturation. (If they have been chilled/pressurized, the liquid at the throat can be superheated and will therefore flash downstream of the throat, resulting in much larger flow rates.)

Fluid Property Requirements for Zero Phase-change Nonequilibrium Expansion (MCH =-1 or -2). The default choice of nonequilibrium expansion for STUBE and LOSS-type connectors may cause problems in user-defined fluid descriptions if they are unable to handle requests for metastable properties: liquid properties below saturation pressures, or requests for vapor properties above saturation pressures. This choice also tends to cause problems because of the very low throat pressures that can result when flashing is largely disabled. If these problems are experienced, the user should ideally expand the fluid property description to cover the needed range. If this is not feasible, the choking calculations can either be suspended (MCH=0 for the problematic paths, or perhaps for the whole model) or the equilibrium expansion option can be used (MCH>0), but in either case the user assumes responsibility for verifying that these assumptions are justified. If property limitations are reached when using a working fluid that can change phase, the best approach is to use the more range-constrained Dyer's method (MCH=-3).

Throat pressures (PLTH) can often be below outlet pressures (the PL of the downstream lump). The program is charged with finding the minimum FRC (critical flow rate) that balances energy along an isentrope, and doing so might carry the implicit assumption that *some* sort of downstream shock (a flat shock or a Prandtl-Meyer expansion fan) must exist that raises the downstream pressure back up to a higher value. Consider the Dyer method (MCH=-3, as described in Section 3.18.1.6) as an alternative, since it uses a different definition of critical flow rate that doesn't make such implicit assumptions of shock-based pressure recovery.

Interpreting Throat States (MCH =-1 or -2). If MCH is nonzero, the code will predict a corresponding set of FRC (critical flow rate) and PLTH, TLTH, XLTH (*path* parameters describing the throat thermodynamic state in terms of pressure, temperature, and flow quality as listed in Section 3.18.2). It is important to note that this throat state is virtual unless the flow is actually choking: the program is reporting the state that *would* exist if the flow rate were as high as FRC.

If MCH=-1 or MCH=-2 and the code successfully finds a metastable throat state, then the upstream quality and throat quality are equal (excepting perhaps the slight flashing to XLTH=3.0e-6 as discussed above). Also note that XL=XLTH=1.0 is not by itself proof of supercooled vapor at the throat, since superheated vapor upstream may expand to a cooler but still superheated state.

For two-phase metastable throat states, the temperature reported is that of the vapor phase. The liquid temperature is often the same as the upstream liquid temperature (although perhaps slightly cooler for compressible liquids). The throat temperature and pressure may be colder than the triple point if allowed by the fluid properties. (Liquid properties at such a state are never evaluated below the triple point so as not to unduly stress property calculations, since the difference in results is negligible for the liquid phase.)

For two-phase equilibrium throat states, the temperature and pressure is the same for both phases, and the resulting state cannot go below the triple point without causing property error messages to result.

(When using the Dyer method, as explained in Section 3.18.1.6, the throat state represents the *equilibrium* expansion at the current maximum flow rate FRC. The actual throat state will have less phase change ... the quality will be intermediate between XLTH and the upstream XL, though it is never explicitly calculated ... once nonequilibrium estimates have separately been invoked.)

Caution Regarding use of 9000 Series Fluids. If the working fluid is a 9000 series non-volatile liquid (see Section 3.21.5) or if it is a mixture containing such a fluid, the critical flow rate (FRC) might be greatly underestimated since these fluids cannot flash or form vapor, even at very low pressures. To see if this is an issue, use the CHKFLASH utility (Section 7.11.1.10), along with virtual saturation pressure data within the 9000 series fluid definition itself.

Limitations to the Applicability of Zero Phase-change Nonequilibrium Expansion (MCH =-1 or -2). If MCH is negative, FLUINT will attempt to find a metastable throat state when calculating FRC as part of the choking calculations. If MCH=-1 or -2, a zero-phase change state is sought. (MCH=-3 uses different methods, as described in Section 3.18.1.6.)

If such a metastable state is judged to be physically impossible, or if the program manages to detect that the underlying fluid property calculations are unable to reliably extend to such states, then the attempt will be abandoned and an equilibrium throat state will be found instead. This is the difference between the meaning of positive and negative MCH: positive MCH signals the code not to bother attempting a nonequilibrium expansion in the first place. This subsection documents the meaning of “physically impossible” and how the program decides whether or not to abandon the attempt to find a metastable throat state.

At low enough pressure, a liquid will flash (“homogeneous nucleation”) *even with no disturbances or nucleation sites.* Similarly, at high enough density, a vapor will condense homogeneously. These “phase decomposition” limits, within the saturation dome, are the spinodal lines depicted generically in Figure 3-37. An expansion process can be viewed as dropping into the dome from above (high pressure). Below the saturation lines, the opposite phase *can* but *might not* occur. In fact, it rarely does occur to any significant extent in a rapid expansion process: there simply isn’t time for phase change to have occurred. Below or between the spinodals, however, a two-phase state *must* exist (or at least be in the process of forming).

Unfortunately, the locations of the spinodals are usually not known: they are difficult to determine experimentally, and although they could be predicted from basic thermodynamic considerations, the PVT (pressure-volume-temperature) surfaces of fluids are not valid or accurate near the spinodals (again, because of sparse experimental data).

As described in Section 3.18.2, the virtual path throat state is given by PLTH, TLTH, and XLTH (pressure, temperature, quality). If XLTH is zero or unity and MCH is -1 or -2, then for a pure substance the PLTH can be compared against the saturation pressure at TLTH (using the VPS function), and the TLTH can be compared against the saturation temperature at PLTH (using the VTS function) to see how far from equilibrium a throat state is. Unfortunately, no firm guidelines are available to determine how much superheating or supercooling is “too much,” and indeed partial phase change is the most likely scenario.

When MCH is -1 or -2, FLUINT attempts to find a zero-phase change nonequilibrium throat state and, failing to find such a solution, it reverts to an assumption of equilibrium expansion. Because severe property problems can occur when testing for a state beyond the spinodals, and because the spinodals represent a point past which a metastable throat state can’t exist, FLUINT uses an internal estimate of the spinodal limits (independent of the fluid description) as explained

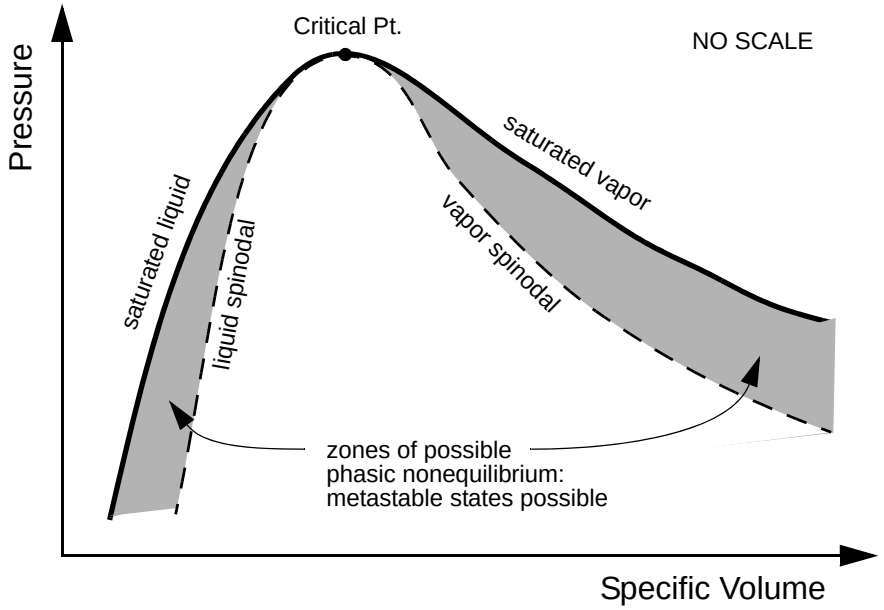


Figure 3-37 Vapor and Liquid Spinodals

below. If FLUINT detects nonconvergence in PVT calculations short of the predicted spinodal lines, it will similarly abandon attempts to find a metastable throat state in favor of an equilibrium expansion.

The internal estimations for the spinodal limit are based on the Peng-Robinson (PR) equation of state (EOS), a cubic equation of state. The reason that these classes* of EOS are called “cubic” can be seen by superimposing a isotherm for such a cubic equation of state (mathematical description of the PVT surface) through the dome (Figure 3-38). The positively sloped region in Figure 3-38 is physically impossible, and the two limits of that region (where $\partial P/\partial v|_T = 0$) represent the spinodals ... at least as predicted by that particular EOS.

The use of a PR-based estimate for spinodal limits on throat expansion is usually an overestimate; the PR equation of state often overestimates the size of the metastable zones. This is unfortunate for two reasons. First, in engineering terms, the full range of metastable states is unlikely to be experienced in the noisy, chaotic flow of a typical expansion process in a valve or orifice: the phases are likely to begin to decompose (flash or fog) before the theoretical limit (that the spinodal represents) has been reached. Second, pragmatically, the calculations for the fluid properties (which can vary under user control via FPROP block inputs) are more and more likely to experience con-

* Van der Waals and Redlich-Kwong are also examples of cubic equations of state. They tend to underpredict metastable zones. Unlike the Peng-Robinson EOS, they do not take into account the acentricity of each species’ saturation curve, which makes them inappropriate to use as a general-purpose spinodal estimation method.

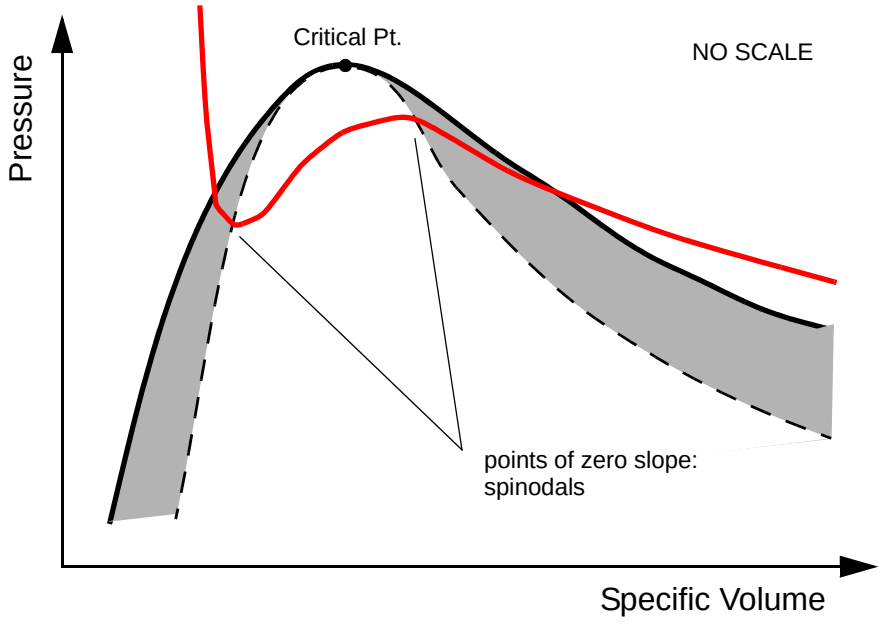


Figure 3-38 Spinodals for a Cubic PVT Surface Isotherm

vergence or other numerical problems the farther they get from the saturation line, and of course the likelihood that those extrapolations represent a real metastable state also decreases. For both of these reasons, it is best to slightly underestimate the size of the metastable zone.

Using the Peng-Robinson EOS, it is possible to calculate the minimum vapor temperature and maximum liquid temperature at any pressure, given the acentric factor for the fluid (see Section 3.21.2.3).

Using reduced properties (normalized by the critical temperature or pressure: $T_r = T/T_{crit}$, $P_r = P/P_{crit}$), the minimum vapor temperature at a specific pressure can be found using the PR EOS:

$$T_{r,min,v} = f_{PR}(P_r, \omega_{SRK})$$

In SINDA/FLUINT, this estimate is then adjusted to reduce the size of the metastable region by increasing the minimum vapor temperature, pushing it toward the saturation temperature $T_{r,sat}$ (which is calculated based on the "actual" fluid properties using the VTS function, and not based on the PR EOS):

$$T_{r,min,eff} = T_{r,min,v} - (T_{r,sat} - T_{r,min,v})*(1 - T_{r,sat})^{0.14}$$

The above function is somewhat arbitrary and has no justification other than it prevents property errors from occurring in REFPROP-derived fluid property calculations: the predicted range of metastable states is almost always smaller than the range predicted by REFPROP.*

Suspension of Nonequilibrium Expansion from a Cold Supercritical State (MCH=-1 or MCH=-2). If the upstream state for a path is in the ambiguous cold supercritical region (TL<T_{crit} and PL>P_{crit}) for a 6000 series fluid (see Section 3.21.7 in general and Figure 3-41 in particular) *and* if the upstream quality is unity (XL=1.0), then nonequilibrium expansions are disabled for choking calculations for this path. In other words, for such paths, MCH is temporarily treated as |MCH| (the absolute value of MCH). This allows the quality at the throat (XLTH) to be a small number: the path can flash into the low-quality end of the saturation dome even if the upstream state is 100% "vapor."

3.18.1.6 Alternative Nonequilibrium Method (Dyer, MCH=-3)

For MCH = 1, 2, -1, or -2, when an expansion occurs (AFTH is less than the inlet area, or the upstream lump is stagnant), then the program seeks to find an energy-conserving state along an isentrope at which the sound speed is unity. This condition is used as the FRC, or critical flow rate.

If MCH=-3, an alternate methodology is used that offers various improvements for nonequilibrium expansions. This section describes that alternate approach: the Dyer method. This is the default method for ORIFICE, REDUCER, and EXPANDER connectors, and should also be considered for LOSS-like and STUBE connectors if an expansion exists (either the upstream lump is stagnant, or AF<AFTH).

If no expansion occurs,* or if there is no phase change possible in the working fluid, then the methods described previously apply. Specifically, if MCH=-3 and such conditions are encountered, the program proceeds as if MCH=-2.† (If MCH=+3, on the other hand, MCH=+2 methods are used when nonequilibrium expansion is irrelevant.)

While no speed of sound correlation is used for this formulation, energy conservation along an isentrope still applies for calculating one *part* of the Dyer solution: FR_{hem}, the 'homogeneous equilibrium flow rate.' However, this flow rate is not expanded below the downstream pressure. Neglecting English unit conversion factors:

$$FR_{hem} = AFTH*\rho_t*[2*(h_{in} - h_t)]^{0.5}$$

The inlet enthalpy (h_{in}) is the *total* enthalpy: the thermodynamic enthalpy plus the velocity term $V_{in}^2/2$. The throat enthalpy (h_{in}) is a *static* enthalpy, where the throat state has the same entropy as the inflowing state but is at a lower pressure. The far right term in the above equation corresponds to V_t, the throat velocity. When V_t is multiplied by the throat area AFTH and the density at the throat, ρ_t, the result is FR_{hem}. Left unanswered is what static pressure should be used to define the throat state (PLTH).

---

* For built-in library fluids such as "717" (ammonia), which are much more approximate than REFPROP fluids, the EOS is not as accurate and therefore not as easily extended (extrapolated) to metastable states. For these fluids, the corrected PR-based estimates of the spinodal are too large, and the solution is more likely to return an equilibrium throat state than a metastable throat state as a result. Use of REFPROP-derived fluid properties is recommended over use of the largely outdated library fluids for this reason, as well as many others.

* "EXP?" is N in this case in the CHOKETAB column.

† The "(USED)" column in CHOKETAB will contain a -2 or 2, in this case.

For other methods (MCH= 1, 2, -1 and -2), the throat state is found by locating the point at which the speed of sound is unity. For MCH=-3, a critical throat pressure (PLTH) calculated: the pressure for which FR_{hem} is maximized as a function of downstream pressure along the isentrope: $\partial FR_{hem}/\partial PL_{down}|_s = 0$. Further decreases in PL_{down} below that critical pressure do not affect FR_{hem}. However, the throat pressure (PLTH) can be no lower than PL_{down}, and is often equal to it as a result. (This is in contrast with other methods, which allow PLTH to be lower as needed to reach the sonic velocity, assuming some shocks or other recovery must exist at the outlet.)

For the Dyer method, FR_{hem} is only part of the solution. If equilibrium were to prevail assuming an infinite transit time were available for phase change (in other words, if FRC were equal to FR_{hem}), and the critical flow rate would usually be underpredicted. Nonetheless, this is the meaning of MCH=+3: use a critical pressure method and full equilibrium expansion.

The degree of actual phase change during the expansion from inlet to the throat has always been uncertain. Dyer's key insight* was a simple estimated ratio between bubble growth time constants and fluid transit time constants:

$$\kappa = [(PL_{up} - PL_{down})/(P_s - PL_{down})]^{0.5}$$

In the original formulation, P_s is the saturation pressure at the upstream temperature. (In FLUINT, this has been generalized to be the dew or flash point along an isentrope to help extend to compressible liquids and vapor† inlets.)

Dyer noted that the larger κ, the less phase change should take place, and a simple relationship was proposed (later corrected by Solomon‡) that produces reasonable** results compared to test data:

$$FRC_{dyer} = \kappa/(1+\kappa) * FR_{spi} + 1/(1+\kappa) * FR_{hem}$$

In the original formulation, FR_{spi} corresponded to the "single-phase incompressible" or zero phase-change solution. In FLUINT, this term has been generalized to be the flow rate that would exist if no choking had been applied.

Note that if the inlet is saturated, then κ=1.0, and $FRC_{dyer} = 0.5*(FR_{spi}+FR_{hem})$. In a manner of speaking, no more than half of the maximum possible phase change is allowed.

Note too that MCH=+3, whose use should be rare, corresponds to κ=0.0.

---

* "Modeling Feed System Flow Physics for Self-Pressurizing Propellants," J. Dyer et al, AIAA2007-5702.

† Droplet formation rates are assumed to follow the same pressure scaling parameter: (P_s-PL_{down}).

‡ "Engineering Model to Calculate Mass Flow Rate of a Two-Phase Saturated Fluid Through an Injector Orifice." B. J. Solomon, Utah State University, report for Master's degree, Paper 110, 2011.
"Mass Flow Rate and Isolation Characteristics of Injectors for Use with Self-Pressurizing Oxidizers in Hybrid Rockets, B.S. Waxman et al, AIAA 2013-3636.

** Dyer estimated +/- 15% for nitrous oxide, and Waxman reported +/- 10% for the same fluid. Comparisons to other fluids such as CO_2 under other conditions have shown wider variation (up to 100%), but the results are still often better than alternative methods. Caution and conservatism are still required.

Other choking methods can be thought of as the calculation of two curves: a curve of an unchoked or "frictional" FR* versus pressure drop, and a curve of FRC. The lowest value is always used, so the point at which the two curves intersect represents a discontinuity (which is one reason for using nonzero HCH or choking hysteresis). Figure 3-39 represents such curves and the intersection between them.

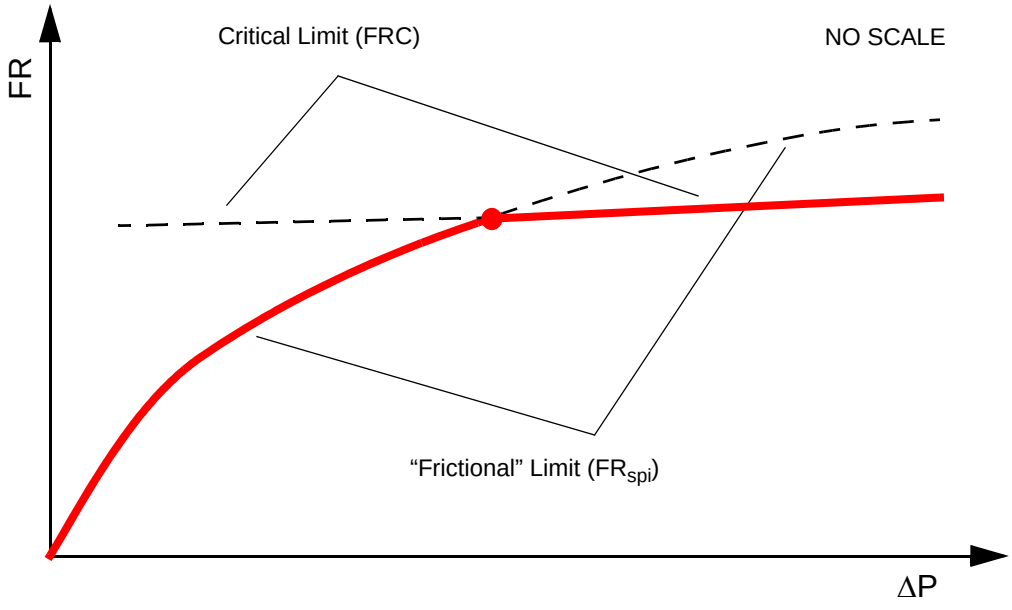


Figure 3-39 "Frictional" Limit Versus Critical Flow Limit for Choking

For the Dyer method, FR_{spi} and FR_{hem} are actually nearly equal to each when (1) there is no phase change currently occurring, (2) irrecoverable losses are not high, (3) when the outlet is a junction (or tank in STEADY) with no discontinuities in flow area such that kinetic energy is conserved, and (4) the downstream pressure (PL_{down}) is about the same as the throat pressure (PLTH). In other words, in these situations there is no discontinuity, and the two curves are virtually the same, even down to zero pressure difference and low flow rates.

In these situations, which are common for REDUCERs at low pressure differences (such that XLTH~XL_{in}), FR hovers near FRC, as if the flow were almost always "choked" even though no discontinuity in behavior is evident. In these situations, it is important to remember that PLTH is defined (for MCH=3 or -3) as a critical pressure limited by the downstream pressure. The flagging of "choking" by the program is, as a result, often just a distraction in such cases since there is no difference in flow rate or pressure drop in either condition.

* The quotes on the term "frictional" acknowledge that there are often many other forces at work in determining the unchoked flow rate. For the purposes of this section, this extended curve will be refered to as "FR_{spi}."

3.18.2 Relevant Path Variables for Choked Flow

In addition to the throat area (AFTH, Section 3.18.1.1), each path has six variables relating to the choked flow calculations: two input parameters (MCH, HCH) and four output parameters (FRC, PLTH, TLTH, XLTH).

MCH . . . . . (input, integer) Defines the choking and critical flow method.
|MCH| = 1 or 2 defines the method to be used for calculating two-phase speed of sound (Section 3.18.1.2): 1 = Wallis's adiabatic method, 2 = Bursik's metastable method. If expansion is included (Section 3.18.1.3), MCH=-1 or -2 may be used to signal the attempted use of nonequilibrium (no phase change) expansion as described in Section 3.18.1.5. MCH=-2 is the default for most paths (see exceptions below): Bursik's metastable speed of sound method with nonequilibrium expansion (for those paths in which expansion is applicable).
MCH=-3 is the default for ORIFICE, REDUCER, and EXPANDER connectors, and applies Dyer's method (Section 3.18.1.6) if an expansion with phase change is possible, otherwise the program reverts internally to MCH=-2.
Use MCH=0 to disable choked flow simulation for a path, or set MCH=0 in a PA DEF subblock to disable choked flow simulations for subsequent paths. When MCH=0, FRC (the critical flow rate, see below) will equal 1.0E30 and the throat state (see PLTH, TLTH, XLTH below) will be the upstream state.
For flows with condensible/volatile species, please see also Section 3.18.1.5 and Section 3.18.1.6 for modeling issues and cautions.

HCH. . . . . (input, real) Defines the hysteresis for entering and leaving the "choked flow mode" versus the normal unchoked mode (Section 3.18.3). The critical flow rate (FRC, see below) itself is *not* affected by the value of HCH. The default is 0.02, meaning that the flow will not choke until it exceeds 102% of the critical value, and will then stay in the choked mode until the flow rate drops below 98% of the critical value. While hysteresis can be a real phenomenon for two-phase flows, this default exists mostly for reasons of numeric stability (larger time steps, better convergence). Values of zero HCH are valid and often helpful, especially if there are multiple choke points in series. Values of up to 0.1 are recommended for two-phase flow in tubes, perhaps as high as 0.2 for twinned (slip flow) tubes. HCH values must be between zero (inclusive) and one (exclusive).

FRC . . . . . (output, real) The current critical flow rate (FR_{crit}) for this path. Equal to 1.0E30 if MCH=0 (choking detection and simulation has been suspended).

XMA . . . . . (output, real) The current Mach number at the upstream end of the path *before any expansions*. The ratio FRC/FR is more indicative of the fraction of critical flow for paths with stagnant upstream lumps or AFTH<AF (i.e., restrictions).

PLTH . . . . . (output, real) The current throat state (critical) pressure. Equal to the upstream pressure if no expansion occurs (Section 3.18.1.3). It may be lower

than the downstream pressure if expansion does occur and |MCH|=1 or 2. It will not be lower than the downstream pressure if |MCH|=3 and expansion occurs.

TLTH . . . . . (output, real) The current throat state temperature. Equal to the upstream temperature if no expansion occurs.

XLTH . . . . . (output, real) The current throat state quality. Equal to the upstream flow quality (which is not necessarily the same as the upstream lump's quality, XL) if no expansion occurs.

MCH and HCH may be set in FLOW DATA. All of the above parameters can be referenced in logic and expressions. For example, PIPE.FRC33 refers to the critical flow rate in path 33 of submodel pipe.

Note: MFRSET, VFRSET, PUMP, TURBINE, COMPPD, COMPRESS, and NULL connectors use a default of MCH=0 (no choking detection or simulation), and the value of MCH for these connectors cannot be set using PA DEF subblocks: values of MCH set in PA DEF subblocks are ignored for these devices. Therefore, if the user desires nonzero MCH for these devices, those values must be defined explicitly in those path subblocks.

Note: REDUCER, EXPANDER, and ORIFICE connectors use a default of MCH=-3 (Dyer's nonequilibrium method), and the value of MCH for these connectors cannot be set using PA DEF subblocks: values of MCH set in PA DEF subblocks are ignored for these devices. Therefore, if the user desires nonzero MCH for these devices, those values must be defined explicitly in those path subblocks.

Caution: FPROP DATA and mixtures: FLUINT will not perform choked flow calculations (*and will issue no warnings*) if appropriate properties have not been input for all user-defined fluids used in a submodel. (If no choked flow simulations are possible, values of -1.0 will be appear as the Mach numbers of paths tabulated from the routines PTHTAB and TWNTAB.)

Furthermore, note that throat states (AFTH < AF) can often be at very low pressures, especially for |MCH|=1 or 2. Make sure that the range of valid pressures is sufficient for throat states if expansion exists, otherwise cautions regarding "critical flow rate overestimated" will result and other errors or warnings are possible. Use MCH=0 (perhaps as a default in a PA DEF subblock) to turn off choking detecting and simulation, or MCH=-3 to apply Dyer's method (which does not go below downstream pressures).

Caution: Nonzero values of HCH are nonphysical for gases and mixtures with nonvolatile liquids, but may be required for numerical stability. For the best-estimate values, use zero HCH as long as no convergence or time-step problems are encountered.

Note: The current Mach number, XMA (and also the number printed by PTHTAB and TWNTAB) is an approximation if expansion occurs or if the paths are twinned. Regardless of the value of the Mach number, the path is currently in the choked flow mode if a “c” appears after it in PTHTAB or TWNTAB, even if the path flow rate FR is less than critical flow rate FRC (perhaps due to hysteresis).

Caution: **Choking in Tubes or STUBEs Modeling Reducers.** The default settings for AFTH and MCH may be inappropriate for tubes and STUBEs used to model reducers: outlet flow area less than inlet flow area. (For positive flow, AFI>AFJ. Note that this caution also applies to negative flow, with AFJ>AFI.)
REDUCER connectors are strongly urged as an alternative, so this section is only relevant if that advise is not applicable.
For such tubes and STUBE connectors, use of MCH=-3 is urged.
Otherwise, for non-macro tubes and STUBE connectors, a warning may appear. For duct macro paths, if there are paths in the same duct macro downstream, choked from may be ignored (“DISREGARDED FOR THIS MACRO PATH”), deferring to a downstream path for choked flow detection. If so, the FRC for this within-macro reducer may be lower than FR yet no choking is imposed.
One sign that FRC may be too low for an STUBE modeling a reducer is if PLTH is less than the outlet pressure. Choking calculations are applied as a local calculation independent of whatever might be happening downstream of the current path, which is why they’re applied at the inlet and intentionally disregard downstream conditions. The downstream pressure can’t be “trusted” from the viewpoint of the current path, since it might not be a converged solution yet, or it might be influenced by the lag in a tank, or it might be affected by side flow passages, heat transfer, etc.

The output routine, CHOKETAB (Section 7.11.8), is designed to provide information on the throat state and the choking status of each path.

3.18.3 Choked Flow Mode

When any path’s flow rate exceeds 1+HCH times the critical value, (1+HCH)*FRC, it enters the choked flow mode. In this mode, its normal sensitivities to pressure drop, quality, input parameters (e.g., diameter, length, pump speed, etc.) are temporarily overridden.

When a connector (or tube in STEADY) is choked, the flow rate becomes a weak function of pressure difference: it behaves almost like an MFRSET connector tracking the current critical flow rate. (In theory, the flow rate should be completely independent of downstream pressure for |MCH|=1 or 2, and only a weak function of upstream pressure. As applied within SINDA/FLUINT, the flow rate is a weak function of the pressure *difference* across the path.) It will exit the choked flow mode if its flow rate drops below 1-HCH times the critical flow rate, (1-HCH)*FRC, which can be caused by a decrease in flow rate or by an increase in the current critical flow rate, or both.

When a tube is choked, even its time-dependence (inertia) is overridden as needed to comply with the new constraint of not exceeding critical flow rate. *If choking is expected, a STUBE connector is usually a faster and more stable modeling choice than a tube.*

For tubes and long STUBEs, if choking is expected at the outlet (perhaps because it represents an exhaust), it is best to introduce a "sacrificial" LOSS element (with the same AF, but a small FK of say 0.01) at the exit of that duct. The purpose of this LOSS element is to allow the losses in the tube or STUBE to be modeled explicitly and separately from the pressure drop of the stationary shock at the exit, which will be captured across the LOSS if it chokes. Otherwise, if this sacrificial LOSS were missing, then the full pressure drop of the tube or STUBE itself will replaced by choking at the entrance of the path, perhaps overestimating flow rate in the line.

Confusion sometimes arises when the user expects choking only to occur when the outlet to inlet pressure ratio drops below a critical value (e.g., 0.53 for air). If the resistance of the path is very small, the critical flow rate could be exceeded at much smaller pressure drops than this ratio would indicate.* In fact, if the path were recovering static pressure (e.g., AC>0 for a tube or STUBE, perhaps because AFJ>AFI and both TLEN and FK are small), choking can occur even if the outlet pressure is greater than the inlet. On the other hand, if the resistance of the path is very large (e.g., a large K-factor), choking will not limit the flow rate until the downstream pressure drops much farther than the expected ratio.

3.18.4 Choking in Twinned Paths

If critical flow is detected in any active twinned (slip flow) path, the path is not forced into homogeneous flow (as was done by the obsolete CHOKER routine). Instead, either or both phases may be choked. Usually, either both paths are choked (dispersed flow regimes: bubbly and slug) or just the vapor path is choked (separated regimes: annular and stratified). However, if the flow is bubbly or slug and becomes choked, the user should consider assuming homogeneous flow (see the NOSLIP routine, Section 7.11.1.3).

When one twinned path is choked but the other is not, the flow in the choked twin is insensitive to the flow in the unchoked twin, but the unchoked twin will still obey the slip flow interactions (FD etc.).

Only the MCH and HCH of the primary path are used, so changes to the secondary path's values in logic or expressions are ignored (following the same rules as apply to DH, TLEN, etc.). Also, the returned PLTH, TLTH, and XLTH will be the same for each pair of twins. However, the FRC value will be different for each twin. If the paths are in the homogeneous flow mode, the FRC for the secondary path will be 1.0E30. Otherwise, they will have different values of critical flow rate, the total of which is approximately equal to the total critical flow rate. The critical flow rate should always be treated as approximate in the case of two-phase flow, but when twins are active both the flow rate *and* the apportionment of that flow rate between each phase should be treated with caution.

* For ORIFICE devices in compressible flow, perhaps due to a conservatively estimated vena contracta size, choking can occur even if the downstream pressure is above the throat pressure, PLTH, if |MCH|=1 or 2.

3.19 Multiple-constituent Flows (Mixtures)

Each fluid submodel may contain up to 26 individual fluids, currently limited* to zero or one condensible/volatile species (library fluid, or 7000 or 6000 series fluid as described in Section 3.21.6 and Section 3.21.7, respectively), and zero or more perfect gases, real gases, and simple incompressible liquids (8000 series, and 9000 series, and 6000 series NEVERLIQ fluids as described in Section 3.21.4, Section 3.21.5, and Section 3.21.7 respectively).† One or more of the perfect gases (8000 series fluids) can be soluble in one or more of the liquids, as described in Section 3.23. This section contains data and information unique to submodels containing two or more constituents. See also Sample Problem D (separate volume).

Masses for each constituent are conserved, requiring additional solution procedures and their associated cost compared to single-constituent models. If the concentration of each constituent is essentially constant throughout a single-phase submodel (e.g., one employing air), then use of a single effective substance is therefore strongly recommended.‡ On the other hand, since no single effective substance exists for two-phase mixtures (e.g, air and water), use of multiple-constituent submodels is effectively required in such cases.

3.19.1 Mixture Property Rules

Each fluid is usually assumed to behave ideally and independently when mixed with others. In other words, when real or perfect gases mix, a partial pressure (Dalton's) approach is used.** Transport properties of gases are calculated on the basis of a weighted average of the partial pressure of each constituent. Thermodynamic properties of gases are based on Dalton-Gibb's method, in which each constituent's contributions to properties such as enthalpy are calculated at its partial pressure.

When miscible liquids (and solutes) mix, the Hankinson-Brobst-Thomson method is used to predict mixture density. Conductivities based on mass average, not mole average. Viscosities based on method of Grunberg and Nissan per Reid, Properties of Liquids and Gases, 4th Ed., Eqn. 9-13.1 p. 474, except all interaction factors are assumed zero. Surface tension based on Macleod-Sugden with parachor estimated on the basis of species surface tension and liquid density, per Reid,. Eqn 12-5.3 p. 644. The critical temperature, pressure, molecular weight, and the modified acentric factor are required for all liquids in a mixture in order to predict such effects.

Note that the properties of a liquid mixture are also dependent on whether or not the liquids have been declared to be immiscible (Section 3.21.5).The properties of those portions of liquids that are miscible are treated together using the above correlations or other estimation methods. Liquids that are not miscible are treated separately, using their individual species' properties. For liquid mixtures

* The types of constituents allowed in mixtures is expected to be expanded in future versions.

† Use PR8000 or PR9000 (Section 7.11.2.5) to create gas-only or liquid-only subsets of the library fluids that are compatible with this multiple-constituent modeling requirement.

‡ Refer to PR8000 and PR9000 for simplifying a mixture into a single effective substance (Section 7.11.2.5).

** This approach is suitable for low to moderate pressure mixtures. For high pressure mixtures, Dalton's rule may underpredict pressures and Amagat's partial volume method is more appropriate. However, Amagat's rule is not available currently as an alternative because it lacks simple methods for thermodynamic (enthalpy, entropy, etc.) calculations. Using ifaces to keep gases separate, or reducing tank volumes to increase pressures for the same mass, are both partial work-arounds in the meantime.

consisting of some miscible and some immiscible liquids, the final bulk properties are calculated using a volumetric average. For example, consider 3 liquids A, B, and C, of which A and B are miscible and C is immiscible. The density of the mixture is calculated by dividing the total liquid mass by the volume of fluid C plus the volume of the A-B mixture, which was calculated using the Hankinson-Brobst-Thomson method. Other properties (surface tension, viscosity, conductivity, compliance and sound speed) follow analogous rules.

When a condensible/volatile species is present, the presence of other substances is assumed to have no effect on its boiling point (other than the effect of partial pressures in the gaseous phase).

3.19.2 Usage Summary

This section rephrases information found in this and other chapters regarding input and output options for multiple-constituent flow modeling. It is primarily intended to provide an overview to those users familiar with single-constituent modeling.

3.19.2.1 Input Summary

If only one fluid identifier is provided within the HEADER FLOW DATA subblock, that fluid submodel is referred to as a *single-constituent* or pure substance model, with any fluid type being valid. Otherwise, the user may specify a list of constituents in the HEADER FLOW DATA block, distinguishing each fluid species by an English alphabet letter (A through Z, as explained in Section 3.2.2). Any set of unique letters may be used within each submodel. The same letter may be used in different submodels, just as the same tank or node identifier may exist in more than one submodel. For example, fluid submodels “ALPHA” and “BETA” may each contain a constituent “D”, whether these are the same substance (and therefore the same FPROP DATA block) or not.

Multiple submodels may be used, and each may contain either a single substance or a mixture.

Within each submodel, the same constituent must be consistently referenced using a user-selected letter designator. For this reason, choosing a mnemonic is recommended: “W” for water, “A” for air, etc. The initial mass fraction of each species, relative to its phase, is required to specify the initial thermodynamic state (Section 3.3).

In single-constituent models, of course, there is no need to distinguish between constituents and no letter designators are needed. Nonetheless, they may be used. If no designator is provided (e.g., “FID=nnn”) and multiple constituent data are referenced, the default designator is “A”.

3.19.2.2 Output Summary

Two standard tabular routines are available to present constituent concentration information within each lump: CNSTAB and LMCTAB (Section 7.11.8).

LMCTAB is intended for models with 10 or fewer constituents. It tabulates constituent mass fractions for each lump.

Whereas LMCTAB tabulates by lump, CNSTAB tabulates by constituent. It provides more detailed information about the amounts of all constituents throughout the model.

3.19.2.3 References in Logic and Expressions

The mass and pressure fractions of each gaseous constituent "x" are available for output, inspection, or calculations in logic blocks or in input expressions as translatable parameters XGx and PPGx, respectively. In other words, "XGR(10)" refers to the mass fraction of constituent R in the gas phase of lump 10. Note that this fraction refers to the ratio of gas R mass to the total *gas* mass in the lump, which will be less than the total fluid mass if any liquid is present. (The fraction of gaseous mass to total mass is provided by the quality, or "XL(10)" in this case.) Similarly, "PPGR10" refers to the pressure fraction (*not* the partial pressure) of constituent R in lump 10.*

For liquid constituents, the mass and molar fractions of each liquid constituent "x" is available for inspection as the variables XFx and MFx, respectively. (Note "MF" is a floating point variable!) Once again, note that these fractions refer to the fraction of *liquid phase* properties, and not total properties. For example, "BETA.MFQ(55)" refers to the molar fraction of constituent Q within the liquid phase of lump 55 in fluid submodel BETA.

The sums of XGx over all constituents "x" will be unity for each gas- or vapor-containing lump in the system. The same will be true for each of the sums of PPGx, XFx, and MFx since all these variables represent a different fractional partitioning of the liquid or vapor phases in each lump. Other lump property variables (e.g., XL, AL, DG, DF) refer to total lump properties. Changes to any thermodynamic variable during processor execution can only be made indirectly via calls to the CHGLMP routine (Section 7.11.1.1).

As is explained in Section 7.11.2, property routines and other calls may make use of a translatable fluid identifier "FI." It is possible to refer to fluids directly (e.g., "FI6070" for water), or indirectly (ALPHA.FI for the fluid used in submodel ALPHA). When multiple constituents are present, the indirect reference must be expanded to name the constituent as well. For example, "ALPHA.FIH" refers to constituent H in submodel ALPHA. Refer to Section 7.11.2 for more details.

3.19.3 Two-Phase Mixtures

Multiple-constituent mixtures may consist of mixtures of gases, liquids, or both. While most two-phase single-constituent options and calculations function identically or analogously with two-phase mixtures, there are certain important differences that are the subject of this section.

3.19.3.1 Phase Change

No phase change occurs within two-phase mixtures of multiple constituents unless one of the constituents in the mixture is condensible/volatile (i.e., a library fluid, a 7000 series fluid, or a 6000 series fluid in which NEVERLIQ is optionally not used). Because of the nature of each fluid (refer Section 3.21), the 8000 series gases are assumed to be noncondensible and insoluble, and the 9000 series liquids are assumed to be nonvolatile.

---

* The actual Fortran arrays are two-dimensional arrays named XG, XF, PPG, and MF: the designation "XGH," for example, means the location of species H within the XG array. Refer to INTLMP and especially INTSPE in Section 7.11.1.6 for more information on internal storage and access to the arrays directly.

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 without a condensible/volatile constituent is heated, then both phases will simply get warmer and the quality will remain relatively constant. If a condensible/volatile constituent *is* present in the mixture, then both the temperature *and* the quality will change upon heating.

3.19.3.2 Two-Phase Heat Transfer Calculations in the Absence of Phase Change

Neglecting phase change within the mixture has a significant effect on the calculation of two-phase convective heat transfer. The following discussion applies to the internal correlations employed by HTN, HTNC, and HTNS ties. Refer to Appendix B.4 for more details.

If no flow regime information is available (IPDC is not 6 nor negative for the tied 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. For example, in annular flow, the heat transfer is calculated on the basis of the liquid and vapor film conductances in series. This treatment represents a departure from the single-constituent two-phase options, which take into account boiling or condensation but which neglect flow regime information. Furthermore, no distinction is made between heat addition and heat rejection, since no phase change takes place.

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 (Section 3.19.3.6). The effects of solutes on surface tension is neglected.

3.19.3.3 Two-Phase Heat Transfer Calculations in the Presence of Phase Change

The combination of a condensible/volatile constituent with noncondensible and/or nonvolatile constituents also has a significant effect on the calculation of two-phase convective heat transfer. The following discussion applies to the internal correlations employed by HTN, HTNC, HTNS, and HTP ties. Refer to Appendices B.4, B.5, and B.6 for more details.

In the limit of no condensible/volatile species, the flow-regime based methods of Section 3.19.3.2 are used. In the limit of a pure condensible/volatile species, the single-constituent boiling and condensation correlations are used (Appendices B.2 and 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 condensible gas is condensed in the presence of nonvolatile liquids, the nonvolatile liquids (which are assumed to remain miscible) are not assumed to form a barrier layer per say, but they do contribute to a thicker liquid layer and their properties affect the liquid layer properties. If no noncondensibles are present, then condensation heat transfer is dominated by conduction across this liquid film.

If noncondensibles *are* present in a condensing mixture, an additional resistance is added in series. This resistance is caused by the diffusion of the condensible species through a barrier of noncondensible gases that tends to build up against the liquid layer, such that the interface temperature is colder than the bulk gas temperature (whereas the two are essentially equal in the condensation

of a pure substance). Modeling of this diffusion resistance requires the addition of molecular weight data and diffusion volumes to user-described fluids, as noted in Section 3.19.3.6. Thermodynamic effects, such as the shifting of the apparent saturation temperature due to the partial pressure of the gas, can have even greater effects on the system.

When a volatile liquid is boiled in the presence of noncondensible gases, no effect other than the shifting of the apparent boiling point is modeled. When nonvolatile liquids are also present, an additional minor resistance is present due to diffusion of the volatile substance through the nonvolatile substances. However, this effect is currently neglected, although the nonvolatile substances do have an effect on the liquid properties, and therefore have an indirect effect on the nucleate boiling predictions.

Dissolution and evolution of noncondensible substances can be modeled if desired, as described in Section 3.23.

3.19.3.4 Thermal Equilibrium

When both phases are mixed, thermodynamic equilibrium is assumed by default between phases (as is true in single-constituent flows): the gas and liquid temperatures are the same. While this assumption is certainly less appropriate for mixtures containing noncondensible gases than it is for traditional single-constituent two-phase flows, it is a good starting point and is in fact perfectly acceptable in many applications. The twinned tank features (Section 3.25) are available to help avoid this assumption if it conflicts with the requirements of a particular problem.

Like single-constituent two-phase models, the speeds of sound calculated by the choking utilities (Section 7.11.9.5) and speed of sound routines (Section 7.11.2) will be larger than that of pressure waves propagating through the two-phase portions of a homogeneous tank and tube model. This difference is caused by the assumption of thermal equilibrium between phases, which is not used in the choking or speed of sound routines but *is* used in the calculation of the compressibility of each two-phase tank. In other words, for waterhammer and other fast transient analyses of two-phase multiple-constituent mixtures, note that the effective wave speed will be less and the compressibility greater than a more typical adiabatic phase method would predict. ***Twinned tanks (Section 3.25) must be used to correctly handle wave propagation through two-phase lines.***

3.19.3.5 Difficulties with Minuscule Vapor Mass in Two-phase Tanks

Extremely small void fractions (less than say 0.01%) can be problematic in multiple-constituent tanks and should be avoided if possible if the liquid is incompressible and the tank wall is not compliant. In such cases, the mass of the bubble is nearly insignificant yet its effect on the thermodynamics and hydrodynamics is highly significant. In other words, small errors in the solution can cause noticeable and perhaps unstable fluctuations in the tank pressure. (Note that the internal use of double precision has little effect on this limit.)

The problem is even more acute when the liquid is volatile, especially if its partial pressure is low (e.g., it is cold and in the presence of noncondensible gases). In such a case, the amount of condensible mass in a small bubble becomes diminishingly small, yet it has a significant effect on the behavior of the tank.

This problem can also be experienced when using a pure-substance at extremely low vapor pressures. The code cannot safely resolve qualities below about 10^{-7}: they are at risk of being set to zero if they fall below this threshold, even if the void fraction is significant.

To avoid this problem, either use a compressible liquid (Section 3.21.7) or a compliant tank wall (and perhaps the COMPLQ utility, Section 7.11.9.6).

3.19.3.6 Two-Phase Data Requirements

In order to be able to use CAPIL connectors, flow regime mapping, or the default two-phase heat transfer correlations (which in turn use flow regime mapping), the user must supply surface tension data for all liquid constituents. This data is otherwise optional in those descriptions (Section 3.21.5). For multiple-component two-phase mixtures, such data should be considered mandatory.

Similarly, compliance data must be supplied for all liquid constituents if speed of sound or choking routines are employed.

If a mixture with phase change is modeled or a real gas (6000 series with NEVERLIQ) is present in the mixture, then molecular weight data (MOLW) must be added to 6000 and 7000 series descriptions. Diffusion volume (DIFV, Section 3.21.2.1) information must be added to 6000, 7000, and 8000 series descriptions used within mixtures if HTN, HTNC, HTNS, or HTP ties are used and condensation occurs.

Additional fluid property data is required to model soluble gases, as described in Section 3.22 and Section 3.23.

A fatal execution error will result if use of any property is required that has not been supplied.

3.19.4 Species-Specific Suction Options

For plena or tanks in transient solutions (and STDSTL), one or specific species may be extracted from the lump using species suction options. These options are useful for simulating certain phenomena such as mass transfer and chemical reactions.

Like phase suction, species suction is a statement of a preference but not a condition or constraint: if the requested species is not present, flow will not be stopped. As will be discussed later, *species suction options are not directly applicable to junctions* (unless HLDLMP has been called) nor for tanks during STEADY (aka, “FASTIC”) solutions. In such situations, species-specific suction options are accepted but largely ignored.*

* One possible modeling work-around is to add a plenum next to a junction, and connect the two with MFs of equal and opposite flow rate (i.e., one flowing from the junction to the plenum, and the other from the plenum to the junction). Then set the plenum state to be the junction state minus the constituents that are to be extracted (this state will need to be updated in FLOGIC 0 during the solution to stay current). While mass conservation will be perfect, the same will not be true for energy conservation: the plenum will add or subtract from the network. To minimize this effect, reduce the plenum pressure if necessary to subtract the partial pressure(s) of the missing substance(s).

Phase-specific suction options may be used in combination with species-specific suction, although some interactions between these options exist as noted below.

3.19.4.1 Setting Species-Specific Suction

Species-specific suction options are invoked and controlled via the path STAT flag. For example, to specify that *only* species W is to be extracted from the defined upstream end of a path, set STAT=SPW. To make it affect only the defined downstream end (in case flow reverses), use STAT=RSPW. To apply to both up and downstream ends, use STAT=DSPW. All other species will be accepted through that path.

To exclude species V, use STAT=NSPV (upstream end), STAT=NRSPV (downstream end), or STAT=NDSPV (both ends).

As can be seen from the above examples, the generic format is:

```
STAT = [N][D or R]SPx
```

where x is the species letter identifier within the submodel.

More than one STAT value may be set per path in FLOW DATA as needed to specify the suction status for both phases and all desired species.* The effects are in general cumulative, but the user is warned that additional specifications may override previous ones, and that

1. STAT=NORM removes all previous phase *and* species specific suction actions.
2. STAT=SPx will remove any other species-specific action specified on the upstream end of that path, since the user is implying that only species “x” should pass. Similarly, STAT=RSPx will remove any other species-specific action specified on the downstream end of that path. STAT=DSPx will remove all other restrictions on species suction.
3. STAT=NSPx etc. is cumulative: the user may restrict as many species as necessary. (Of course, restricting all species is the same as restricting none, since such a condition cannot be met--close the path instead. As a reminder, suction options are a request or a preference, and not an unconditional demand.)
4. Phase suction options will be changed if they conflict with species suction options. (This may therefore affect junctions and STEADY tanks, which are otherwise not affected by species specific options.) For example, if the user uses STAT=LS to extract liquid from an upstream lump, but then also uses STAT=SPG, where “G” is a gas-only (perhaps 8000 series perfect gas or 6000 series NEVERLIQ real gas) fluid, then the code will reset the phase suction to be STAT=VS.

* An exception is the PA DEF subblock, in which only one STAT flag may be set, to be applied to all subsequent paths. To undo a previous PA DEF specification, use STAT=NORM in such a subblock.

5. Phase suction options may also be changed as needed to “help” with species suction options. (This may therefore affect junctions and STEADY tanks, which are otherwise not affected by species specific options.) For example, if the user uses STAT=SPL to extract a liquid species from an, and that liquid species is a 9000 series (nonvolatile liquid), then STAT=LS will also be set since that species only exists in the liquid phase anyway.

6. Otherwise, phase suction and species specific suction work together. For example, to let only the gaseous phase of a condensible substance C pass through a path (if available), the user might specify STAT=DVS and STAT=DSPC, which excludes both the liquid phase of the condensible substance and any noncondensible gases that might be present.

During processor execution, path suction options may be reset with the CHGSUC routine, keeping in mind the above rules. Note that several calls to CHGSUC may be made for any one path, analogous to the use of multiple settings of STAT in FLOW DATA.

3.19.4.2 STAT Option Reporting and Accessing

The PTHTAB routine (Section 7.11.8) lists the current STAT flag. Species-specific suction flags are only listed if no phase suction has been set (either by the user or by the program per rules #4 and #5 above). 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). In such cases, the program will simply report “(sp)” to indicate that species specific suction options are active. For more complete details about the species flowing through paths, a separate tabulation style output routine PTCTAB is available (Section 7.11.8).

A routine called XTRACC is available (Section 7.11.1.10) that is analogous to the phase-extraction routine XTRACT. XTRACC reports the mass fraction of any species flowing through a path.

3.19.5 Modeling Considerations and Limitations

This section describes limitations that exist in the current multiple-constituent modeling features. Since this is an area of active development, many of these will be eliminated in future releases.

3.19.5.1 Limitations and Restrictions

As noted above, all constituents (whether vapor or liquid) are assumed to be at the same temperature and total pressure with a lump, conforming to the usual assumption of perfect mixing or thermodynamic equilibrium within lumps.

Axial diffusion of constituents from lump to lump, like the related axial conduction term, is neglected. Advection is assumed to be the dominant mass transport mechanism.

Species-specific suction options (analogous to phase-specific suction options) do not apply to junctions (or to tanks within STEADY). This places limitations on the use of CAPPMP macros in the presence of nonvolatile liquids (or the presence of noncondensible gases if condensation takes place) when the CAPPMP is attached to a junction.

3.19.5.2 Minimum Concentration

Very small mass fractions can be tracked, but a limit exists on the fraction of each phase that can be tracked, and this limit can therefore place a limit on species mass fraction tracking in two-phase systems. Quality in a lump is not allowed to be less than about 10^{-7} before convecting to all liquid), nor greater than about 0.999999 before converting to all vapor/gas. The mass of any species *in the trace phase* may not be conserved if this happens.

This does not imply that low or zero mass fractions are illegal or even undesirable, but merely that the program is unable to track elements in trace *phases*. In fact, just because a constituent is named within a HEADER FLOW DATA subblock does not mean it need appear anywhere in the submodel: its concentration may be zero in every lump. Nevertheless, there is a hidden cost to preserving such “unused species” instead of deleting them from the list. Because of the implicit nature of the solution, extra memory is reserved and extra equations are solved even if those species are never present.

Otherwise, without these issues associated with trace phases, the masses of trace species are normally tracked with great accuracy.

3.19.5.3 Mixing Liquids with Different Densities

When liquids with significantly different densities are mixed within a tank during a transient, a small error might be noted in the tank’s temperature. In other words, as the concentrations of liquids within a hard-filled tank changes, the temperature might drift upwards or downwards as much as one degree in the absence of heat addition. This error is numerically extremely small. It is caused by a numerical approximation in the solution set, and may be reduced or eliminated by lowering DTMAXF. However, in the sample applications analyzed thus far, the impact is negligible. Hence, the accumulation of error caused by the current time step control limits was judged to be acceptable for reasonable solution speeds.

A similar limit exists for single-constituent liquid-filled tanks with extremely large changes in density over small temperature ranges (i.e., with extremely large coefficients of thermal expansion).

3.20 Modeling Tips for Efficient Use

FLUINT efficiency is much more dependent on good modeling practices than is SINDA. Unexpected results and high solution costs are more likely, but these can be controlled by carefully approaching the modeling problem. This section presents some useful tips for getting the most out of a finite computer budget.

3.20.1 Resolution

Spatial resolution—Solution costs for FLUINT increase with almost the square of the number of tanks and junctions (approximately), but plena are almost free. These facts should be weighed into any modeling decision. In other words, the user should use the minimum number of tanks and junctions that will satisfy the modeling requirements. Unjustified high resolution can be costly. Fewer, larger tanks should be a goal.

Also, plena should be used not only for open systems but also wherever the thermodynamic state of a lump does not change appreciably. For example, a single fluid system can be subdivided into separate, smaller submodels using plena if enough locations are found that can be approximated by a plenum. If a 1000 lump submodel were subdivided into two 500 lump submodels, the resulting two submodel system could be solved in about half the time of the former single submodel system. Also, consider using a plenum not only to set a reference pressure but also as a perfect heat exchanger (see also heater junctions and the HTRLMP routine, Section 7.11.1.4.)

One easy way to eliminate unnecessary lumps is to combine paths. There is usually little need to resolve pressure gradients along most lines. For incompressible flow, all fittings along a line can usually be combined algebraically into a single LOSS element or valve model. Furthermore, LOSS elements can be eliminated by applying an FK to a nearby tube or STUBE connector *When creating detailed models that are faithful to the system schematic, users should remember that they may be implicitly asking the program to split analytic hairs.*

Two cautions against overly coarse models are in order. First, the heat transfer rates will be underestimated in single-phase flows if the model is too coarsely discretized and HTNS or HTUS ties are not used. Second, a junction (or STEADY tank) downstream of a high impedance tube or STUBE connector (with UPF less than 1.0) may experience an artificially high pressure drop if the flow is compressible, possibly to the point of causing property warning messages. Both of these problems are due to violations of lumped parameter modeling practices: inadequate discretization to resolve an important temperature gradient in the first case, and inadequate discretization to resolve an important pressure (and therefore density) gradient in the second case.

Temporal resolution—Solution costs for FLUINT also increase dramatically as the resolution of time-dependent changes is increased. Whereas spatial resolution refers to the *number* of elements used to model a system, temporal resolution refers to the *type* of element. A network built entirely from tanks and tubes represents the highest possible temporal resolution (and is able to resolve transient temperature and pressure histories that occur over intervals in the range of microseconds to minutes), whereas a network built entirely from junctions, plena, and connectors represents the lowest possible temporal resolution (i.e., the network responds instantaneously to changes in parameters or boundary conditions). As a reminder, however, no network can be built entirely from junctions, and diabatic junction loops can cause problems.

For most thermal-structural analyses with a characteristic time on the order of minutes to hours, an instantaneous (junction, plenum, connector) model will suffice. If specific events inside of the fluid system need be resolved (such as control transients), the user may need to resort to a more temporally detailed model.

With respect to single-phase liquid systems, if fluid lag times are unimportant and energy can be assumed to propagate instantaneously (at least, compared to time scales of interest) around the network or any portion of the network, use junctions instead of tanks. However, the cost of handling a rigid single-phase tank is not significantly higher than that of a junction, and junction heat transfer calculations are not easily modified. A compliant tank is somewhat more expensive to analyze than a rigid tank, but has numerous advantages. Conversely, the cost of handling two-phase tanks *is* significantly higher than that of junctions—avoid small two-phase tanks unless energy lag times and/or liquid inventory changes are important.

In general, use STUBE connectors instead of tubes if flow rates can change discontinuously or are expected to have zero flow. Tubes should not always be avoided, however. They are actually cheaper to handle than STUBE connectors per solution step, and although they can directly cause a smaller time step, connectors can also restrict the time step indirectly due to their effect on adjacent tanks. Tubes also allow easier custom tailoring of friction and acceleration calculations (AC, FC, FPOW, etc.) in FLOGIC 1. As a rule of thumb, *use tubes with tanks, or use STUBE connectors with junctions for duct models, especially for gaseous and two-phase flows.*

Allowing mixed-temporal resolution models (tanks with STUBE connectors, and junctions with tubes) is intentional. Long, thin lines are likely to have significant inertia (especially if they contain liquid) but negligible volume: junctions and tubes make sense in this case. Short, large-diameter lines have significant volume, but insignificant inertia (especially if they contain gas): tanks and STUBE connectors make sense in this case. However, mixed-temporal resolution cannot be used globally, and can be especially problematic in two-phase flows consisting of pure substances; a conflict of assumptions can lead to numerical problems such as small time steps or artificial time steps. Thus, the rule of thumb presented in the last paragraph is worth paraphrasing: *for modeling pipes containing two-phase non-mixture flows, use tubes with tanks, or use STUBE connectors with junctions.*

"Chemical" resolution—Solution costs for FLUINT increase as the number of constituents within a submodel increase since the mass of each constituent must be conserved. A model with two constituents will cost approximately twice as much to solve as one with a single constituent, a model with three constituents will cost approximately four times as much, etc.

Also, models with mixtures that contain real gases and/or condensible/volatile constituents are considerably more expensive to analyze than ones neglecting phase change or real gas effects. Consideration should be given to assuming that cold liquids are nonvolatile, and that low pressure and/or high temperature gases are perfect.

If concentration gradients do not justify the extra resolution and tracking of individual species, a cost savings can be realized by lumping them into effective fluids in order to decrease the number of constituents tracked.

"Two-phase" resolution—FLUINT offers perhaps the most complete and powerful two-phase thermohydraulic modeling options available. However, *just because they are available doesn't mean they should be used.* Making assumptions, idealizations, and even abstractions are important ingredients in efficient usage. After all, the faster a solution can be run, the more variations on it can be run to glean even more information or to make the code better participate in the design process (such as when using the Solver).

Three capabilities that should be used only when required include: (1) homogeneous versus slip flow, as dictated by the use of twinned paths, (2) perfectly mixed or nonequilibrium two-phase flow, as dictated by the use of twinned tanks, and (3) dissolution/evolution of gases in mixtures, as dictated by the use of MIXTURE DATA blocks.

The above capabilities require substantially larger sets of equations, especially when combined with each other. They also enable additional degrees of freedom that permit a system to "wander" or oscillate, and they frequently require smaller time steps to capture the microscale events deemed important by the user having requested these extra capabilities.

3.20.2 Initial conditions

FLUINT cannot know whether the initial conditions given in the preprocessor were intentionally chosen for a transient analysis or were simply guesses at a steady-state solution. The program may therefore spend a lot of time trying to achieve a steady-state solution from careless initial conditions. Although it is not mandatory, it is well worth the user's time to make sure that "guessed" initial conditions are at least self consistent. This includes making sure that flow rates around loops and into lumps sum to zero, and that zero flow rates are only input where no flow is expected. Also, once a useful steady state has been achieved, it should be saved for future use with RESAVE or SAVPAR calls in the user logic blocks. The use of STEADY can greatly reduce the time spent finding valid initial conditions, but cannot always resolve nonsensical inputs. Also, hysteresis is not uncommon; in some models more than one equally valid steady-state solution can result given different initial conditions.

Excursions beyond the property routine limits are not unusual during steady-state runs, and only indicate problems if they persist or preclude convergence. The routines CHGLMP, CHLMP_MIX, HLDLMP, HLDTANKS, HTRLMP, RELLMP, BALTPL, and SPRIME (for capillary devices) can all be used to control or guide steady-state runs. Also, many fluid systems simply have no steady-state: unlike thermal systems, they may oscillate indefinitely even with fixed boundary conditions.

Preliminary runs—Most of the problems and unknowns discussed above can be resolved by making preliminary runs with coarse models (less spatial or temporal resolution) or with models of subnetworks (broken down using plena as boundary conditions). This will provide both good initial conditions as well as help resolve modeling choices such as how many segments to use for a heat exchanger.

3.20.3 Modeling Practices

Conserve volume—Only tanks have definite volume. However, as mentioned before, tanks should be used only where needed. Junctions can provide much of the same data without slowing down transient analyses. However, the user should never "throw away" volume when using a junction. Rather, it should be added into the volumes of adjacent tanks. For example, tanks are inconvenient to use when modeling a tee, but the volume within the tee (and any connecting lines) should be attributed to nearby tanks through which the flow moves.

Use continuous flow areas—If a small diameter line terminates at a tank or junction and a large diameter line leaves it, the program does not "know" that a sudden change in flow area has occurred. After all, many paths might enter or leave any one lump. Moreover, even if it did detect this area change, it does not have enough information to *model* it. Was it a sudden expansion, for example, or a missing diffuser? FLUINT will attempt to detect such instances in order to produce a one-time warning, since at the very least kinetic energy is not being conserved through such a discontinuity. *If the flow area changes at any place along a flow network, add a REDUCER or EXPANDER to model such a transition explicitly.*

Include accumulators or reservoirs in closed system—In real systems, accumulators serve several purposes such as dampening pressure fluctuations, absorbing volume changes caused by thermal expansions/contractions and by vapor or gas generation, and making up for fluid lost to leaks. Almost all real fluid systems have some type of accumulator, and they should not be neglected in the corresponding fluid model. Both tanks and plena may be used as accumulator models. Plena should be used unless the performance of the accumulator itself is important. (Any junction or tank may be dynamically placed into a boundary state and returned to its normal state using the HLDLMP and RELLMP routines, Section 7.11.1.4.) In general, the presence of a plenum in a closed system will smooth system operation and could reduce the number of solution intervals needed for both transient and steady-state analyses. In fact, *if there are any rigid tanks used in a model of a completely hard-filled (single-phase liquid) system, a plenum* ***must*** *be present.* Otherwise, any thermal expansion or contraction of the fluid would cause extreme pressure fluctuations, possibly exceeding the range of available properties. If tanks are used as accumulators, the VDOT, COMP, and QL values of the tank can be manipulated as necessary to simulate the accumulator response, and other features such as twinned tanks (Section 3.25) and ifaces (Section 3.8) may be useful.

Two-phase lines and heat exchangers—Junctions should be considered instead of tanks in heat exchanger and pipeline models where two-phase flow is expected. *Small, two-phase or vapor (soft) tanks can significantly slow the solution.* However, if liquid inventory changes and/or lag times are important, tanks can be used. In this case, *it is recommended that tubes be used instead of STUBE connectors when modeling two-phase ducts with tanks.* Otherwise, the instantaneous nature of con-

nectors combined with the very time-dependent nature of soft tanks and the nonlinearities of two-phase correlations can lead to instability. Consider setting DTTUBF=0.5 for models where tanks are required, and tubes are used for stability reasons. Otherwise, the tubes will tend to dominate the time step limits. Even with junctions, STUBEs may allow the flow rate to change too quickly, especially in condensers where deceleration (i.e., AC and FR are the same sign) can be dominant. Also, as mentioned in 4.7.5, center-lumped line and heat exchanger segments are much more desirable for two-phase ducts.

Small diameter lines—Arithmetic nodes represent the limit of diffusion nodes of diminishing size. Similarly, junctions represent a replacement for a small tank. A point of common confusion is that STUBE connectors represent the limit of a *short* tube, but not necessarily a *small* one in terms of diameter. In fact, ***the smaller the diameter of a line, the more likely it should be represented by a tube instead of an STUBE connector*** because inertia scales with $\rho L/D$. The smaller the line diameter, the more likely that the line should be modeled with junctions instead of tanks because the volume within the line is of diminishing importance. Although mathematically proper, this leads to confusion since a junction and tube model of a small-diameter line has inertia, but no mass. Finally, it should be noted that the smaller the diameter (compared to the length), the more likely a line must be subdivided to account for gradients in velocities, temperatures, etc.

Constant heat flux vs. constant wall temperature—Text book heat transfer problems are generally broken down into two ideal types: constant heat flux and constant wall temperature. Of course, most real systems are somewhere in between these two extremes but can be modeled as one or the other. With FLUINT, constant heat flux cases (such as electrical dissipative heating) should be modeled using the QL parameter on a single lump if no thermal model is used for heat acquisition (perhaps also changing the default Nusselt number for laminar flow, XNUL). Otherwise, constant wall temperature cases (usually the case during heat rejection) should be handled with the minimum number of heat exchanger segments that will satisfy the modeling accuracy requirements. (HTUS and HTNS ties tolerate very coarse resolution for constant wall temperatures if the flow is single-phase.) These are rules-of-thumb only. As with real systems, *constant heat flux is dangerous if the flow rate through the heated or cooled section drops off unexpectedly.*

3.20.4 Control Systems and User Logic

Ideal control systems—Control systems should be idealized where possible. Fluid submodels are much more sensitive to poor control logic in user logic blocks than are thermal submodels. Temperature (or quality) and pressure control in a line can be achieved simultaneously or independently, as shown in Figure 3-40.

For both temperature and pressure control, a plenum can be inserted in series with the line to be controlled. This method is valid only if the network is constructed such that mass added or subtracted by this plenum is negligibly small, which will be true for incompressible systems with no other plenum or held lump (via the HLDLMP routine, Section 7.11.1.4).

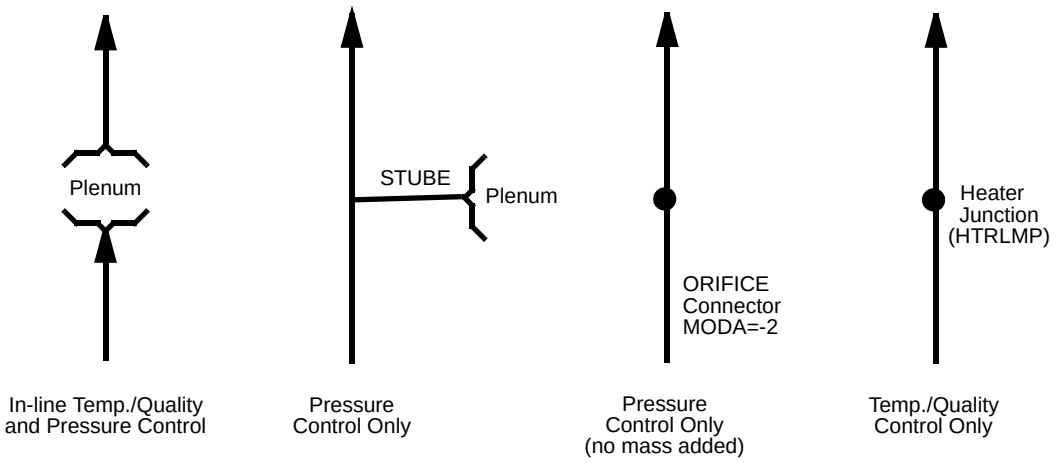


Figure 3-40 Controlling Lines with Plena, Control Valves, and Heater Junctions

To control only pressure without affecting the line temperature or quality, connect a plenum to a line using an STUBE connector. This method is valid only if the network is constructed such that flow rate in this STUBE connector is be negligibly small, which will be true for incompressible systems with no other plenum or held lump (via the HLDLMP routine).

To avoid the use of a plenum for controlling pressure, pressure regulating valves may be used. For UPRVLV and DPRVLV connectors, however control is imperfect: it may be unable to achieve the desired results (as would a real valve in the same situation), and may experience lags and overshoots. The use of a controllable ORIFICE model (MODA=-1 or -2) is preferred, since it more realistically models a control valve with only modest input requirements.

To control temperature but not pressure (i.e., an ideal heat exchanger or mixing point), a heater junction may be used. A heater junction holds a constant enthalpy, adjusting the QDOT as needed (see routines HTRLMP and RELLMP in Section 7.11.1.4).

To control both temperature and pressure without using a plenum (and thereby avoiding adding mass to the system), heater junctions can be used in combination with control valves.

Flow rate control can be achieved using MFRSET connectors, which will maintain the given flow rate independent of flow conditions or pressure gradients. Also useful are VFRSET and PUMP devices. More complicated controls are possible with CTLVLV devices, which can be used to open up and shut off whole subnetworks, though an ORIFICE connector (with MODA=1 or 2) is preferred if a more realistic opening or closing of a valve is needed.

Finally, the user can impose a fixed delta pressure across a path using the HC factor in an STUBE connector.

Feedback control system modeling—One of the capabilities that makes FLUINT unique is its ability to handle a wide variety of feedback control systems via the user logic blocks (usually FLOGIC 0), analogous to the way that SINDA can be used to simulate thermostatic control logic (usually in VARIABLES 1). As alternatives to logic blocks, references can be made in the input expressions to processor (response) variables. Among the parameters that can be varied are pump outputs or shaft speeds, valve or loss element resistance values or open/close status, and heat and volume rates. These parameters can be varied according to any desired logic using any available system variable.

However, there are several cautions that must be observed when modeling flow control systems. First, the control system may be *physically* unstable, causing oscillations, nonconvergence, or property range excursions. Note many real control systems produce oscillations in the system when working properly. Second, the control system may be *numerically* unstable—the real control system would work properly but its representation within FLUINT is inadequate. While one of the principal uses of FLUINT is to evaluate such control systems, the modeling of the action/reaction logic *is the user's responsibility.*

Some of the key points to remember about modeling such feedback control systems within FLUINT are:

1) The input or sensed system variable should not be able to change discontinuously (such as a connector flow rate or a junction state). This would cause steady-state solutions to be troublesome, especially with STEADY (aka, "FASTIC"), and might require high temporal resolution to resolve. If the input variable *can* change discontinuously, it should be fairly independent of the control system reaction (which would be unusual for fluid systems where everything usually affects everything else).

2) The output or reaction of the control system should not be abrupt—it should occur gradually giving the system adequate time to respond to the changes. If a control system is causing problems, the output can be artificially damped. For instance, logic can be added to avoid binary oscillations, or to limit the change in the response each pass through the logic, perhaps by using a running average. While the future time step can only be guessed, the last time step is available in the form of the control constant DTIMUF, and is frequently used for PID control simulations (see also Section 7.7.8). This constant is initially zero.

3) Use idealized control systems wherever possible. Instead of faithfully tracking transient changes, the output of the control system may be controllable directly. For example, use an MFRSET connector to maintain a given flow rate in a bypass branch rather than a detailed model of a three-branch valve.

Smoothing sensitive systems—Many systems are physically realistic but very delicate due to numeric approximations. The user can help make models more rugged by making them more tolerant—by removing some of the more inflexible assumptions.

Placing LOSS elements in parallel with MFRSET devices is an example of smoothing. Because meeting the requirements of an unforgiving (GK=0.0) MFRSET connector can be occasionally difficult for the rest of a system, a good practice is to place a LOSS element with a large FK factor

in parallel with the MFRSET connector to act as a bypass to reduce pressure surges. The FK factor should be sized to allow a small percentage of the total flow to bypass the MFRSET device. This sizing can be done within the FLOGIC 0 block during a steady state run. This is but one example of the usefulness of multiple, parallel paths to model a more complex device.

Another example of smoothing is using compliances in otherwise rigid tanks to allow them to respond to flow changes, as discussed previously. In fact, noncondensible gases are sometimes introduced purposely into liquid systems to achieve the physical equivalent of this numeric method. Refer also to the simulation routine COMPLQ in Section 7.11.9.6, and to the compressible liquid option (COMPLIQ) in user-defined 6000 series FPROP DATA fluids (Section 3.21.7).

Updating Factors for Instant Elements in Logic Blocks—Similar to the concerns voiced above, many FLUINT parameters can cause immediate and perhaps severe changes in network behavior, perhaps to the point of invalidating the assumptions made when updating the parameter. For example, updating the UA on an HTU tie to a junction can instantly (at least, in the next solution step) change the state of that junction to the point where the rationale used to choose that particular UA value (perhaps the junction quality) is no longer valid.

The following is a list of such instant variables, which apply to all connectors and junctions in any solution routine, as well as to tubes and tanks in STEADY (aka, “FASTIC”):

QL . . . . . .on junctions and STEADY tanks
UA, DUPL, DUPN . . . . . .on ties to junctions, STEADY tanks, and arithmetic nodes
DUPI, DUPN, (STAT) . . . . .connectors and STEADY tubes
FC, FPOW AC, HC . . . . . .STUBE connectors and STEADY tubes
DH, DEFF, TLEN, FK. . . . .STUBE connectors and STEADY tubes
UPF, IPDC, WRF . . . . . .STUBE connectors and STEADY tubes
FD, FG, CURV. . . . . .STUBE connectors and STEADY tubes
AF, AFTH, MCH, HCH . . .most connectors and STEADY tubes
ROTR, RADI/J, VAI/J. . . . .most connectors and STEADY tubes
FKI, FKJ, DHI, DHJ . . . . . .REDUCER and EXPANDER connectors
MODEC, AFI, AFJ . . . . . .REDUCER and EXPANDER connectors
TLEN, RAD_A, RAD_R. . .REDUCER and EXPANDER connectors
FK (FKB, PSET, etc.) . . . . .LOSS family of connectors (includes valves)
SMFR. . . . . .MFRSET connectors
SVFR . . . . . .VFRSET connectors
SPD, GPMP, DGDH . . . . . .PUMP, TURBINE, and other turbomachine connectors
RC, CFC, XVH, XVL . . . . .CAPIL connectors, CAPPMP macro NULLs
AORI, MODO, CDIS . . . . .ORIFICE connectors
FK, MODA, etc. . . . . .ORIFICE connectors
OFAC. . . . . .TABULAR connectors
GK, HK, EI, EJ, DK . . . . . .NULL connectors

3.21 Alternative Fluid Descriptions: FPROP DATA

Fluid submodels require the complete description of the working fluid's properties. This description may be provided in one of two ways:

1) by naming one of 19 *standard library fluids* using the appropriate ASHRAE number (see Appendix C) on the HEADER FLOW DATA record, or

2) by creating an FPROP DATA block defining a new *user-defined fluid*, and then naming that fluid on any HEADER FLOW DATA record.

Library fluids (option 1 above) are largely an obsolete choice. Most modern models use option 2: "user-defined" fluid (often supplied by CRTech as an aid) for improved range and accuracy.

A user-defined fluid may be used as the working fluid or as a constituent or reactant in any, all, or none of the fluid submodels. In addition, the user may refer to that fluid in any relevant property routine (Section 7.11.2) in any logic block. The model does not have to contain any fluid submodels to be able to make such property calculations; the FPROP DATA blocks can be used merely as a convenience in thermal submodels.

FPROP DATA blocks were designed such that they can be easily reused in any input file without worrying about unit system conversions. The work involved in defining and debugging a fluid description need only be performed once. Users are encouraged to maintain their own library of alternate fluid descriptions in separate files which can be added to any SINDA/FLUINT input file using the INSERT command. A special output routine PRPMAP is available to assist in preparing and debugging a fluid description (Section 7.11.8), and routines such as PR8000 and PR9000 (Section 7.11.2.5) exist which can be used to create custom and fast but simplified FPROP DATA blocks from other fluids (including mixtures).

Each FPROP DATA block contains the complete description for one working fluid. Up to 30 FPROP DATA blocks may appear anywhere in the input stream after OPTIONS DATA.

Although some of the data is optional within fluid descriptions, the user is strongly encouraged to provide all available data such that the fluid description is more reusable. In particular, additional data is required if the fluid is to be used within mixtures (Section 3.19.3.6).

3.21.1 Introduction

FLUINT submodels require the complete description of the working fluid's properties. This description must include transport properties such as conductivity and viscosity, and *self-consistent* thermodynamic properties such as temperatures, pressures, and densities. As an example of self-consistency, if a temperature is calculated given specific values of pressure and enthalpy, $T_o = T(P_i,H_i)$, then the input value of pressure should result if it were calculated given the original enthalpy and the newly calculated temperature value: $P_i = P(T_o,H_i)$. For real gases and compressible liquids, thermodynamic properties such as enthalpy and entropy must correspond to so-called "departure functions" representing integrations and differentiations of the PVT (pressure-volume-temperature) surface. This requirement for self-consistency does not preclude simplified or idealized descriptions.

Extensive data is required to completely and consistently describe a two-phase working fluid from subcooled liquid, through and around the saturation dome, to superheated and possibly supercritical vapor. For this reason, the descriptions of 19 two-phase refrigerants are available in prestored form (see Appendix C); these fluids are called the *standard library fluids*. Of course, this limited selection does not fulfill all modeling needs. Also, the cost of self-consistency for full functional two-phase descriptions is high; all properties must be calculated as if a liquid could boil and a gas could condense, and potentially restrictive limits on temperatures and pressures are necessary. Therefore, the user may also define the properties of alternate working fluids: *user-defined fluids*.

Four types of alternate fluid types are available. They are identified by using fake four-digit ASHRAE fluid numbers, where the first digit identifies the type:

1) A perfect gas (that cannot condense): “8000 Series.”

   Both the specific heat and transport properties may vary with temperature. In other words, the gas does not have to be calorically perfect. The fluid identifiers for these fluids can range from 8000 to 8999. 8000 series fluids may be used as constituents within mixtures. They may also be used as solutes (Section 3.23).

2) A simple liquid (that cannot boil): “9000 Series.”

   The liquid is assumed to be incompressible, but density as well as transport properties can vary with temperature. The fluid identifiers for these fluids can range from 9000 to 9999. 9000 series fluids may be used as constituents within mixtures. It may optionally contain a saturation curve such that the validity of its “never boil” assumption can be tested.

3) A simplified description of a two-phase fluid: “7000 Series.”

   This option is somewhat like a combination of the above two fluids, with saturation dome and two-phase properties also required. This option is intended to allow a limited description of a two-phase fluid to be quickly added as an alternate fluid using commonly available properties. The fluid identifiers for these fluids can range from 7000 to 7999.

4) An advanced (complete) description of a two-phase fluid or real gas: “6000 Series.”

   The user may replace some or all of the standard library property routines with alternates; in other words, the user may input a functional description of a fluid over a very wide range of properties, including supercritical if desired. It is also the only fluid that can currently employ a thermodynamically compressible liquid phase, or a variable molecular weight option for representing decomposed or ionized gases, or equilibrium collections of reactants as a single fluid. Other important variations include a real gas (no liquid phase) suboption.

   This option is really more of a logic block than a data block. A full link to NIST’s REFPROP program is available, for example. Also, fast tabular descriptions of many fluids have been created using this option. See Section 3.21.7.9 and Section 3.21.7.10. The fluid identifiers for these fluids can range from 6000 to 6999.

The remainder of this section describes each of these types of working fluids.

Perfect gases (8000 series fluids) may also be used as solutes into the liquid phases of other fluids. Such usage requires additional data to be defined within each fluid description. It also requires separate input blocks (MIXTURE DATA, Section 3.22) to be input that describe the solubility of the gas into the liquid. Dissolved gas modeling options are described in Section 3.23.

The utilities PR8000 and PR9000 (Section 7.11.2.5) can be used to convert more complicated fluid descriptions or even mixtures into 8000 and 9000 series fluids, respectively.

3.21.2 Advanced Properties

Most fluid properties required for program execution will be familiar to most users: densities, specific heats, viscosities, etc. Some properties, however, are more advanced and are either more difficult to find in common references, or must be estimated for each fluid. This section describes these advanced inputs. *These properties are not required for most common, simple analyses,* especially if the working fluid is not a mixture of substances. Subsequent sections will define for which modeling purposes these additional parameters are required. Failure to define these properties (if they have no defaults) will cause program termination only if these properties were required by the analysis requested.

The Properties of Gases and Liquids, Fourth Edition, by Robert C. Reid et al is the recommended reference for this section.

3.21.2.1 Diffusion Volumes (DIFV)

When gaseous or two-phase user fluids (6000, 7000, and 8000 series) are used in combination with a model containing a condensible species and one or more noncondensible species, then additional data must be supplied for those fluids if HTN, HTNC, or HTNS convective heat transfer ties are used, or if twinned tanks are used. In those situations, estimates are made the diffusion of one gaseous species through another, or through a mixture of gases.

This additional data takes the form of a single-valued constant, the diffusion volume, which is specific to each fluid. The diffusion volume may be specified as “DIFV” in the subblock formed by the HEADER FPROP DATA statement.

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:

$$D_{AB} = \frac{10^{-3} \cdot T^{1.75} \cdot [(M_A + M_B)/(M_A M_B)]^{1/2}}{P \cdot [(\Sigma v)_A^{1/3} + (\Sigma v)_B^{1/3}]^2}$$

The diffusion volumes can be estimated for various fluids by summing their atomic components (hence the use of the summation symbol Σ above). For example, the v for hydrogen is 2.31, and the v for oxygen is 6.11, so a reasonable estimate of Σv for water is 2.0*2.31 + 1.0*6.11 = 10.73. Actually, the value for water is more precisely known: 13.1. Although this number differs "significantly" from 10.73, such are the range of uncertainties involved in these analyses, which truly deserve the term "best guess." For example, another resource lists 1.98 for hydrogen, 5.48 for oxygen, and 12.7 for water. *The Properties of Gases and Liquids, Fourth Edition*, by Robert C. Reid et al was used for calculation of the library fluid properties, and values from that text are presented in Table 3-14 for use in estimating DIFV for other fluids.

Table 3-14 Diffusion Volumes and Elemental Increments

Element	Diff Vol Incr	Fluid	Diff Vol	Fluid	Diff Vol
C	15.9	He	2.67	CO	18.0
H	2.31	Ne	5.98	CO2	26.9
O	6.11	Ar	16.2	N2O	35.9
N	4.54	Kr	24.5	NH3	20.7
F	14.7	Xe	32.7	H2O	13.1
Cl	21.0	H2	6.12	SF6	71.3
Br	21.9	D2	6.84	Cl2	38.4
I	29.8	N2	18.5	Br2	69.0
S	22.9	O2	16.3	SO2	41.8
Aromatic Ring	-18.3	Air	19.7		
Heterocyclic Ring	-18.3				

3.21.2.2 True Acentric Factor (WTRUE)

The true acentric factor is defined as $\omega = -\log_{10}(P_{sat}/P_{crit}) - 1.0$, where the saturation pressure P_{sat} is evaluated at a reduced temperature (T/T_{crit}) of 0.7. The appendix of *The Properties of Gases and Liquids, Fourth Edition*, by Robert C. Reid et al lists values of the true acentric factor, ω, for over 800 fluids.

Within SINDA/FLUINT, this property is used as part of the Peng-Robinson equation of state, a variation of which is applied to the prediction of spinodals in choked flow (Section 3.18.1.5). Therefore, it is only required for volatile fluids (6000, 7000 series).

3.21.2.3 Modified Acentric Factor (WSRK)

The modified acentric factor (ω_{SRK}) is required for modeling systems containing liquid mixtures. It is also required to compute liquid compressibility in 7000 series fluids. If it is missing and WTRUE (ω) has been supplied, the true acentric factor will be used as an approximation of the modified acentric factor.

A definitive source for the values of ω_{SRK} for over 400 compounds is *The Properties of Gases and Liquids, Fourth Edition,* by Robert C. Reid et al. A subset list of ω_{SRK} for some common fluids, taken from the above reference, is given in Table 3-15. While ammonia and propane are available as library fluids, they are listed for comparison purposes.*

Table 3-15 Modified Acentric Values (WSRK) for Selected Fluids

Fluid	ω_{SRK}	Fluid	ω_{SRK}	Fluid	ω_{SRK}
Water	0.3852	Helium	-0.4766	Acetone	0.3149
Hydrogen†	-0.2324	Neon	-0.0362	Methanol	0.5536
Oxygen	0.0298	Argon	-0.0092	Ethanol	0.6378
Nitrogen†	0.0358	Krypton	-0.0050	Isopropanol	0.6637
Air	-0.0031	Xenon	-0.0023	Methane	0.0074
Carbon Monoxide	0.0295	Flourine	0.0493	Ethane	0.0983
Carbon Dioxide	0.2373	Chlorine	0.0822	Propane	0.1532
Sulfur Dioxide	0.2645	Kerosene	0.4783	n-Butane	0.2008
Ammonia	0.2620	Fuel Oil	0.6009	Isobutane	0.1825
Hydrazine	0.3410	Dowtherm A	0.4084	n-Octane	0.3998
Nitrogen Dioxide	0.8634	Carbon Tet (CCl_4)	0.1875	Ethylene	0.0882
Nitrogen Tetroxide	0.8573	Formaldehyde	0.2656	Propylene	0.1455

† Heed the warning below regarding accuracy of speeds of sound for these fluids.

3.21.2.4 Association Factor (PHI)

The association factor is required when modeling dissolved substances in order to calculate liquid diffusion constants. Wilke and Chang, as referenced in Reid et al, recommend at value of 2.6 for water, 1.9 for ethanol and 1.0 for “unassociated fluids.” Thus, a value of 1.0 is defaulted for most fluids, but can be overwritten by users having advanced knowledge of the association factor for particular fluids.

3.21.2.5 Heat of Formation and Enthalpy at STP

Every type of user-defined fluid that is intended to be used as part of a chemical reaction (Section 3.24) can accept two otherwise optional parameters in the main (top) subblock: heat of formation (HFORM), and enthalpy at standard temperature and pressure (HSTP). The units for both parameters are either J/kg or BTU/lb_m,† depending on the local units of the FPROP DATA block. For example, for water in SI units (which is a liquid‡ at STP):

* In fact, the library descriptions use ω_{SRK} internally. However, as a caution, note that the internal library description of ammonia is slightly more complicated to give better accuracy for that aberrant fluid.

† Data for heat of formation is often given as kcal/mol or kJ/mol, where “mole” is synonymous with “gram-mole” (gmol or gm-mole). FLUINT uses a *mass* basis, instead of a *molar* basis, because the molecular weight is variable for some fluids. The molecular weight may be interpreted as either kg/kmol, lb_m/lb-mol, or gm/gmol, where 1000 gmol = 1 kmol. Also, note that 1 cal = 4.186J = 3.967e-3 BTU.

```
HFORM = -15.874e6    $ J/kg, water as a liquid at STP
```

The molar heat of formation is often denoted as $\Delta H_f°$ in references. The heat of formation is *not* the same as the heat of combustion, whose value depends on a specific reaction. Rather, the heat of formation is a property intrinsic to each substance: it is independent of any reaction. It is zero for elements in their natural state (e.g., gaseous He, Ar, O_2, H_2, N_2, and solid carbon in graphite form). Otherwise, sources such as the CRC handbook contain this data for various molecules. For fuels treated as a single substance, an equivalent HFORM value may need to be calculated.

A heat of formation value (HFORM) is necessary in order for a species to participate in a reaction (Section 3.24.1), even if that value is zero.

HSTP, on the other hand, *is rarely required* as an input since it will be calculated by the program automatically assuming that the enthalpy at STP (1 atmosphere and 298.15K) is within the range of the FPROP DATA block, or that a linear extrapolation to STP is adequate. The purpose of supplying a specific value of HSTP is for when this condition is not true.

For example, if the coldest allowable temperature (TMIN) for a particular fluid's FPROP block is 350K, which is above T_{STP}=298.15K, then FLUINT will calculate HSTP by extrapolation (usually assuming a constant C_p below TMIN):

$$HSTP = H_{P=1\ atm.,\ T=TMIN} - C_{p,T=TMIN}*(TMIN - T_{STP})$$

If such an extrapolated value of HSTP is inappropriate, a value of HSTP may be defined by the user within the FPROP block.

Because only the difference between HFORM and HSTP is relevant (see the QCHEM equations in Section 3.24.1), another use of HSTP is to provide a correction to HFORM.

Normally, however, it is better to leave HSTP as the default value as calculated by the program, and instead to adjust the HFORM value itself and leave a comment in the FPROP block to that effect. For example, the phase of water at STP is liquid, but an 8000 series fluid or a 6000 series NEVERLIQ fluid (real gas) may be chosen to represent water as steam for use in modeling high temperature combustion. In this case, an HFORM value corresponding to liquid water is inappropriate since liquid can never occur using an all-steam subset description of water.

‡ This means that either a liquid-only 9000 series or a two-phase description of steam and water would be required for this HFORM value to be consistent. If instead a simplified 8000 series (perfect gas) or 6000 series NEVERLIQ (real gas) description were input, the HFORM should then be augmented by the heat of vaporization at T_{STD} (25°C).
If the heat of combustion is used as the basis of the fuel's HFORM value, then the HHV (higher heating value or upper heating value or gross energy, assuming liquid water as a product) should be used along with a two-phase water description. If instead the LHV (lower heating value of net energy) is used, then water should be assumed to be a gaseous product (8000 series or 6000 series NEVERLIQ).

In a two-phase water description, the HFORM value should correspond to liquid at STP. A call to the PR8000 utility (Section 7.11.2.5) using such a full two-phase fluid as an input would generate a perfect gas description whose HFORM had been augmented by the heat of vaporization of water at T_{STP}. If the user had used another all-gas description of steam, such a correction should similarly be made, such as:

```
c       HFORM = -15874000.     $ J/kg, as liquid
c the above is the official heat of formation, but since this
c is an intrinsically single-phase gas description, use instead
        HFORM = -13432324.     $ J/kg, as steam
c for operating at temperatures below saturation, use a full
c two-phase description along with the uncorrected HFORM
```

For library fluids, heat of formation is available for propane (290), ammonia (717), and propylene (1270) only. ***Other library fluids cannot be used as reacting species without generating a program abort.***

Many of the non-refrigerant substances available as 6000 blocks (produced from NIST's REFPROP software* and available from www.crtech.com) include HFORM data in appropriate units. If such data is missing, it can be easily supplied by the user by editing the INCLUDE file.

3.21.3 FPROP DATA Blocks: General Input Formats

This subsection applies only to 7000, 8000, and 9000 series fluids. *The 6000 series inputs are really logic blocks rather than data blocks: the following discussion does not apply to them.*

FPROP DATA blocks are composed of two types of subblocks. The first kind is defined by the header record itself, which may extend across several lines of input. It is called the *header subblock,* and every FPROP block has one and only one such subblock. The second kind, the *array subblock,* is identified by an "A" in column 1. The number of array subblocks within an FPROP DATA block is variable and may be zero. Data entry in each subblock continues until a new array subblock or header record is encountered. The rules for data entry within each subblock are described in the next subsections.

There are two ways of describing a property in an FPROP DATA block. The first is *reference style***,** where the user provides a reference temperature, the value of a property at that reference temperature, and a temperature coefficient (which may be zero). The general formula for calculating such a property as a function of temperature is then:

$$X(T) = X + (T - TREF)*XTC$$

* See Section 3.21.7.9 and Section 3.21.7.9**.** REFPROP is actively maintained by NIST (National Institute of Standards and Technology, a division of the US Department of Commerce), so periodic updates should be expected.

where

X(T) . . . . . the calculated value of property X at temperature T
X. . . . . the input reference value of the property at temperature TREF
TREF. . . . . the input reference temperature*
XTC . . . . . the temperature coefficient for X (defaulted to zero meaning constant properties); positive if the property increases with increasing temperature.

There is only one reference temperature input per phase; all properties described in this style must be consistent with this requirement. All reference style descriptions are contained within the HEADER FPROP DATA subblock.

The second method for describing properties is *array style* using a bivariate array of the property versus temperature. Each array style property is input using a separate array subblock, as described below. If both reference style and array style inputs are used for the same variable, a caution is printed and the latest input, always an array subblock, is used.

Arithmetic expressions, perhaps using built-in constants and functions, are valid within these blocks (except in the lower part of 6000 series blocks, which are Fortran). However, *it is strongly recommended that registers **not** be used in these blocks*, since their use makes the FPROP block model dependent. User constants may not be referenced within these data blocks.

3.21.3.1 Header Subblocks

The header subblock in an FPROP DATA block can be used to supply the fluid name (which implies the fluid type), the local unit systems, operating range limits, and any reference style data. The format is:

```
HEADER FPROP DATA, Nnnn [,uid [,abszro] ]
       [,keyword=value][,keyword=value]
```

where:

Nnnn. . . . . the **four** digit number assigned by the user to the fluid about to be described. The first digit (N) must be either an 8 (meaning perfect gas), a 9 (meaning incompressible liquid), a 7 (meaning simplified two-phase), or a 6 (meaning advanced two-phase). (No default.)
uid . . . . . local unit system: SI or ENG (see CONTROL DATA, GLOBAL). Defines the *local* unit system of *this and only this* HEADER block. (Default: value input in CONTROL DATA)
abszro . . . . . absolute zero in the local unit system (see CONTROL DATA, GLOBAL). Defines the *local* unit system of *this and only this* HEADER block. (Default: value input in CONTROL DATA if uid not given, otherwise -459.67 or -273.15 according to the uid value input)

* Note that TREF is *not* the enthalpy reference or basis, which is always absolute zero for 7000, 8000, and 9000 series fluids.

`keyword...` a keyword appropriate to the fluid type being input, such as 'X' and 'XTC' where X is a property name

`value.....` a value appropriate to the previous keyword

Together, the values of uid and abszro define the unit system used within the FPROP DATA block. All data within the block must then be consistent with this chosen system. For example, if uid=ENG and abszro=-459.67, then all temperatures are expected to be in °F, and thermal conductivity is expected in BTU/hr-ft^2-°F. Note that both abszro and uid are optional, but uid *must* be input if abszro is to be input. *It is strongly recommended that the user input* both *uid and abszro.* This not only avoids confusion, it makes FPROP DATA blocks transportable to any other model without unit system collisions. Note that if pressures are required, they are always assumed to be in absolute units.

The remaining inputs in the subblock are dependent on the fluid type.

3.21.3.2 Array Subblocks

Array subblocks are used to input bivariate arrays of a property versus temperature (applicable to all FPROP DATA blocks except 6000 series). The format is:

```
AT, X, t1,v1, t2,v2
       ... ,ti,vi, ... tn,vn
```

where:

`X.........` property name

`t,v ......` temperature/value data pairs

The temperature values must rise monotonically. AT blocks are interpolated within the array limits and extrapolated beyond them. A global maximum of 10000 data points may be used in any one model (e.g., 5000 data pairs).

Remember, all reference style inputs must occur *within* the subblock formed by the HEADER FPROP DATA command, and all AT subblocks occur *after* the HEADER subblock. The appearance of an AT subblock will override any reference style input for the same property.

Only the "A" in "AT" may be placed in column 1; *this first column should be otherwise empty.*

3.21.4 Perfect Gas (8000 Series)

At high temperatures and/or low pressures, almost every substance can be treated as a perfect gas. Such fluid types may also be used as constituents in multiple-constituent (mixture) fluid submodels. They may also be used as solutes if additional data is supplied in both the FPROP block and in one or more MIXTURE DATA blocks (Section 3.22).

3.21.4.1 Description

The equation of state for a perfect gas is:

$$P = \rho RT$$

where:

- P . . . . . the absolute pressure
- ρ . . . . . the density
- R . . . . . the gas constant, $\Re/MW$, where $\Re$ is the universal gas constant and MW is the molecular weight of the gas
- T . . . . . the absolute temperature

The specific enthalpy, h, and internal energy, u, are defined by:

$$h = \int_0^T C_p(\tau)\, d\tau$$

$$u = h - P/\rho = h - RT$$

where:

- T_{ref} . . . . . an arbitrary reference temperature, chosen as absolute zero
- C_p . . . . . the specific heat at constant pressure

Because of the chosen reference temperature, enthalpies will always have positive values. If these gases are used in multiple-constituent submodels that also contain liquid constituents, then the enthalpies will be higher (lower reference state) by an arbitrary constant such that internal numerical schemes can better distinguish between phases. Since only enthalpy differences are important, this shifting has no impact on accuracy or results.

3.21.4.2 Input Formats

A perfect gas is completely described by four quantities: a gas constant, a specific heat at constant pressure, a thermal conductivity, and a dynamic viscosity. The gas constant is given by $\Re/MW$, where $\Re$ is the universal gas constant (8314.34 J/kgmol-K or 1545.33 ft-lb_f/lbmol-R) and MW is the molecular weight. Either the gas constant $\Re/MW$ or the molecular weight MW must be input. The remaining properties may vary with temperature and should be provided in standard units. Refer

to Appendix D for more information on units.

All input properties must be positive. The default temperature range for fluid submodels using a perfect gas description is 10^{-3} to 10^{10}, *or that range over which all input properties are positive.* The program will print an advisory message describing any such limitations at the start of processor execution. (See Appendix C.) Specifying a reasonable value of PMIN can avoid certain types of instabilities.

The formats within the header subblock are:

```
[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]
```

where:

RGAS . . . . . the gas constant. (No default, but MOLW may be provided instead.)

MOLW . . . . . the molecular weight. (No default, but RGAS may be provided instead.)

DIFV . . . . . the diffusion volume (Section 3.21.2.1). Recommended, but not required unless this fluid is used in a model of condensation in a mixture, and HTN, HTNC, HTNS, or HTP ties are used.

TREF . . . . . the reference temperature for all reference style properties.* This is not needed if the all the properties are constants or if they are all input as arrays. (Default: 0.0 in *local* temperature units. It is strongly recommended that the TREF value be stated explicitly.)

TMIN . . . . . the minimum allowable temperature. (Default: 1 degree in *absolute* temperature units, or the minimum implied by other inputs.)

TMAX . . . . . the maximum allowable temperature. (Default: 10^{10} in *absolute* temperature units, or the minimum implied by other inputs.)

PMIN . . . . . the minimum allowable pressure. (Default: 10^{-2} psia in *absolute* units, or the equivalent in SI units. It is strongly recommended that a reasonable value be input to avoid the low default value and avoid instabilities. Values less than 10^{-10} psia are prohibited to avoid mathematical errors.)

* Note that TREF is *not* the enthalpy reference or basis, which is always absolute zero for 7000, 8000, and 9000 series fluids.

PMAX. . . . . the maximum allowable pressure. (Default: 10^{20} psia in *absolute* units, or the equivalent in SI units)

K. . . . . the value of thermal conductivity at TREF. Must be supplied if AT,K subblock is not. (No default.)

KTC . . . . . thermal conductivity temperature coefficient. (Default: 0.0, constant thermal conductivity.)

CP . . . . . the value of specific heat at TREF. Must be supplied if AT,CP subblock is not. (No default.)

CPTC. . . . . specific heat temperature coefficient. (Default: 0.0, constant specific heat.)

V. . . . . the value of dynamic viscosity at TREF. Must be supplied if AT,V subblock is not. (No default.)

VTC . . . . . dynamic viscosity temperature coefficient. (Default: 0.0, constant dynamic viscosity.)

TCRIT . . . . . critical temperature. (No default.) Required when used as a solute.

PCRIT . . . . . critical pressure. (No default.) Required when used as a solute.

HFORM . . . . . heat of formation (J/kg or BTU/lb_m). Required when used in a reaction (Section 3.21.2.5)

HSTP. . . . . enthalpy (J/kg or BTU/lb_m) at standard temperature and pressure (STP). Use if STP is out of range of this fluid, or if the phase at STP is liquid (Section 3.21.2.5)

WSRK. . . . . Modified acentric factor (Section 3.21.2.3). (Default: calculated using the saturation curve generated on the basis PSAT inputs, if available.). Required when used as a solute.

VNB . . . . . Liquid *molar* volume at the normal boiling point (one atmosphere). (No default.) Required only when used as a solute. Units: m^3/kg-mol or ft^3/lb-mol. See guidance below.

VSTAR . . . . . Special molar volume near the critical point. (No default.) Required only when used as a solute. Units: m^3/kg-mol or ft^3/lb-mol. See guidance below.

There are three optional AT subblocks, one for each temperature-varying property:

```
{AT,K, ...}
{AT,CP, ...}
{AT,V, ...}
{AT,PSAT,t1,p1}
```

The PSAT block requires the specification of a single pair of consisting of a temperature and its corresponding saturation pressure. This subblock is required if the gas is used as a solute *and* Raoult's law is used to predict solubility (in MIXTURE DATA, Section 3.22). When input, these two points along with the critical point information (TCRIT, PCRIT) will be used to create a proportionality constant "k" in the equation $Log_{10}(P_{sat}) = k/T_{sat}$ that will be used to predict saturation pressures as a function of temperature.

VNB and VSTAR Values—VNB is used to calculate diffusion coefficients of a solute in a solvent. To calculate it, multiply the specific volume of saturated liquid at 1 atmosphere pressure by the molecular weight. VSTAR is used to calculate the densities of liquids containing this gas as a solute. Its units are the same as VNB. Methods for calculating VSTAR can be found in *The Properties of Gases and Liquids, Fourth Edition,* by Robert C. Reid et al. However, lacking better data VSTAR can be approximated as the specific critical volume multiplied by the molecular weight.

Values of VNB and VSTAR for common soluble or pressurant gases are listed below as a convenience to the user:

Gas	VNB (SI) (m^3/kg-mol)	VNB (ENG) (ft^3/lb-mol)	VSTAR (SI) (m^3/kg-mol)	VSTAR (ENG) (ft^3/lb-mol)
He	31.98E-3	0.5123	54.6E-3	0.874
H_2	28.48E-3	0.4562	64.2E-3	1.03
N_2	34.65E-3	0.5550	90.1E-3	1.44
O_2	28.06E-3	0.4495	73.8E-3	1.18
Air	32.96E-3	0.5279	87.5E-3	1.40
NH_3	24.98E-3	0.4001	70.1E-3	1.12

Example 8000 Series FPROP DATA Blocks:

```
HEADER FPROP DATA,8001,ENG,-460.0     $ GAS 8001
       RGAS           = 1525.0
       V              = 1.0E-4,      VTC            = -1.0E-6
       K              = 22.0,        TREF           = 100.0
AT,CP,22.0,0.1,33.0,0.115,100.0,0.15
C
HEADER FPROP DATA,8888,SI             $ abszro defaults
       CP=2.0, V=1.E-3, K=0.1, RGAS=55.0
C
HEADER FPROP DATA,8717
       TMIN           = 50.0,                        TMAX           = 1000.0
       RGAS           = 8313.34/18.0,                TREF           = 100.0
       V              = 1.0 + 2.0,                   VTC            = -0.004
AT, K, 100.0,         12.0,
       200.0,         14.0,
       300.0,         15.0
AT, CP, 0.0,0.2,      200.0,0.21,    1000.0,0.19
```

3.21.5 Nonvolatile Liquid (9000 Series)

FLUINT models using 9000 series working fluids execute very quickly. Also, this fluid type may be used as a constituent in a multiple-constituent (mixture) fluid submodel, perhaps as a solvent as well. In such mixtures, it might also be specified to be immiscible, perhaps as needed to model solidification or melting. Therefore, the term “liquid” should not be taken too literally: this description is applicable to any working fluid where the density can be assumed constant with respect to changes in pressure.

9000 series fluids are always thermodynamically incompressible. An incompressible assumption does not mean that the density cannot change with temperature. Furthermore, slight compressibility can be added to the hydrodynamic analysis of any working fluid by using compliant tank walls, yielding valid results in water hammer and other fast transient analyses. In fact, an optional descriptor to a 9000 series fluid is its compliance (related to the compressibility) in order to simplify these calculations, and to enable choked flow calculations. Otherwise, the assumption of incompressibility is intrinsic in this working fluid.

Therefore, choking calculations (Section 3.18) and use of the speed of sound routine (VSOSF) have no meaning when used in conjunction with 9000 series fluids unless the compliance data has been provided. Even if compliance data has been provided, 9000 series fluids cannot flash to vapor or two-phase states in low pressure lumps or in virtual path throats, and therefore the critical mass flow rate (FRC) could be greatly over-predicted as a result. *Choking might be missed.*[*]

Compliance data is also required in choked flow calculations if this fluid is used in a multiple-constituent model where gases are also present. In such models, the compliance data is required for all liquid constituents. In such models surface tension data is similarly required for all liquid constituents if flow regime mapping options are required, CAPIL connectors are used, or when more careful estimation of heat transfer conductance is needed in HTN, HTNC, and HTNC ties (which internally access flow regime information if it is available, meaning that IPDC=6 for the pertinent paths).

If compliance or surface tension data is absent, and the use of such data is directly or indirectly required, a fatal execution error will result. Therefore, *it is strongly recommended that this data be provided to the maximum extent possible during the creation of a new 9000 series fluid FPROP DATA block.*

The molecular weight data (MOLW) is strongly recommended but not required unless the fluid is used in a mixture.

* Use the CHKFLASH routine (Section 7.11.1.10) to see if this is a potential issue.

3.21.5.1 Description

Because of the assumption of incompressibility, pressure becomes unimportant in the property calculations for these liquids. There is no difference between the specific heats at constant pressure or constant volume for an incompressible fluid, so the enthalpies for liquids are calculated as follows:

$$u = \int_0^T C_p(\tau)\, d\tau \qquad (C_p \approx C_v)$$

$$h = u + P/\rho$$

Pressures are allowed to range from -1.0E10 to 1.0E10. Enthalpies will almost always be positive values, except for extremely large negative values of pressure. Note that a model using a simple liquid as a working fluid will require almost an order of magnitude less CPU time than one using a standard library fluid that never boils (refer to Section 7.11.2.5).

A 9000 series description may optionally contain a saturation curve such that the validity of its "never boil" assumption can be tested using the CHKFLASH routine (Section 7.11.1.10).

3.21.5.2 Input Formats

An incompressible, nonvolatile liquid is completely described by four quantities: a specific heat at constant pressure, a density, a thermal conductivity, and a dynamic viscosity. Three optional properties are compliance, surface tension, and fractional miscibility. All of these properties may vary with temperature and should be provided in standard units. Refer to Appendix D for more information on units.

All input properties must be positive. The default temperature range for fluid submodels using an incompressible liquid description is 10^{-3} to 10^{10}, *or that range over which all input properties are positive.* The program will print an advisory message describing any such limitations at the start of processor execution. (See Appendix C.)

The formats within the header subblock are:

```
[,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]
[,MISC=R] [,WSRK=R] [,PHI=R]
```

where:

TREF. . . . . the reference temperature for all reference style properties.* This is not needed if the all the properties are constants or if they are all input as arrays. (Default: 0.0)

MOLW. . . . . the molecular weight. (No default. Required if used in a mixture.)

TMIN. . . . . the minimum allowable temperature. (Default: 1 degree in *absolute* temperature units, or the minimum implied by other inputs.)

TMAX. . . . . the maximum allowable temperature. (Default: 10^{10} in *absolute* temperature units, or the minimum implied by other inputs.)

K. . . . . the value of thermal conductivity at TREF. Must be supplied if AT,K subblock is not. (No default.)

KTC . . . . . thermal conductivity temperature coefficient. (Default: 0.0; constant thermal conductivity.)

CP . . . . . the value of specific heat at TREF. Must be supplied if AT,CP subblock is not. (No default.)

CPTC. . . . . specific heat temperature coefficient. (Default: 0.0; constant specific heat.)

V. . . . . the value of dynamic viscosity at TREF. Must be supplied if AT,V subblock is not. (No default.)

VTC . . . . . dynamic viscosity temperature coefficient. (Default: 0.0; constant viscosity.)

D. . . . . the value of density at TREF. Must be supplied if AT,D subblock is not. (No default.)

DTC . . . . . density temperature coefficient. (Default: 0.0, constant density—no thermal expansions/contractions will be considered.)

ST . . . . . the value of surface tension at TREF. Must be supplied if AT,ST subblock is not and its use is required in two-phase multiple-constituent models. Otherwise, this is an optional property. (No default.)

STTC. . . . . surface tension temperature coefficient. (Default: 0.0; constant surface tension.)

COMP. . . . . the value of compliance, (dVOL/dPL)/VOL, at TREF. Must be supplied if AT,COMP subblock is not and its use is required in two-phase multiple-constituent models, choked flow calculations, or waterhammer calculations. Otherwise, this is an optional property. (Default: zero.)

COMPTC . . . . . compliance temperature coefficient. (Default: 0.0, constant compliance.)

TCRIT . . . . . critical temperature. (No default.) Required when used as part of a mixture with other liquids.

PCRIT . . . . . critical pressure. (No default.) Required when used as part of a mixture with other liquids.

HFORM . . . . . heat of formation (J/kg or BTU/lb_m). Required when used in a reaction (Section 3.21.2.5)

* Note that TREF is *not* the enthalpy reference or basis, which is always absolute zero for 7000, 8000, and 9000 series fluids.

`HSTP` . . . . . enthalpy (J/kg or BTU/lb_m) at standard temperature and pressure (STP). Use if STP is out of range of this fluid, or if the phase at STP is gas (Section 3.21.2.5)

`MISC` . . . . . fractional miscibility. The default value is 1.0, meaning fully miscible. A value of zero means completely immiscible with any liquid. See also Section 3.21.5.3.

`WSRK` . . . . . Modified acentric factor (Section 3.21.2.3). (No Default.) Required when used as part of a mixture with other liquids.

`PHI` . . . . . Association factor (Section 3.21.2.4). (Default: 1.0) Required when used as a solvent.

Guidance: Refer to the introduction to this fluid type, to Section 3.19.3.6, Section 7.11.2, Section 7.11.9.5, and to Section 7.11.9.6 for more information on when ST and COMP (surface tension and compliance) data are required or useful.

Caution: The “COMP” in these input blocks has the same units as the tank parameter COMP, and can be used to calculate and update tank values if needed via the COMPLQ routine. *Otherwise these parameters are unrelated*. Tank COMP values are not by default affected by 9000 series COMP inputs.

There are eight optional AT subblocks, one for each temperature-varying property:

```
{AT,K, ...}
{AT,CP, ...}
{AT,V, ...}
{AT,D, ...}
{AT,ST, ...}
{AT,COMP, ...}
{AT,MISC, ...}
{AT, PSAT,  t1, p1 {,t2,p2 {,t3,p3} } }
```

Optional Saturation Pressure-Temperature Relationship. The last property listed above, PSAT, requires additional explanation. Currently, saturation relationships are only used to produce warning messages for insufficient subcooling, which may be a sign of cavitation or choking that is not otherwise being simulated correctly due to the intrinsic assumptions of a 9000 series fluid. In future versions, this data will also be used to check net positive suction head (NPSH) at pump inlets.

If a saturation relationship has been defined, then warnings can be produced if fluid drops below the saturation pressure (indicating a two-phase model or counter design measures are needed) by calling the routine CHKFLASH (Section 7.11.1.10). To define a saturation relationship, the critical temperature and pressure (TCRIT, PCRIT) *must* be specified, along with one, two, or three pairs of temperature/pressure saturation points. At least one point (t1, p1 in the above format) is needed, and three points are recommended if data is available. Pressures must be in absolute units (e.g., psia for local English units). Although saturation pressure p_i must correspond to the previous saturation

temperature t_i, the temperatures need not be input in any order (whereas in other blocks temperatures must be ascending). If enough data (3 points plus the critical point and the zero slope at that point, $dP_{crit}/dT_{crit} = 0$) is available, then the program will represent the saturation curve as:

$$\ln P = A + B/T + C\ln T + DT + ET^2$$

Otherwise, if only two points are available, E=0. If one point is available, D=E=0.

Example 9000 Series FPROP DATA Blocks:

```
HEADER FPROP DATA,9711,SI,0.0$ LIQUID 9711
       TMIN           = 100.0,       TMAX = 1000.0
       TREF           = 400.0,                      $ REF. TEMP
       CP             = 1004.0*0.001                $ SP. HEAT
       D              = 60.0,        DTC = -0.001   $ DENSITY
AT,K,0.0,100.0                                      $ COND. ARRAY
       300.0, 50.0
       500.0,30.0, 700.0,40.0
AT,V,200.0,13.0E-6                                  $ VISC. ARRAY
C
C THE NEXT BLOCK DEFAULTS uid AND abszro:
C
HEADER FPROP DATA,9333,K=1.0,CP=2.0,D=3.0,V=4.0,COMP = 1.0E-5,ST=0.03
```

3.21.5.3 Miscible vs. Immiscible Liquid Mixtures

Mixtures of liquids are assumed by default to be completely miscible (MISC=1.0), invoking correlations to calculate the density and other properties of such a mixture (Section 3.19.1). For certain mixtures (for example, combinations of polar and nonpolar substances such as water and oil), this assumption should be overridden: the liquids should be assumed to be immiscible instead.

Immiscibility is an optional property of any simple incompressible user-defined fluid. For example, consider a simple binary mixture of a two-phase fluid (library, 6000 series, or 7000 series) with a 9000 series fluid. By default, the liquid phases are completely miscible. Specifying the 9000 series fluid as immiscible overrides this default assumption. Note that no change is needed to the other liquid.

Any number of 9000 series fluids may be flagged as immiscible (MISC=0.0), but this does not mean that they are miscible with each other. Rather, *each immiscible substance becomes isolated from all other substances in terms of property calculations.* In other words, that portion (usually 100%) of a substance that is immiscible is assumed to exist separately from all other liquid substances in terms of property calculations. The remaining portions of fluids that are assumed to be miscible are all considered together as a single homogeneous mixture within each lump.

Note that this compartmentalization of liquid substances exists only in terms of fluid properties: there is no physical size assigned to each substance and all liquid species move together at the same velocity. In other words, other than species-specific suction options, it is not possible to model oil droplets moving through an aqueous medium at a different velocity. Twinned tanks and paths (which apply distinct energy/mass and momentum equations) are applicable only to *phases*, not to *species*.

As a simplification, solutes are assumed to have zero solubility in liquids that are assumed immiscible: *immiscibility implies insolubility.* As noted in Section 3.22.1, that if a liquid species has zero solubility for some solute (perhaps because MIXTURE DATA was not input for that pair of substances), then that solute will be nearly insoluble in any miscible mixture containing that liquid. However, if that insoluble liquid species is completely immiscible, then the solute might be highly soluble in the remaining mixture. Immiscibility therefore allows a user to exclude a liquid species from dissolution effects when its solubility is unknown and can be assumed to be negligible.

Temperature-dependent miscibilities and their applications. Normally, miscibility is "all or nothing" ... either a liquid will be 100% miscible or 100% immiscible. However, it is possible to define a *fractional* miscibility as well, perhaps as a function of temperature. There are various modeling applications for this feature, including solidification and precipitation. For example, a molten wax might be miscible at high temperatures, but might separate and solidify at colder temperatures and therefore be immiscible. FLUINT treats such "solids" as highly viscous liquids that are *fully entrained in the liquid flow and that do not accumulate on the duct walls*. Solid phases are not specifically recognized otherwise, so there are limits to the applicability of this feature.

For example, the following subblock defines a fluid that is immiscible below 273.1 degrees but miscible above 273.2 degrees, with a linear transition in between:*

```
AT, MISC,   273.0,      0.0
            273.1,      0.0         $ all ice (immiscible)
            273.2,      1.0         $ all water (miscible)
            273.3,      1.0
```

Note that FPROP property arrays are normally extrapolated beyond their specified range. Thus, four points are input instead of two: extra points below 273.1 and above 273.2 are used to specify constant values ("plateaus") of MISC outside of the transition range.

If the above case were used to model freezing or thawing over the range 273.1 to 273.2, then other properties could similarly be adjusted over that same range. For example, ice density could be used below 273.1, and liquid density above 273.2. A high (*but not unreasonably high*) viscosity could be used to represent ice as a "sluggish liquid."

If a heat of fusion needs to be included, then a spike in the specific heat (C_p) array could be applied, such that the integrated C_p over the transition temperature range yielded the correct heat of fusion. For example, consider a fluid that melts at 10 degrees, with an ice C_p of CPICE (energy/mass-temperature specified elsewhere as a register), a liquid C_p of CPLIQ, and a heat of fusion of HFUS (energy/mass).

* In other words, at 273.15 degrees, exactly half of the mass of this liquid will be in the miscible mixture (along with any other liquid species and solutes), while the other half will be treated separately as the immiscible portion.

If we expand the freezing point to cover the range of 9.9 to 10.1 degrees (a 0.2 degree band), then the input might look like:

```
AT, CP,     9.0,  CPICE
            9.9,  CPICE
            10.0, 0.5*(CPICE+CPLIQ) + 2.*HFUS/(10.1-9.9)
            10.1, CPLIQ
            11.0, CPLIQ
```

The "2.*" in the expression for the C_p at 10 degrees is needed because the resulting spike is integrated as the area of a triangle. In other words, the integral of the above C_p curve from 9.9 to 10.1 degrees should yield HFUS (the heat of fusion) plus 0.2 degrees times the average C_p of ice and liquid.

The current fractional miscibility of any one liquid species can be found using the VMISC routine (Section 7.11.2) which, like other property routines, requires absolute temperature as an input.

3.21.6 Simplified Two-Phase Fluid (7000 Series)

This input option allows the user to describe an alternate two-phase working fluid using parameters readily found in common reference materials. Fluids input using this method must be numbered between 7000 and 7999.

3.21.6.1 Description

Although the range limits are somewhat narrow because of the simplifying assumptions, the properties are reasonably accurate for preliminary design work. This subsection documents the methods used for generating consistent thermodynamic properties given a minimum of simple inputs.

The liquid density, gas specific heat, and all transport properties may be input as constants or as arrays with temperature (i.e., the same method used for 8000 and 9000 fluids). The generation of the P-v-T and P_{sat}-T_{sat} relations require more work because such data is not typically available in reference sources and yet must be derived on the basis of data that *is* available. Also, the analytic form of these relations must be known by the program because of speed considerations.

For the P-v-T surface, the van der Waal's equation is used:

$$P = \frac{RT}{v-b} - \frac{a}{v^2}$$

Although some handbooks list constants a and b, they can be calculated given the critical point for the fluid. This is the best general P-v-T relation that can be calculated on the basis of simple user-provided properties. However, the approximation is not good near or above the critical pressure, and might be inadequate for liquid metals and highly polar molecules.

For the P_{sat}-T_{sat} relation, the following general equation is used:

$$\ln P = A + B/T + C\ln T + DT + ET^2$$

This relationship is adequate for most fluids over a limited range. The five constants in the equation are calculated by the program on the basis of the critical point, and two values each of saturation pressure and heat of vaporization at given saturation temperatures.

FLUINT uses these data and relationships to derive all other required data, including enthalpies and entropies. Thus, using only a few values available in handbooks, the user may adequately describe a two-phase fluid.

Enthalpies, internal energies, and entropies are found by closed form integration of the P-v-T surface from zero density, and numerical integration of the $C_v°$ curve (that is, the curve of C_v versus temperature at the limit as pressure goes to zero) from the reference temperature, which is absolute zero:

$$u = \int_0^T C_v^o(\tau)\, d\tau + \int_0^\rho \left[P - T \left(\frac{\partial P}{\partial T} \right)_\rho \right] \frac{d\rho}{\rho^2}$$

$$h = u + P/\rho$$

These equations can be used to calculate saturated vapor properties. Across the saturation dome, the Clapeyron equation ($h_{fg} = Tv_{fg}(dP/dT)_{sat}$) is used to find the heat of vaporization. The dP/dT derivative is found from the saturation pressure curve. Thus, saturated liquid properties can be calculated. For subcooled states, a $(P-P_{sat})/\rho$ term is added to the enthalpies. Because of these methods, all properties are thermodynamically consistent, and are all referenced ultimately from a low pressure gaseous state.

These approximations are all very good for nonpolar substances at low gas pressures ($P << P_{crit}$). This makes them especially appropriate for cryogens such as nitrogen, hydrogen, and oxygen. For analyses near or above the critical point, a more detailed description is required—refer to the next subsection on 6000 series fluids. Because the above methods are more accurate for a gas than a perfect gas assumption at all points, the user may wish to use this option in place of the 8000 series option for single-phase gas analyses. However, the user should avoid this option for single-phase liquid analyses and should use instead the 9000 series option. For liquids, the methods used in the 7000 series fluids are both more expensive and less accurate than those used in the 9000 series. (Also, the 7000 series should never be viewed as an alternative to the REFPROP-derived options described in Section 3.21.7.9 and Section 3.21.7.10.)

In fact, *the property with the least accuracy is the specific heat for liquids*, which can differ from the true value by as much as 50% for certain inputs. Thus, a good rule of thumb for checking the accuracy of the description for a given fluid is to check the liquid C_p values listed by the PRPMAP routine with tabulated values. For such a simplified description, differences on the order of 10% can be considered an excellent match. The user should not be overly concerned with such large errors in most two-phase analyses because sensible heating is small compared to the heat of vaporization.

For example, Sample Problem D (separate volume) produces nearly identical results whether run with library or very simple 7000 series descriptions of ammonia (a highly polar molecule). This is true even though the specific heats appear to be in error.

Liquid Compressibility—Liquids are thermodynamically incompressible for a 7000 series fluid, but some data regarding the compressibility is often needed nonetheless in order to enable some options (including choked flow detection, compressed liquid density calculations, liquid and two-phase speeds of sound, liquid densities of mixtures, etc.). This additional data consists of a single constant, ω_{SRK}, is the acentric factor for each fluid (Section 3.21.2.3), modified as needed to make the Soave-Redlich-Kwong equation of state match the saturation pressure.

If the desired fluid is not listed in Table 3-15 and the above reference cannot be found, then the true acentric factor for the desired fluid should be input as a good first guess (Section 3.21.2.3). Thus, *if no value of WSRK is given, FLUINT will calculate the true acentric factor* (Section 3.21.2.2) based on the saturation curve that is calculated as a best fit to the input data if a reduced temperature of 0.7 is within the valid range of the fluid (which is nearly always the case). Otherwise, an error message might result from attempts to call or use routines that need this data: VDLC, VSOSF, VSOSFV, and choked flow check routines.

Warning: This approach is simple but approximate, and was chosen because it could encompass the widest range of fluids with the least input—perhaps even none. While the compressed liquid densities are reasonably accurate for all fluids, the corresponding liquid speeds of sound are much less accurate for some fluids. The problem is most acute for cryogens near the saturation line, in which case the liquid speed of sound is overpredicted (because the compressibility is underpredicted), some times by as much as a factor of two or three. The accuracy is good at low temperature and high pressures, but poor at higher temperatures near the saturation line. This problem has been found to be most acute for nitrogen and parahydrogen, but oxygen is curiously immune. Fortunately, the liquid speed of sound contributes little to the two-phase speed of sound except at extremely low qualities. In fact, *the speeds of sound in liquid are almost completely irrelevant for choking calculations even if the local fluid is subcooled liquid*, because in order to choke the flow must first be expanded into a throat, which will almost always result in a two-phase state within the throat except at inordinately high pressures. Still, the user should apply caution in using the VDLC routine to calculate liquid state compliances or local sound speeds. ***For cryogens, faster and more accurate tabular descriptions are available as 6000 series FPROP blocks*** (Section 3.21.7.9 and Section 3.21.7.10**.**

3.21.6.2 Input Formats

The input rules for this option are very similar to the 8000 and 9000 series FPROP options, with data required for each phase as well as for saturation properties.

Most values may be input as constants, linearly varying constants, or arrays vs. temperature in the same style as that used in 8000 and 9000 series FPROP blocks. The only new style of inputs is the *mandatory* AT,DOME section, where two data points are entered to determine the saturation dome. Also, two reference temperatures (one for each phase) may be input instead of one, and range limits may be explicitly defined. Unlike temperatures, *all pressures are assumed to be in absolute units.* All units must be in the unit set named on the HEADER card (see Appendix D).

The formats within the header subblock are:

```
HEADER FPROP DATA, 7nnn, [,uid [,abszro] ]
       [RGAS=R or 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]
```

where:

RGAS . . . . . the gas constant. (No default, but MOLW may be provided instead.)

MOLW . . . . . the molecular weight. (No default, but RGAS may be provided instead.)

DIFV . . . . . the diffusion volume (Section 3.21.2.1). Recommended, but not required unless this fluid is used in a model of condensation in a mixture containing 8000 series gases, and HTN, HTNC, or HTNS ties are used.

TCRIT. . . . . critical temperature. (No default.)

PCRIT. . . . . critical pressure. (No default.)

TREFG. . . . . the reference temperature for all GAS properties that will be input as reference values and coefficients. This is not needed if the all the gas properties are constants (XTC=0.0) or if they are all input as arrays. (Default: 0.0 in *local* temperature units. It is strongly recommended that the TREFG value be stated explicitly.)

TREFL. . . . . the reference temperature for all LIQUID properties that will be input as reference values and coefficients. This is not needed if the all the liquid properties are constants (XTC=0.0) or if they are all input as arrays. (Default: 0.0 in *local* temperature units. It is strongly recommended that the TREFL value be stated explicitly.)

KG. . . . . the value of gas thermal conductivity at TREFG. Must be supplied if AT,KG subblock is not. (No default.)

KGTC . . . . . gas thermal conductivity temperature coefficient. (Default: 0.0, meaning gas thermal conductivity is constant.)

KL. . . . . the value of liquid thermal conductivity at TREFL. Must be supplied if AT,KL subblock is not. (No default.)

KLTC . . . . . liquid thermal conductivity temperature coefficient. (Default: 0.0, meaning liquid thermal conductivity is constant.)

VG. . . . . the value of gas dynamic viscosity at TREFG. Must be supplied if AT,VG subblock is not. (No default.)

VGTC . . . . . gas dynamic viscosity temperature coefficient. (Default: 0.0, meaning gas dynamic viscosity is constant.)

VL . . . . . the value of liquid dynamic viscosity at TREFL. Must be supplied if AT,VL subblock is not. (No default.)

VLTC. . . . . liquid dynamic viscosity temperature coefficient. (Default: 0.0, meaning liquid dynamic viscosity is constant.)

CPG . . . . . the value of the gas specific heat at TREFG *(low pressure value)*. Must be supplied if AT,CPG subblock is not. (No default.)

CPGTC . . . . . gas specific heat temperature coefficient. (Default: 0.0, meaning gas specific heat is constant.)

DL . . . . . the value of saturated liquid density at TREFL. Must be supplied if AT,DL subblock is not. (No default.)

DLTC. . . . . liquid density temperature coefficient. (Default: 0.0, meaning liquid density is constant and no thermal expansions/contractions will be taken into account.)

ST . . . . . liquid to vapor surface tension at TREFL. Must be supplied if AT,ST subblock is not. (No default.)

STTC. . . . . surface tension temperature coefficient. (Default: 0.0, meaning surface tension is constant.)

AVDW. . . . . Van der Waal's coefficient A. This parameter has units of pressure times density squared. In English units, AVDW has units of psf-ft^6/lb$_m$^{2}. (Default: calculated by program.)

BVDW. . . . . Van der Waal's coefficient B. This parameter has units of specific volume. (Default: calculated by program.)

TMIN. . . . . lowest allowed temperature. May be overridden by program. (Default: calculated by program or 1 degree absolute. Lowest allowed pressure is the saturation pressure at this value.)

TGMAX . . . . . highest allowed gas temperature. May be overridden by program. (Default: calculated by program, 1000*TCRIT or 10^{10} maximum.)

PGMAX . . . . . highest allowed gas pressure. May be overridden by program. (Default: PCRIT or 10^{10} maximum. Highest allowed liquid temperature is the saturation temperature at this value.)

HFORM . . . . . heat of formation (J/kg or BTU/lb$_m$). Required when used in a reaction (Section 3.21.2.5)

HSTP. . . . . enthalpy (J/kg or BTU/lb$_m$) at standard temperature and pressure (STP). Use if STP is out of range of this fluid, or if the actual phase at STP is not consistent what this FPROP DATA block predicts (Section 3.21.2.5)

WTRUE . . . . . True acentric factor (Section 3.21.2.2). (Default: calculated using the saturation curve generated on the basis AT, DOME inputs).

WSRK. . . . . Modified acentric factor (Section 3.21.2.3). (Default: WTRUE.) Required for liquid compressibility estimation, or when used as part of a liquid mixture.

PHI . . . . . Association factor (Section 3.21.2.4). (Default: 1.0) Required when used as a solvent.

There is one mandatory and seven optional AT subblocks, one for each temperature-varying property described above:

```
AT,DOME,t1,p1,hfg1,t2,p2,hfg2
{AT,KG, ...}
{AT,KL, ...}
{AT,VG, ...}
{AT,VL, ...}
{AT,CPG, ...}
{AT,DL, ...}
{AT,ST, ...}
```

where:

t1/2 . . . . . temperature values at which dome properties will be given
p1/2. . . . . saturation pressure at temperatures t1, t2
hfg1/2 . . . . . heat of vaporization at temperatures t1, t2

3.21.6.3 7000 Series Fluids: Guidance and Restrictions

Liquid Properties—Note that the C_p of the liquid is never input—it is calculated on the basis of other inputs. Note also that the surface tension is treated as a liquid property, and that the liquid density is for saturated liquid. Liquids are assumed incompressible; the properties are independent of pressure. Also, be warned that liquid enthalpies are frequently negative. This is irrelevant since only enthalpy *differences* are important.

Gas Properties—It is important that the C_p of the gas is the value at zero or very low pressure. This value is sometimes denoted $C_p°$ in tables. As for the remaining gas values (KG and VG), a decision should be made between saturated values and superheated values at constant pressure since these properties are assumed to be independent of pressure. If the analysis includes a variety of pressures with low superheating, the saturation values should be input and an upper temperature limit for the gas should be input. If the analysis takes place at primarily one pressure, the values at that pressure should be input and an upper limit on gas pressure should be input. (Refer to the last example in the next subsection.)

Dome Properties—The saturation dome is characterized by the critical point and two pairs of pressure and heat-of-vaporization values. These last values are input in the mandatory AT,DOME subblock. The choice of which temperature values to use deserves some thought. In general, the temperatures should bracket the range of expected saturation values, *but values at lower temperatures will yield better results than those at higher temperatures.* A good rule of thumb might be to take one value at the lowest available temperature (perhaps just above the freezing point), and the second value at about the temperature anticipated in the analysis. Values near the critical point should be avoided.

Implicit Range Limits—This option is restricted to low pressures compared to the critical point for gases, whereas the pressures in the liquid phase may be arbitrarily high. Conversely, the gas temperature may be arbitrarily high, whereas the liquid temperature is limited by the saturation value at the highest allowed gas pressure. As a rule of thumb, the gas pressure should never exceed one third of the critical value. This limits the liquid temperature to about $0.8T_{crit}$.

While the user may explicitly state the range limits, FLUINT performs additional data checks and may further decrease the valid range to maintain consistency and avoid illegal property values. Note that the program can only avoid *illegal* properties; it is the user's responsibility to avoid *inaccurate* properties by explicitly reducing the range limits. The final set of range limits is enforced by the program, and may be accessed by the user by calling VTMIN, VTGMAX, VTLMAX, and VPGMAX. Note that slight tolerancing is used to avoid the limits, which are assumed to define the valid range exclusively (i.e., the limits themselves represent invalid states).

The top half of Figure 3-41 illustrates the range limits for 7000 series fluids.

3.21.6.4 Example 7000 Series FPROP DATA Blocks

Sample 7000 Series FPROP DATA Block:

```
HEADER FPROP DATA,7001,SI,0.0
       TREFG          = 400.0,       TREFL          = 200.0
       DL             = 100.0,       DLTC           =-0.001,       RGAS= 1525.0
       VG             = 1.0E-4,      VTC            = -1.0E-6
       KG             = 22.0
       ST             = 0.18,        STTC           = -.0035
       TCRIT          = 600.0,       PCRIT          = 1.0E8
       TGMAX          = 800.0,       TMIN           = 100.0,       PGMAX = 1.0E8
AT,KL,  0.0,100.0,     300.0, 50.0,   500.0,30.0
AT,VL,  200.0,13.0E-6
AT,CP,  22.0,0.1,      33.0,0.115,    100.0,0.15
AT,DOME,200.0,1.0E5,2.0E4,            300.0,1.0E7,1.0E4
```

Simplified FPROP DATA Block For Nitrogen:

```
HEADER FPROP DATA,7728,SI,0.0
C
C SIMPLIFIED N2 TWO-PHASE (FROM 63 TO 104K, NEAR 80 K)
C VAPOR PROPERTIES ARE FOR SAT VAPOR (EXCEPT CP WHICH MUST BE FOR LOW PRESSURE GAS)
C
       MOLW = 28.01,  TCRIT          = 126.2,       PCRIT          = 3.4E6
       TREFG          = 80.0
                      CPG            = 1.039E3,     CPGTC          = 0.0
                      KG             = 7.7E-3,      KGTC           = 0.125E-3
                      VG             = 5.6E-6,      VGTC           = 0.076E-6
       TREFL          = 80.0
                      VL             = 148.0E-6,    VLTC           = -5.35E-6
                      KL             = 132.2E-3,    KLTC           = -1.84E-3
                      DL             = 796.24,      DLTC           = -4.739
                      ST             = 8.27E-3,     WSRK           = 0.0358
       TMIN = 63.15,  PGMAX          = 1.0E6,       TGMAX          = 120.0
AT,DOME,63.15,        0.012530E6,    64.739E3+150.45E3
       80.0,          0.13699E6,     79.065E3+116.02E3
C ELSE: 90.0,         0.36071E6,     84.982E3+94.914E3
```

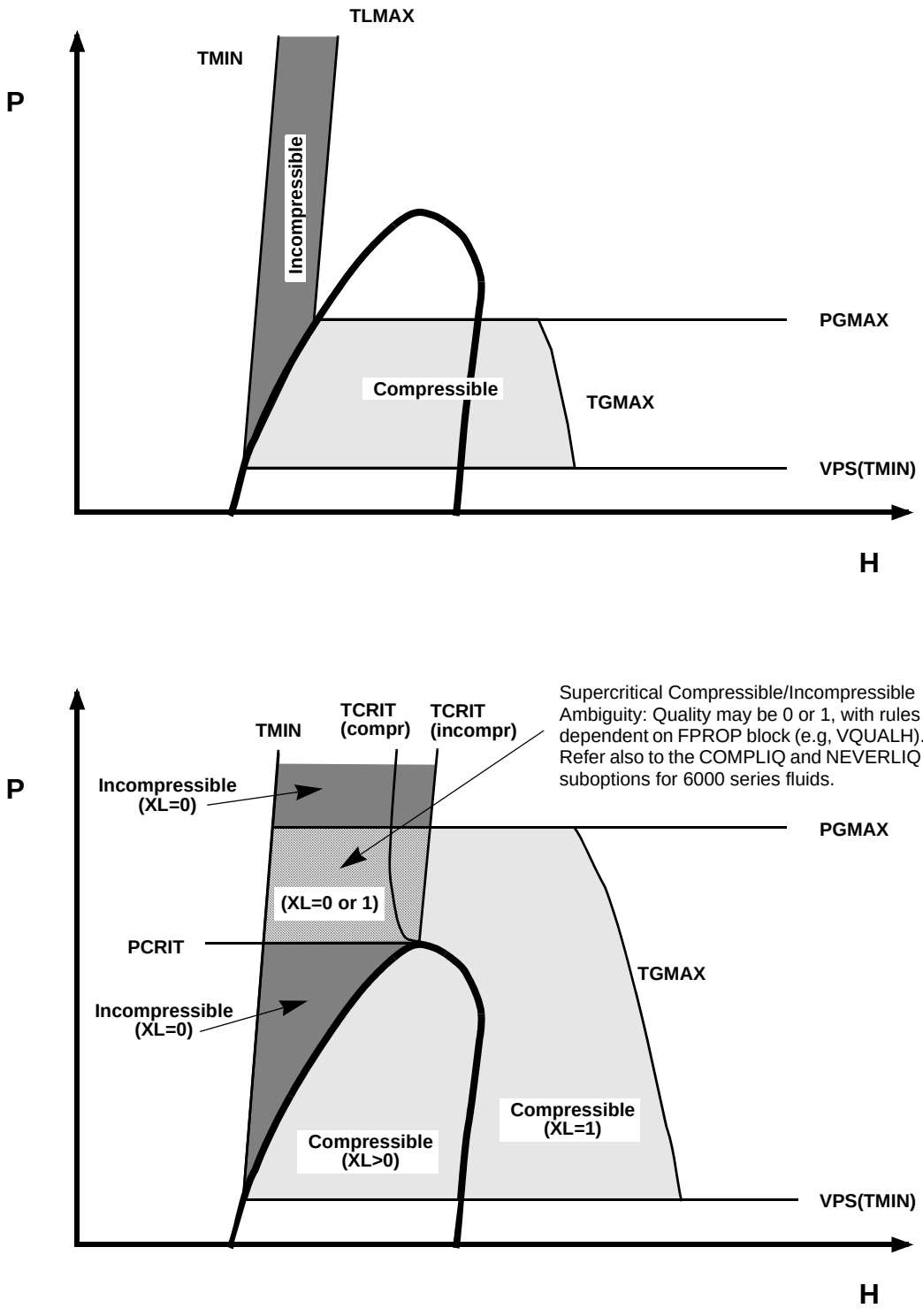


Figure 3-41 Property Limits Illustrated for Two Hypothetical Two-Phase Fluids

Complete FPROP DATA Block For Nitrogen:

```
HEADER FPROP DATA,7728,SI,0.0
C
C MORE COMPLETE N2 TWO-PHASE (FROM 63 TO 104K, NEAR 80 K)
C VAPOR PROPERTIES ARE FOR SAT VAPOR
C (EXCEPT CP WHICH MUST BE FOR LOW PRESSURE GAS)
C
        RGAS            = 8314.34/28.01
        TCRIT           = 126.2
        PCRIT           = 3.4E6
        TREFL           = 80.0,         WSRK            = 0.0358
        ST              = 8.27E-3,      STTC            = 0.0           $ UNKNOWN STTC
        TMIN            = 63.15
        CPG             = 1.039E3,      CPGTC           = 0.0
        PGMAX           = 1.0E6,        HFORM           = 0.0
        TGMAX           = 125.0
AT,VG,  65.0,4.40E-6
        77.36,5.44E-6, 80.0,5.60E-6,  85.0,5.96E-6,  90.0,6.36E-6
        95.0,6.80E-6,  100.0,7.28E-6, 105.0,7.82E-6, 110.0,8.42E-6
        115.0,9.25E-6, 120.0,10.7E-6, 125.0,14.4E-6, 126.2,19.1E-6
AT,KG,  65.0,6.1E-3,   75.0,7.1E-3,   77.36,7.4E-3,  80.0,7.7E-3
        85.0,8.4E-3,   90.0,9.1E-3,   95.0,10.0E-3,  100.0,11.1E-3
        105.0,12.3E-3, 110.0,13.8E-3, 115.0,16.0E-3, 125.0,19.5E-3
C *** ALTERNATE INPUTS FOR VAPOR PROPERTIES ***************************************
C IF RUNNING ANALYSES FOR HIGH TEMP GAS, PRESSURES NEAR 1 ATM,
C THEN USE THE FOLLOWING LINES IN PLACE OF THE ABOVE 10 LINES:
C       PGMAX           = 2.0E5
C       TGMAX           = 500.0
CAT,VG, 65.0,4.40E-6
C       77.36,5.44E-6, 80.0,5.59E-6,  85.0,5.9E-6,   90.0,6.22E-6
C       95.0,6.54E-6,  100.0,6.87E-6, 105.0,7.19E-6, 110.0,7.52E-6
C       115.0,7.83E-6, 120.0,8.15E-6, 125.0,8.0E-6,  126.2,8.65E-6
C       130.0,8.78E-6, 140.0,9.4E-6,  180.0,11.8E-6, 200.0,12.9E-6,
C       220.0,13.9E-6, 240.0,15.0E-6, 260.0,16.0E-6, 280.0,16.9E-6,
C       300.0,17.9E-6, 340.0,19.7E-6, 440.0,23.7E-6, 460.0,24.4E-6,
C       480.0,25.2E-6, 500.0,25.9E-6
CAT,KG, 65.0,6.1E-3,   75.0,7.1E-3,   77.36,7.4E-3,  80.0,7.6E-3,
C       85.0,8.0E-3,   90.0,8.5E-3,   95.0,8.9E-3,   100.0,9.4E-3,
C       105.0,9.8E-3,  110.0,10.3E-3, 115.0,10.7E-3, 125.0,11.7E-3,
C       130.0,12.1E-3, 150.0,13.9E-3, 160.0,14.7E-3, 180.0,16.5E-3,
C       200.0,18.3E-3, 220.0,19.9E-3, 240.0,21.5E-3, 300.0,26.0E-3
C       320.0,27.4E-3, 340.0,28.7E-3, 380.0,31.3E-3, 400.0,32.5E-3,
C       480.0,37.5E-3, 500.0,38.6E-3
CAT,CPG,65.0,1.039E3,320.0,1.039E3,   380.0,1.042E3, 460.0,1.050E3
C       500.0,1.056E3
C *** END ALTERNATE INPUTS ********************************************************
AT,DOME,63.15,          0.012530E6,     64.739E3+150.45E3
        80.0,           0.13699E6,      79.065E3+116.02E3
C ELSE: 90.0,           0.36071E6,      84.982E3+94.914E3
AT,VL,  65.0,274.0E-6, 70.0,217.0E-6, 75.0,177.0E-6, 80.0,148.0E-6,
        85.0,127.0E-6, 90.0,110.0E-6, 95.0,97.2E-6,  100.0,86.9E-6,
        105.0,78.5E-6, 110.0,70.8E-6, 115.0,59.9E-6, 120.0,48.4E-6
        125.0,31.6E-6, 126.2,19.1E-6
AT,KL,  65.0,0.160,    70.0,0.151,    75.0,0.1413,   80.0,0.1322
        85.0,0.1231,   90.0,0.1142,   95.0,0.1053,   100.0,0.0966
        105.0,0.0880,  110.0,0.0795,  115.0,0.0710,  120.0,0.0628
AT,DL,  63.15,867.78,  65.0,860.78,   70.0,840.77,   75.0,819.22,
        80.0,796.24,   85.0,771.87,   90.0,745.99,   95.0,718.38,
        100.0,688.65,  105.0,656.20,  110.0,620.04,  115.0,578.14,
        120.0,525.12,  123.0,480.11,  125.0,431.03,  126.2,314.0
```

3.21.7 Advanced Two-Phase Fluid or Real Gas (6000 Series)

This option allows the user to provide alternate fluid property routines, providing the user with the greatest freedom and the greatest responsibility at the same time. Therefore, new 6000 series inputs should be developed only after other methods have been exhausted, and only by the user with a firm understanding of thermodynamics and FLUINT operation. Fortunately, a large library of these fluid inputs already exists (Section 3.21.7.9 and Section 3.21.7.10), and they are often the preferred option for modern usage. Therefore, "advanced" refers to *creating* such a fluid (as described in this section), and not to *using* an existing description. See www.crtech.com for a listing of available fluids.*

6000 series FPROP DATA blocks are the only ones for which liquids can be thermodynamically compressible, and for which supercritical fluid states are available.

Fluids described with this method must be numbered between 6000 and 6999.

3.21.7.1 Introduction

Unlike previously described FPROP DATA options, this option requires the user to input *functional descriptions* of properties rather than *data values*. (Of course, the term "functional" does not preclude a function that merely looks up data values from a table.) The FPROP DATA block outlines only the *interface* between FLUINT and the user's routines; the user-supplied routines themselves may be provided in SUBROUTINE DATA or in an external user object library that is linked along with the processor library. Used in this manner, a 6000 series FPROP block behaves more like a logic block than a data block.

When using this option, the user must provide logic to replace about 10 FLUINT property routines, and may optionally replace any or all of the remaining set. To replace a routine, the user inputs Fortran statements in a subblock named after the routine that logic replaces (see Section 7.11.2). For example, if the keyword VPS is encountered in column 1, the statements that follow will be used to calculate saturation pressure P given temperature T. The Fortran logic is copied verbatim into a subroutine that is compiled and called from the processor; the logic can be arbitrarily complex and can call any other functions or subroutines supplied by the user. The user must respect the naming conventions used to interface their code with FLUINT (T = temperature, P = pressure, etc. See below). All subblocks for each fluid are placed in a single subroutine; a special subblock named VINIT may be used to input any initializations, declarations, and common blocks.

This property input option is considered advanced because the burden is placed on the user to produce a consistent and fast description of a fluid over a suitable property range. Again, this is actually a logic block disguised as a data block; the same care and debugging that applies to FLOGIC and VARIABLES blocks should be applied here. The use of the routine PRPMAP should be considered mandatory during the development of a 6000 series fluid description. The next subsection describes the speed and consistency requirements, and range limits are discussed in the subsequent subsection.

* All of the fluids listed in NIST's REFPROP program are available as 6000 series FPROP blocks in both incompressible and compressible liquid forms.

Fortunately, many property routine libraries are available that meet the requirements outlined below; the user need only be concerned with establishing the interface to such routines. Also, like any other FPROP DATA subblock, once a description has been developed it may be reused in any model as long as the referenced property routines are available in SUBROUTINE DATA or in an object library.

While the default option enables replacement of any or all property subroutines covering liquid, two-phase, and vapor properties, a suboption is also available for single-phase compressible fluids (real gases and supercritical fluids) that reduces the input requirements. By invoking this “never liquid” suboption, only gas properties need be entered. The presence of a liquid phase and the saturation dome is then eliminated along with the associated input.

3.21.7.2 Fluid Description Requirements

The property calculations should be thermodynamically consistent—the same state should be found as a function of any permutation of valid independent properties. For example, if an enthalpy H is found given temperature T and pressure P, then the same temperature should result when calculated using P and H. Also, performing a cyclic operation on a control volume should bring the state back to its original point. For these reasons, and because property derivatives are evaluated internally, the user must exercise caution when employing curve fits to tabulated thermodynamic data, which might cause numerical problems. Methods for calculating self-consistent sets of properties are beyond the scope of this document. However, a good reference on this subject may be found in Section 2 of *Thermodynamic Properties in SI*, by W. C. Reynolds of the Stanford University Department of Mechanical Engineering. Note also if the liquid phase is assumed to be incompressible; energy and entropy properties input for that phase should be consistent with this formulation.

The speed requirements for executed user code are critical. Unlike FLOGIC and VARIABLES blocks, which are only called a few times, property calculations are called *very* frequently. It is not uncommon to find routines like VH, VPS, and VSV called tens of thousands of times, even in relatively short and simple runs. In fact, *property calculation modules take by far more CPU time than do any other module in FLUINT, even matrix inversion modules!* The user must be therefore especially concerned with the execution speed of the subroutines they supply. This requirement usually precludes the use of generic integration routines; integration should be done by closed form solutions specific to each P-v-T surface. The use of divisions, logarithms, and powers should be minimized, and all iteration techniques must be as honed as possible.

When correctly done, tabular descriptions can be both fast and complete, although somewhat large. Unlike any other fluid description, their accuracy and speed are not very sensitive to the size of the domain. Unfortunately, generating such descriptions is not trivial nor easily described. Rather than attempt to document the algorithms and procedures required to duplicate such descriptions, *the user is advised that tabular descriptions of water, ammonia, hydrogen, nitrogen, oxygen, ethane, argon, and others are available* and should be used directly or as templates for other fluids. The tabular ammonia description executes approximately 2 to 2.5 times faster than the corresponding standard library description. Unless the fluid model is very small, tabular descriptions are the definitive choice for interfaces to large property libraries.

If a 6000 series fluid is used within a working fluid mixture, then a value of molecular weight (MOLW) must be supplied. Furthermore, if this fluid is used in combination with 8000 series fluids in a model in which condensation occurs, the diffusion volume (DIFV) must also be supplied. *The user is strongly encouraged to provide these simple inputs to preserve flexibility.* In addition, property routines must be able to handle extremely small pressures corresponding to small gaseous concentrations, as described in the next subsection.

3.21.7.3 NEVERLIQ (Real Gas) Variation

For the supercritical regime, or for the analysis of real gases where the presence of the dome and a liquid phase are irrelevant, the user can use the NEVERLIQ option described below. Multiple real gases (NEVERLIQ fluids) can be used as constituents in mixtures (Section 3.19), whereas only one volatile/condensible species is otherwise permitted.

The NEVERLIQ option causes all liquid and dome properties to be ignored, and results in a fluid that is always considered to be compressible (XL=1.0), however slightly. The value PCRIT is then used as a lower limit on pressure, and need not correspond to the actual critical pressure of the fluid. In other words, lower “PCRIT” arbitrarily as needed to include lower pressure states.

Otherwise, all dome and liquid properties become irrelevant and need not be input. If input, they will be preprocessed and compiled but never called. (Options may exist in the future that allow a NEVERLIQ decision to be revoked dynamically from within the processor. Therefore, the preprocessing and compilation steps must be successfully passed *if* dome or liquid properties are provided.)

The NEVERLIQ option can be used to modify existing fluid descriptions in a manner that is easily reversible. In other words, the NEVERLIQ command can be added to a existing two-phase and supercritical descriptions to avoid the incompressible/compressible discontinuity problems illustrated in Figure 3-41 (see Section 3.21.7.6 for more details) if the COMPLIQ option (Section 3.21.7.4) is not available. Of course, it can also be used to develop a new description in which liquid and dome properties are never input and the NEVERLIQ option is permanent in that FPROP DATA block.

3.21.7.4 COMPLIQ (Compressible Liquid) Variation

While the NEVERLIQ option *reduces* the amount of inputs required for a 6000 series FPROP DATA block by eliminating the liquid phase and the saturation dome, the COMPLIQ option *increases* the amount of required inputs as needed to eliminate the assumption of an incompressible liquid, and to replace it with a thermodynamically compressible liquid phase.

By default, liquids (quality “XL” equal to zero) in SINDA/FLUINT are thermodynamically incompressible: their thermodynamic and transport properties are functions of temperature only, not pressure.

This expedient treatment eliminates high frequency events, which are usually spurious, in favor of large time steps. However, the assumption of incompressibility causes problems for cold supercritical states (see Section 3.21.7.6 and the bottom of Figure 3-41). If the liquid phase is incompressible, then a discontinuity exists at the critical isotherm and the critical isobar between states defined

as “liquid” and those defined as “vapor.” The resulting zone of ambiguity ($P_{crit} < P < P_{gas,max}$ and $T < T_{crit}$) causes no numerical problems for lumps that operate strictly above P_{crit} or below T_{crit}, although incompressible liquid properties are increasingly inaccurate at high subcooling ($P >> P_{sat}$), especially near the critical point.

When high frequency transients (e.g., waterhammer) *are* desired, or when the assumption of incompressible liquid causes undesirable or artificial responses, a common work-around is to set the tank wall compliance (COMP) to be equal to the compressibility of the liquid, using perhaps the COMPLQ subroutine. This treatment adds *hydrodynamically* compressible liquids and is often adequate for waterhammer-style analyses, but leaves the liquid at its incompressible (pressure-independent) thermodynamic state.

Some systems either traverse the ambiguous zone or even circumnavigate the critical point, such as some Joule-Thomson blow-down systems (and the related Gifford-McMahon refrigeration or heat pump cycle) and transcritical carbon dioxide (CO_2, refrigerant 744) vapor compression cycles. These systems require the elimination of the ambiguities in the cold supercritical region: they require a full thermodynamic treatment of a compressible liquid.

Some cryogenic liquids such as helium and hydrogen are very compressible. Many analyses involving these fluids benefit from if not require full thermodynamic treatment of compressible liquid phases.

Full thermodynamic liquid compressibility can be invoked only in a 6000 series FPROP block, and only by supplying the additional properties required as a result. Once defined, such an FPROP block can be used by itself (as a pure substance) or as a mixture, perhaps combined with incompressible liquids such as 9000 series FPROP blocks.

By default, a 6000 series FPROP block expects an incompressible liquid to be defined, although certain properties such as the compressed liquid density (VDLC) may be optionally input to support calls to the COMPLQ routine or for the calculation of liquid speeds of sound for choking calculations.

The COMPLIQ option, if invoked in the top part of the FPROP block, changes the fluid into a fully compressible liquid. This choice adds to the list of requirements for input properties. For example, VDLC and VHLIQ are no longer optional for a COMPLIQ fluid, they become requirements. For almost all analyses, VSLIQC (liquid entropy) will also be needed, though it is not strictly a requirement since some rare analyses will not employ choking, turbomachinery, or other features that require entropy data.

To completely eliminate the cold supercritical discontinuities, *liquid and vapor properties should be identical along the critical isobar and along the critical isotherm when the COMPLIQ option is employed.*

Many analyses benefit from the assumption of an incompressible liquid phase, which eliminates consideration of many high-frequency events that may not be of current interest. Therefore, the COMPLIQ option is useful but should not be used exclusively.

Notes on commenting out COMPLIQ—Once an FPROP block has been developed for use with the COMPLIQ option, the COMPLIQ line can then be deleted or commented out (e.g., a "C" in column 1) to revert to an incompressible liquid description without further changes to the rest of the block. In this case, saturated liquid properties will be used for the most part (or, in the case of VHLIQ, they will be used as a starting point with incompressible corrections for pressure). For example, if VDLC has been provided instead of VDL, but COMPLIQ is not present or is not detected by the preprocessor, then the code will use VDLC(P_{sat}(T),T, ...) when liquid density (VDL) is required.

On the other hand, if pressure-dependent transport properties (viscosity, conductivity) have been defined, they will be used preferentially even if the liquid is incompressible. Therefore, results produced using an FPROP block with a commented-out COMPLIQ will not be identical to results produced using a traditional incompressible FPROP block that employs only temperature-dependent liquid transport properties.

3.21.7.5 Variable Molecular Weight Option

A special purpose subblock is available for 6000 series FPROP DATA blocks. This optional subblock is called VMOLWPT: molecular weight as a function of temperature and pressure. It may be used in any 6000 series block (i.e., in combination with COMPLIQ or NEVERLIQ). Within the block, the user must define how WMOL is to be calculated as a function of P and/or T. For example (with bold used to identify local keywords):

```
VMOLWPT
        WMOL = 29.15 + (T-291.0)*3.2e-4 + ALOG(P)*5.4e-6
```

If VMOLWPT is provided, any WMOL input in the header subblock is ignored. VMOLWPT is available to be called from user logic blocks for a pure fluid or for a single species. For example:

```
WTEST = VMOLWPT(PL10-PATMOS,TL10-ABSZRO,FI6732)
```

VMOLWPT may be called in user logic even if a constant WMOL was supplied for any one fluid. Therefore, the calling VMOLWPT (instead of VMOLW) represents a "safe" choice if uncertainty exists as to what was defined in the FPROP DATA block.

Possible uses of variable molecular weight fluids include ionized gases and other gases that dissociate, decompose, or recombine in a way that is deterministic with pressure and temperature. Examples include very hot air, nitrogen tetroxide (NTO), and the products of combustion (that can be treated as a single effective substance).

"Deterministic with pressure and temperature" means no hysteresis is possible: the fluid state can only be a function of *current* temperature and pressure, and not a function of prior processing. For example, NO_2 and N_2O_4 ("NTO") are dimers of each other: gas containing one will usually contain the other as well. At high temperatures, the NO_2 form will dominate, whereas the form N_2O_4 appears at cold temperatures. If hot NTO gas is cooled very quickly, the concentrations of N_2O_4

will be less than what thermodynamic equilibrium predicts. Such a quick-cooled mixture is therefore *not* deterministic, whereas the equilibrium state *is* deterministic, but may represent an approximation to reality.

"Deterministic" does not, however, imply that chemical equilibrium is *required*. For example, a fluid description could be developed using chemical equilibrium calculations above a certain threshold temperature, frozen (nonreacting) concentrations below a lower temperature, and some transition could exist between those two temperatures. Such a mixture might, however, have a presumed directionality in its use: it might only be useful when applied to a heating process, and a separate description (FPROP DATA block) might be required for a process in which the fluid is instead started at a hot temperature and then cooled. FLUINT makes no presumptions about processing and does not enforce any restriction on how the fluid is employed within a fluid submodel: it only requires that a single property value be returned for a given temperature and pressure. Therefore, the user must exercise caution when applying such a partially frozen fluid description, including perhaps adding warnings to FPROP DATA file such that third parties do not accidentally use that fluid description for an inappropriate analysis.

Variable molecular weight fluids can be used to model products of combustion, perhaps in combination with the options described in Section 3.24. For example, an FPROP DATA block could be built using equilibrium concentrations of the hot products of the stoichiometric combustion of a fuel and an oxidizer. As a first approximation, this stoichiometric equilibrium pseudo-fluid can be combined with excess fuel or oxidizer in the nonreacting outlet stream, making the assumption that the effects of excess fuel or oxidizer do not greatly affect the concentrations of combustion product species in the outlet of the combustion chamber (which might include flowing through turbine stages, as an example). This assumption of "stoichiometric combustion products plus excess fuel or oxidizer" is not universally valid and must be checked in each application. For example, treating excess flows of complex hydrocarbon fuels (e.g., kerosene) as being at equilibrium is not appropriate, since those fuels will decompose at higher temperatures even if they don't combust.

3.21.7.6 Regimes and Range Limits

Range limits are user-provided, except that temperatures must be within 10^{-3} to 10^{10}, and pressures must be less than 10^{10}. While the program will help enforce user-provided limits, *it is the user's responsibility to make sure that all properties are valid over the selected range, and that any necessary error trapping and handling logic is in place*. Unlike other user-described fluids, no additional checks will be made on the calculations provided. A error/warning message routine, UDFERR, has been provided to allow the user to write to the output file in a controlled manner. Note that slight tolerancing is used to avoid the limits, which are assumed to define the valid range exclusively (i.e., the input limits themselves represent invalid states).

Figure 3-41 is provided as an illustration of the property range limits for two hypothetical fluids, both employing the assumption of a thermodynamically incompressible liquid phase. For consistency, note that TGMAX is greater than or equal to TLMAX, and PGMAX is equal to VPS(TLMAX) if TLMAX is less than TCRIT.

If the valid region includes the critical point, all appropriate property routines should function at that point. This is important because the program immediately calculates and stores critical densities and enthalpies for repeated use thereafter in the program.

Cold Supercritical Ambiguity and Discontinuities—This is the only fluid type that is allowed to operate above both supercritical temperature and pressure (if the user allows). **This creates an ambiguity for P>PCRIT and T<TCRIT (the cold supercritical regime), as depicted in the bottom half of Figure 3-41, if the liquid phase is incompressible ... that is, if the COMPLIQ option is not employed.**

The fluid will be treated as a compressible gas (XL=1.0) if the temperature is supercritical and the pressure is not above PGMAX: XL=1 if T > TCRIT and P < PGMAX. On the other hand, the fluid will be treated as an incompressible liquid* if the temperature is subcritical and if *either* the pressure exceeds PGMAX *or* is subcritical: XL=0.0 if T < TCRIT and either P > PGMAX or P < PCRIT. In other words, if PGMAX > PCRIT expect a region of ambiguity between compressible and incompressible formulations at supercritical pressures and subcritical temperatures. Tanks and plena can be initialized to either a quality of 0.0 or 1.0 in this ambiguous region if the liquid is incompressible.† If a tank is initialized to 1.0 and the pressure drops below PCRIT or raises above PGMAX, the quality will suddenly change to 0.0 with a potential for artificial responses as the new formulation is applied discontinuously with sudden gains or losses of mass and energy. For junctions or STEADY (“FASTIC”) tanks, the rules depend on those of the VQUALH and VQUALS logic subblocks provided by the user in the FPROP block.‡

To avoid this discontinuity, consider either the NEVERLIQ or COMPLIQ options described above. Otherwise, if those options are not available or applicable, consider setting PGMAX < PCRIT if high pressure gas states can be neglected. For fluid dropping in pressure (e.g., throttling) from cold supercritical pressures to subcritical pressures, if XL=1.0 upstream then avoid stepping through this regime (e.g., in a highly discretized duct macro): jump past it instead within a single path.

CAUTION: Low Pressures in Gaseous Mixtures—If the fluid is to be used in combination with other gaseous fluids in a mixture, then the user-provided property routines must be able to handle (as inputs) diminishingly small pressures corresponding to minute concentrations of this fluid. For example, the user-supplied routines (including but not limited to VH, VS, VSV, VCONDF, VVISCF, etc.) must return meaningful answers when absolute pressures approaching zero are provided as inputs. In other words, the normal lower pressure limit (saturation pressure at TMIN, or PCRIT if the NEVERLIQ option is used) only apply if this fluid is *not* used in a mixture. Since fluid descriptions are intended to be reusable, *the user is strongly encouraged not to limit future users of the FPROP block by making sure the developed fluid description is compatible with mixtures.*

Refer to Section C.4 for more information on range limits of mixtures.

---

* An incompressible formulation means most properties are extrapolated from saturated liquid properties. This does not preclude the use of compliance (e.g, using the COMPLQ routine), but such compliances affect the tank wall and not the thermodynamic state within the tank. The COMPLIQ option (Section 3.21.7.4), on the other hand, provides for a fully thermodynamic treatment of the compressibility of the liquid, and thereby eliminates the cold supercritical ambiguity.

† Except for mixtures involving noncondensible gases, all states with P>PCRIT are described as XL=1.0 if for COMPLIQ fluids.

‡ Junctions are always assumed to have XL=1.0 within the ambiguous region.

3.21.7.7 Input Formats

With the exception of a few data points such as the critical point and range limits, almost all information is supplied in the form of a subroutine or function call that is written directly into a Fortran subroutine by the preprocessor. As part of this interface, standard variable names are reserved (in these types of FPROP blocks only) as follows:

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 saturation pressure w.r.t. temperature (dP/dT)sat
DPARG . . . . . .RESERVED
SPARG. . . . . .RESERVED
INTARG . . . . . .RESERVED

The user may perform any internal unit conversions in the subblocks, as long as the incoming and outgoing values are in the unit set named on the HEADER card.

Calculations and subroutine calls may be placed within appropriate subblocks named after the FLUINT subroutine they will override (see Section 7.11.2). All such “V-style” subblocks are called functional subblocks. As with AT subblocks, these must occur after any keyword-style inputs. All FLUINT property routines may be overridden in this manner; but most can be are derived from the basic inputs provided by the user. For this reason, some routines are required whereas others are optional.

Information and code entered by the user in the subblocks are written verbatim to a central Fortran subroutine for each fluid (named UF6nnn, where 6nnn is the name input on the FPROP card). For example, to provide the thermal conductivity of the vapor phase, the user would enter Fortran instructions under the subblock named VCONDV, where the “V” starts in column 1 and denotes the start of a new subblock. Subblocks may be input in any order (with the exception of the special VINIT subblock described below, which must occur first if it is to be used).

Because all instructions and calls will be written to a single routine, it is the user’s responsibility to avoid collisions between variables.

Calculations and subroutine calls should be performed in the set of units selected on the HEADER FPROP DATA card. (Pressures must be in absolute units.) FLUINT will provide the necessary unit conversions between these calculations and the unit system of the current master model. The user must make sure the property routines return units consistent with one of the sets listed in Appendix D.

A special block is provided that will be placed at the start of UF6nnn. This block, named VINIT, may be used to initialize or reset any parameters or common block variables needed for the fluid in question. This block is inserted between the UF6nnn declarations and the executable statements, and so can include both types of statements if desired, including DATA statements. Data common to all calculations may be placed or initialized here, or calls may be made here to a user initialization routine. Any executable statements place here will be executed *every time a property calculation for fluid 6nnn is made.* For this reason, any one-time initializations should be controlled by logic to avoid unnecessary repetition.

Header Subblock Format:

```
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]
```

where:

- `NEVERLIQ...` the fluid is to be treated as a gas (XL=1.0) everywhere, and liquid and dome properties may be input but are not required and will not be used.
- `COMPLIQ....` the liquid phase is to be treated as thermodynamically compressible, and additional liquid properties are required such as VDLC and VHLIQ.
- `PCRIT.....` critical pressure, or lowest allowable pressure if the NEVERLIQ option is used. (No default.)
- `TCRIT.....` critical temperature. (No default. Required unless NEVERLIQ.)
- `MOLW.....` the molecular weight. Strongly recommended, but not required unless this fluid is used in a mixture.
- `DIFV.....` the diffusion volume (Section 3.21.2.1). Recommended, but not required unless this fluid is used in a model of condensation in a mixture containing 8000 series fluids, and HTN, HTNC, HTNS, or HTP ties are used.

TMIN. . . . . lowest allowed temperature. (Unless the NEVERLIQ option is used, the lowest allowed pressure is the saturation pressure at this value or 1 degree absolute.)

TGMAX . . . . . highest allowed gas temperature or 10^{10} maximum.

PGMAX . . . . . highest allowed gas pressure. (Highest allowed liquid temperature is the minimum of the saturation temperature at this value or TCRIT or TGMAX or 10^{10} maximum.)

HFORM . . . . . heat of formation (J/kg or BTU/lb_m). Required when used in a reaction (Section 3.21.2.5)

HSTP. . . . . enthalpy (J/kg or BTU/lb_m) at standard temperature and pressure (STP). Use if STP is out of range of this fluid, or if the actual phase at STP is not consistent what this FPROP DATA block predicts (Section 3.21.2.5)

WTRUE . . . . . True acentric factor (Section 3.21.2.2). (Default: calculated using the saturation curve generated on the basis VPS inputs.)

WSRK. . . . . Modified acentric factor (Section 3.21.2.3). (Default: WTRUE.) Required when used as part of a liquid mixture.

PHI . . . . . Association factor (Section 3.21.2.4). (Default: 1.0) Required when used as a solvent.

Optional First Functional Subblock, Declarations and initializations:

VINIT . . . . . Inserted in routine UF6nnn between declaration statements and first executable statements. May contain array dimensioning, common blocks, DATA statements, and executable lines.

Required Functional Subblocks:

VSV . . . . . Calculates: vapor specific volume V
Given: pressure P and temperature T

VH . . . . . Calculates: vapor enthalpy H
Given: pressure P and temperature T and vapor sp. volume V

VVISCV . . . . . Calculates: vapor dynamic viscosity VISC
Given: pressure P and temperature T

VCONDV . . . . Calculates: vapor conductivity COND
Given: pressure P and temperature T

Required Subblocks (unless the NEVERLIQ option is used):

VPS. . . . . Calculates: saturation pressure P
Given: temperature T
VDPDTT . . . . Calculates: saturation derivative dP/dT
Given: temperature T.
(Must correspond to VPS values.)
VDL . . . . . Calculates: saturated liquid density D
Given: temperature T
VVISCF . . . . . Calculates: liquid dynamic viscosity VISC
Given: temperature T
VCONDF . . . . Calculates: liquid conductivity COND
Given: temperature T
VST. . . . . Calculates: liquid/vapor surface tension ST
Given: temperature T

Required Subblocks (if the COMPLIQ option is used):

VDLC . . . . . Calculates: subcooled (compressed) liquid density D
Given: pressure P and temperature T
(VDL is no longer required if COMPLIQ is used.)
VHLIQ . . . . . Calculates: subcooled (compressed) liquid enthalpy H
Given: pressure P and temperature T

Optional Functional Subblocks, Additional Properties:

VS. . . . . Calculates: vapor entropy S
Given: temperature T and vapor sp. volume V
VSOS . . . . . Calculates: vapor speed of sound SOS
Given: pressure P and temperature T
(optional if VS provided)
VTAV1 . . . . . Calculates: vapor specific volume V and temperature T
Given: pressure P and entropy S
(Must correspond to VS and VSV values. Optional if VS provided)
VQUALS . . . . Calculates: quality X
Given: pressure P and entropy S
(optional if VS provided but recommended nonetheless)
VDLC . . . . . Calculates: compressed liquid density D
Given: pressure P and temperature T.
Must correspond to VDL at saturation.
VMOLWPT . . Calculates: molecular weight WMOL (see Section 3.21.7.5).
Given: pressure P and temperature T.
(If input, overrides WMOL value in header subblock.)

Optional Subblocks, Additional Properties (if the COMPLIQ option is used):

VSLIQC. . . . . Calculates subcooled (compressed) liquid entropy S
Given: pressure P and temperature T
(*Technically* not required, but it is needed in almost all analyses.)

Optional Functional Subblocks, More Appropriate Methods:

VTAV2. . . . . Calculates: vapor specific volume V and temperature T
Given: pressure P and enthalpy H
(Must correspond to VH and VSV values.)

VTS . . . . . Calculates: saturation temperature T
Given: pressure P.
(Must correspond to VPS values, ***except input pressures exceeding the allowable saturation range should result in returned temperatures that are at those limits without exceeding them***.)

VDPDT . . . . . Calculates: saturation derivative DPDT
Given: pressure P.
(Must correspond to VPS and VDPDTT values.)

VCPV. . . . . Calculates: Vapor specific heat CP
Given: pressure P and temperature T.
(Must correspond to VH values.)

VCPF . . . . . Calculates: Liquid specific heat CP
Given: pressure P and temperature T.
(Must correspond to VH, VHFG values.)

VHLIQ. . . . . Calculates: Liquid enthalpy H
Given: pressure P and temperature T.
(Must correspond to VH, VHFG values.)

VTLIQ . . . . . Calculates: Liquid temperature T
Given: pressure P and enthalpy H.
(Must correspond to VH and VHFG values.)

VULIQ. . . . . Liquid internal energy U
Given: temperature T.
(Must correspond to VH, VHFG values.)

VQUALH . . . . Calculates: Quality X
Given: pressure P and enthalpy H.
(Must correspond to VH and VHFG values.)

VHFG . . . . . Calculated: Heat of vaporization HFG
Given: pressure P and temperature T and vapor sp. volume V
(Must correspond to VH and VHLIQ values.)

VALPHA . . . . . Calculates: Void fraction A
Given: temperature T and quality X.
(Must correspond to VSV and VDL.)

VSOSF . . . . . Calculates: liquid speed of sound SOS
Given: pressure P and temperature T
(optional if VDLC provided)
VSOSFV. . . . . Calculates: liquid, vapor, or two-phase speed of sound SOS
Given: pressure P, temperature T, quality X, and method METH
(optional if either VDLC or VSOSF *and* either VSOS or VS provided)
VVISCFC . . . . Calculates: liquid dynamic viscosity VISC
Given: pressure P and temperature T
Do not provide VVISCF if VVISCFC is provided.
May be used with incompressible liquids as well.
VCONDFC. . . Calculates: liquid conductivity COND
Given: pressure P and temperature T
Do not provide VCONDF if VCONDFC is provided.
May be used with incompressible liquids as well.

Optional Subblocks, More Appropriate Methods (if COMPLIQ is used):

VULIQC. . . . . Calculates subcooled (compressed) liquid internal energy U
Given: pressure P and temperature T

3.21.7.8 Example 6000 Series FPROP DATA Block

(The direct link and tabular style property files described in Section 3.21.7.9 and Section 3.21.7.10 and available from www.crtech.com represent additional examples of 6000 series blocks.)

In the following block, a description of ammonia is input using current internal FLUINT property routines as externally provided routines, such that no SUBROUTINE DATA or additional processor libraries are needed. While there is obviously no reason to input such a description when one already exists (as fluid 717), this simple example should illustrate this option using a set of property routines already available to all users.

In these routines, a pointer to a data array must be initialized in the VINIT subblock. Also, the value of a logical flag SI (held in common RCONV) must be set to false before each call, and then reset to its previous value. These are examples of the types of operations that might be performed in typical VINIT blocks.

Note that the bare minimum of routines is input. Entropy and entropy-based calculations will not be available for this fluid, and standard calculation methods will be used to derive all other necessary properties. If the user had better, faster, or otherwise more appropriate property routines, these could have been added to the following list.

```
HEADER FPROP DATA,6717,ENG,0.0
C
C AMMONIA IN ENGLISH UNITS USING STANDARD LIBRARY ROUTINES
C !!!!!! EXAMPLE ONLY !!!!!! NORMALLY FETCH TABLES FROM WWW.CRTECH.COM!
       TGMAX          = 3.0*730.0799,               PGMAX          = 1636.37
       TCRIT          = 730.0799,                   PCRIT          = 1636.37
       TMIN           = 351.8
VINIT
       COMMON/FLUDAT/IFI(1)
       COMMON/FLUDT1/FD(1)
       COMMON /RCONV/ PCON,TCON,HCON,VCON,CCON,SI,PVC,PTOWD
       LOGICAL SI,SIL
       N              = 372
       SIL            = SI
       SI             = .FALSE.
VPS
       P              = ZPS(T,FD(N))
       SI             = SIL
VDPDTT
       DPDT           = ZDPDTT(T,FD(N))
       SI             = SIL
VH
       H              = ZH(P,T,V,FD(N))
       SI             = SIL
VSV
       V              = ZSV(P,T,FD(N))
       SI             = SIL
VST
       ST             = ZST(T,FD(N))
       SI             = SIL
VCONDF
       COND           = ZCONDF(T,FD(N))
       SI             = SIL
VCONDV
       COND           = ZCONDV(T,FD(N))
       SI             = SIL
VVISCF
       VISC           = ZVISCF(T,FD(N))
       SI             = SIL
VVISCV
       VISC           = ZVISCV(T,FD(N))
       SI             = SIL
VDL
       D              = ZDL(T,FD(N))
       SI             = SIL
```

The listing shown below is the Fortran subroutine produced by FLUINT based on the above inputs. Although the user normally does not need to look at the resulting subroutine, it is presented here to help illustrate concepts. In the master model in which this input file was run, the unit system was SI, and ABZSRO=0.0. Note the unit conversions performed before and after each subblock. The encoding and decoding of the DPARG, SPARG, and INTARG arrays are handled by the program.

```
      SUBROUTINE UF6717(DPARG,SPARG,INTARG)
      DOUBLE PRECISION DPARG(1),P
      DIMENSION SPARG(1),INTARG(1)
C VINIT
       DOUBLE PRECISION ZPS
       COMMON/FLUDAT/IFI(1)
       COMMON/FLUDT1/FD(1)
       COMMON /RCONV/ PCON,TCON,HCON,VCON,CCON,SI,PVC,PTOWD
       LOGICAL SI,SIL
       N              = 372
       SIL            = SI
       SI             = .FALSE.
C VPS
      IF    (INTARG(1) .EQ.    1) THEN
       T = SPARG(1) *   1.8000000
       P              = ZPS(T,FD(N))
       SI             = SIL
       DPARG(1) = P *  6.89475977d+03
C VDPDTT
      ELSEIF(INTARG(1) .EQ.    2) THEN
       T = SPARG(1) *   1.8000000
       DPDT           = ZDPDTT(T,FD(N))
       SI             = SIL
       SPARG(   2) = DPDT *   12410.567
C VH
      ELSEIF(INTARG(1) .EQ.    3) THEN
       T = SPARG(1) *   1.8000000
       P = DPARG(1) *  1.45037688d-04
       V = SPARG(2) *   16.018463
       H              = ZH(P,T,V,FD(N))
       SI             = SIL
       SPARG(   3) = H *   2326.1113
C VSV
      ELSEIF(INTARG(1) .EQ.    4) THEN
       T = SPARG(1) *   1.8000000
       P = DPARG(1) *  1.45037688d-04
       V              = ZSV(P,T,FD(N))
       SI             = SIL
       SPARG(   2) = V *  6.24279603e-02
C VST
      ELSEIF(INTARG(1) .EQ.    5) THEN
       T = SPARG(1) *   1.8000000
       ST             = ZST(T,FD(N))
       SI             = SIL
       SPARG(   2) = ST *   14.593176
C VCONDF
      ELSEIF(INTARG(1) .EQ.    6) THEN
       T = SPARG(1) *   1.8000000
       P = DPARG(1) *  1.45037688d-04
       COND           = ZCONDF(T,FD(N))
       SI             = SIL
       SPARG(   2) = COND *   1.7309999
C VCONDV
      ELSEIF(INTARG(1) .EQ.    7) THEN
       T = SPARG(1) *   1.8000000
       P = DPARG(1) *  1.45037688d-04
       COND           = ZCONDV(T,FD(N))
       SI             = SIL
       SPARG(   2) = COND *   1.7309999
C VVISCF
      ELSEIF(INTARG(1) .EQ.    8) THEN
       T = SPARG(1) *   1.8000000
```

```
        P = DPARG(1) *  1.45037688d-04
        VISC           = ZVISCF(T,FD(N))
        SI             = SIL
        SPARG(   2) = VISC *  4.13333328e-04
C VVISCV
       ELSEIF(INTARG(1) .EQ.    9) THEN
        T = SPARG(1) *   1.8000000
        P = DPARG(1) *  1.45037688d-04
        VISC           = ZVISCV(T,FD(N))
        SI             = SIL
        SPARG(   2) = VISC *  4.13333328e-04
C VDL
       ELSEIF(INTARG(1) .EQ.   10) THEN
        T = SPARG(1) *   1.8000000
        D              = ZDL(T,FD(N))
        SI             = SIL
        SPARG(   2) = D *   16.018463
C
       ELSE
         CALL VERROR('UF6717','NO SUCH PROPERTY',6717)
       ENDIF
       RETURN
       END
```

3.21.7.9 Direct Calls to REFPROP: “R files”

In theory, FPROP blocks are “user supplied” and are the responsibility of the user to develop, verify, and maintain. In practice, the amount of work involved can be onerous, so as a aid to FLUINT users, CRTech has incorporated NIST’s REFPROP property program and made available mechanisms for both direct and indirect links to that data. Nonetheless, these files and interfaces are largely independent of SINDA/FLUINT updates, and so it is the user’s responsibility to acquire and maintain the latest updates.

This section and the next section describe these files. This section describes a mechanism for making direct calls to REFPROP from within 6000 series blocks. While accurate and full-ranged, they can be slow to execute. Therefore, alternative table-style methods are available as an alternative since they enable one to two orders of magnitude increase in speed. This acceleration is usually well worth the inevitable but usually slight reduction in accuracy that results from discretized and interpolated data points. Those alternative table-based files are described in the next subsection.

REFPROP contains properties of all important cryogens (helium, oxygen, para- and normal hydrogen, nitrogen, etc.), various hydrocarbons (including natural gas constituents), water, ammonia and (as the name REFPROP implies) almost all refrigerants. Refer to www.nist.gov/srd/nist23.htm and Table A-3 for a complete list. Currently, only pure substances or “pseudo-pure fluids” (PPF) such as air are supported in SINDA/FLUINT as either working fluids or mixture constituents; SINDA/FLUINT mixtures cannot currently exploit the availability of REFPROP mixture calculations.

Direct calls to REFPROP are often called “R files” because the naming scheme starts with an “r:” *r6001_ammonia.inc*, for example. The “default” type is the incompressible liquid version. The NEVERLIQ versions have an “NL” in the name (e.g,. *r6001NL_ammonia.inc*), and the COMPLIQ versions have a “CL” (e.g., *r6001CL_ammonia.inc*).

Because they are reasonably small, all available R files are included with SINDA/FLUINT installation,* along with auxiliary fluid-independent files that are INSERTed in a nested fashion from within the R files. However, this inclusion with SINDA/FLUINT installation does not eliminate the need to periodically check for updates to these files.

For example, to include properties for compressible-liquid helium, the user can peruse the available list of R files in the subdirectory *\insert_files*, perhaps browsing from within Sinaps or FloCAD, and then INSERT the file *r6018CL_helium.inc* and use the fluid ID "6018" in their fluid submodel(s).

Some fluids are lacking data such as surface tensions or transport properties. They will purposely not compile if INSERTed into the input stream as a signal that more data is needed before they can be used in a SINDA/FLUINT model.

For the direct link to REFPROP to work, the program must be able to locate the FLD files in the installation set. The advanced user, exercising considerable caution, can alter the FLD files or insert their own as needed.

One reason to prefer slower R files over F files (as described next) is that they are not sensitive to resolution. Solving in a transient for a tank state which is near the critical point, a SINDA/FLUINT run may abort if it encounters too-coarse properties (see bottom of next subsection). Sometimes, no amount of increased table resolution avoids this difficulty, and R files become necessary. Nonetheless, even REFPROP can experience some difficulty extremely close to the critical point for some fluids (e.g., helium).

The astute user will note the use of memory with the R file logic: preserved list of previously calculated data points. The primary purpose of these lists is to speed up REFPROP for iterative calls, but it also assists in robustness. If a prior point is found, it is returned without calling REFPROP at all. Otherwise, if a prior point is close enough, it is used as an initial condition.

More details on the logic within R files may be gleaned from REFPROP documentation, available with purchase of REFPROP from NIST, or from various online FAQ pages.

3.21.7.10 REFPROP-generated Tables: "F files"

An alternative to calling REFPROP every time a property is needed is to call it ahead of time over a wide range of temperature and pressures, save the data points, and interpolate between them when data is required. Such is the logic behind "F files" (e.g., *f6001_ammonia.inc*).

Tabular descriptions, correctly done, also exhibit both fast execution and a wide domain—the two are normally mutually exclusive. However, they are based upon an approximation (beyond the approximations already contained within REFPROP and SINDA/FLUINT): that linear interpolations between a finite set of points is adequately accurate. The user should note that uncertainties in other parts of the simulation (e.g., appropriateness of correlations, etc.) often overwhelm this approximation, and that the 10X to 100X speed savings that result are usually well worth the slight inaccuracies introduced.

* Versions 5.3 and later. R files will not function with older versions of SINDA/FLUINT.

F files are too large to include within SINDA/FLUINT itself, so they may be fetched as needed from www.crtech.com or requested from CRTech. As with R files, the user is cautioned to stay up-to-date on changes to these files, especially since they are not automatically loaded along with new versions of the software. Like R files, F files have their own auxiliary files (e.g., *bottomCL.inc*), but these smaller fluid-independent files *are* included with SINDA/FLUINT installation.

FPROP tables contain key properties (specific volume, thermal conductivity, etc.) as square tables of temperatures (TEMP array) and pressures (PRES array). Table sizes of 81x81 and 162x162 ("big") are available. The diagonal values (i.e., TEMP(j) and PRES(k) where j=k) of the square represent saturation (for non-NEVERLIQ versions) for low temperatures and pressures, with the critical point located near the middle of the table.* Above the critical temperature or pressure, *stored* properties for liquid and vapor become the same (but *returned* properties may differ depending on whether the liquid is compressible or incompressible, for example).

In the metastable zone (usually past the spinodals), and sometimes for trouble points near the critical point, data has been smoothed (interpolated from nearby valid points). Such altered data is marked using a string of 4's (e.g, 4444) in the least significant bits.

The advanced user, exercising considerable caution, can alter the property files or insert their own as needed.†

"ILLEGAL PROPERTIES ENCOUNTERED"—For some fluids (e.g., helium), when solving in a transient for a tank state which is near (slightly above) the critical point, a SINDA/FLUINT run may abort if it encounters too-coarse properties. Increased table resolution (the "big" files) can help avoid the problem interpolation cells. However, no amount of increased table resolution completely avoids this difficulty, and R files then become necessary (3.21.7.9).

Before proceeding such a slow-solving method as the R files represent, the user should first verify that the model has not "accidently" moved into this problematic region via spurious pressure excursions. In other words, it is possible that a change in modeling choice, improved initial condition, and so forth could help avoid the need to analyze temperatures and pressures just above the critical point.

* Usually cell 54 of 81 for the small tables, and cell 81 of 162 for the "big" tables, though these are subject to change.

† Before attempting to create a new fluid or alter an existing table, contact CRTech for recommendations. For example, variations of these tables for special purposes. Examples include extra-large double precision versions, versions with ice (slush) properties below the triple point, low pressure extrapolations, etc.

3.22 Fluid Interactions: MIXTURE DATA

8000 series user-defined fluids (perfect and noncondensible gases, as described inSection 3.21.4) may be used with other working fluids to create mixtures. By default, the gas is assumed to have zero solubility. However, if required the program can be used to calculate equilibrium and finite-rate dissolution and evolution of noncondensible gases into and out of other liquids present in the system. The program is capable of modeling systems involving one or more solutes and one or more solvents, as described in Section 3.23.

When modeling dissolution of gases, *additional fluid properties must be defined with the FPROP blocks (Section 3.21) of both the solute and the solvent.*

Furthermore, the user must define the equilibrium solubility between any pair of solvents and solutes using MIXTURE DATA, as described in this section.*

3.22.1 Solubility Data

Various means can be used to represent the solubility of a gas within a liquid. Henry's Law, which states that the solubility (in terms of dimensionless mole fraction) is proportional to the partial pressure of the gas, $x = p/H$, is perhaps the most useful relationship for engineering purposes.

Henry's Law is a generalization of Raoult's Law, which predicts that the constant H in the above equation is equal to $H = p_s$, where p_s is the saturation pressure of the gas. For the normal case where the soluble gases are in a supercritical state, p_s can be estimated by ignoring the critical point and extrapolating to fictitious supercritical "saturation" temperatures.

Raoult's Law predicts solubility that is independent of the solvent and is dependent only on the temperatures and pressures of the system. Henry's Law is more empirical, in that the equilibrium constant k is usually a function of *both* the solvent and the temperature. However, Raoult's Law is important since it provides a default model for those gaseous constituents in which a saturation relationship has been provided (which will often be the only information readily available to the user).

Another useful relationship is the Ostwald coefficient, or the ratio of the density of the dissolved gas divided by the density of the gas in the vapor phase: $O = \rho_{gd} / \rho_{gv}$. These two density values represent the mass of the solute in each phase, divided by the total volumes of each phase (not the partial or molar volumes).† The Ostwald coefficient is important because it is reasonably constant for isobaric (but not necessarily isothermal) mixtures. This coefficient is also relatively insensitive to the particular species involved, since it normally remains in the range of 0.001 to 0.1 for most binary systems. In other words, analysts lacking better data can use such an approach to generate preliminary estimates.

---

* As an alternative to MIXTURE DATA, the user can assume control of each lump's Henry's constant (HENx, where 'x' is the species letter identifier) directly using each lump's IDGC control constant.

† Caution: Multiple definitions of Ostwald coefficient exist. Be sure to compare the definition of the data for the source you are using with the one used by SINDA/FLUINT.

Since solubility data itself is difficult to find and can be presented in various forms, the user interface presented below was designed to be flexible enough to handle a variety of solubility data for each binary pair of solvent and solute. Note however, that the generic structure for specifying "Henry's constants" is also reused for other solubility units that are not true Henry's constants.

Multiple solutes and/or solvents: Solubility of multiple solutes in the same solvent will be treated independently with respect to the solvents, but the interdependence of multiple solutes on the vapor space will be accounted for. To illustrate the interdependence of solutes in the vapor space, consider the following problem. Consider a lump containing a nonvolatile liquid with no vapor space at high pressure. If there are two solutes that were to be independent, the solutes would stay completely dissolved if the pressure is above x*H of both solutes, where H is the Henry's constant and x is the mole fraction of the solute. But when the solutes are treated together, the pressure has to be above $x_1*H_1+x_2*H_2$ for the solutes to stay in solution.

In cases where multiple solvents are present, the amount in solution will be computed with an effective solubility constant computed as $\text{Log}(H_{eff}) = \Sigma(x_i*\text{Log}(H_i))$ for each solute (see "Properties of Gases and Liquids," Reid et al.). ***Note that if MIXTURE DATA is missing for any liquid in the system, then its solubility is zero (H is infinite). If the solubility in any one liquid is zero, then resulting liquid mixture will have essentially zero solubility even if the solubility of other solvents has been input.***

Solutes are assumed to have zero solubility in liquids that are assumed immiscible: *immiscibility implies insolubility.* However, in this case the immiscible liquid, while insoluble itself, is assumed to not block the dissolution of gases into the remaining portions of the miscible mixture that might be soluble (see also Section 3.21.5.3).

Internally, all forms of solubility data are converted into Henry's constants. The user may inspect the Henry's constants at any temperature and pressure by calling the HENRY routine (Section 7), which reflects the MIXTURE DATA inputs ignoring the effects of multiple solvents. The MAPHEN routine (Section 7) tabulates the values of Henry's constants as input into MIXTURE DATA, again ignoring the effects of multiple solvents. Otherwise, the effective constant for any species (including the effects of other solvents) may be accessed during program execution using the HEN variable for tanks and junctions as described in Section 3.23.3.

3.22.2 MIXTURE DATA Input Formats

The HEADER MIXTURE DATA block can be used to specify the various combinations of solubility data, including the units of the data. The general format is:

```
HEADER MIXTURE DATA, Type [,fluid identifier list]
       [,keyword=value] [,keyword=value]
```

This block is intended to be expanded in future versions to handle additional types of working fluid interactions. Currently, however, the "Type" is always "BIN" for binary mixtures, and the fluid identifier list is restricted to two entries: a solute and a solvent (in that order).

To facilitate the ease of use of solubility data, a number of different units are supported in the input. These units were selected based on those found in wide cross section of reference data.

In order to provide the user with the capability to model solubility with a minimum of available data, three separate correlations (Henry's Law, Ostwald Coefficients, Raoult's Law) have been provided. Only one such correlation may be used for each binary pair of fluids.

HEADER MIXTURE DATA subblock

The reference-style and array-style inputs of MIXTURE DATA are patterned after the analogous FPROP DATA blocks, with which the reader is assumed to be already familiar (see Section 3.21).

The specific format is of the subblock formed by the HEADER line is:

```
HEADER MIXTURE DATA, BIN, nf1, nf2 [,uid [,abszro]]
       [,TREF=R] [,PREF=R] [,ZJ=R] [,UNITS=I]
       [,HL=R] [,HTC=R]
       [,OC=R] [,OTC=R] [,OPC=R]
       [,RL]
```

where:

`nf1` . . . . . The gas solute fluid identifier (must be an 8000 series fluid).

`nf2` . . . . . The liquid solvent fluid identifier (must be a library fluid, or a 6000, 7000, or 9000 series fluid)

`uid` . . . . . local unit system: SI or ENG. Defines the local unit system of this and only this header block, such that if this block is reused in another model unit conversions can be made. (Default: the UID value of the master model. *The user is strongly recommended to supply a value to assure that this block can be reused safely in other models.)*

`abszro` . . . . . absolute zero in the local unit system. Defines the local unit system of this and only this HEADER block, such that if this block is reused in another model unit conversions can be made. (Default: the ABSZRO value of the master model. *The user is strongly recommended to supply a value to assure that this block can be reused safely in other models.*)

`TREF` . . . . . The temperature at which the reference style inputs are based.

`PREF` . . . . . The partial pressure at which the true Henry's constant data is input (when UNITS=1), in the local unit system (psia or Pa). Utilized for unit conversions and for Ostwald coefficients.

`ZJ` . . . . . Rate constant/Bubble nucleation rate. This is the term Z/J in the formulation used by Forest and Ward to compute the bubble homogeneous nucleation pressure (Section 3.23). Orders of magnitude errors in this term result in only a small change in the predicted nucleation pressure. (Default: 1.0E26)

`UNITS` . . . . . Used to indicate the *input* (not internal or output) units of the values input for "Henry's constant."* The following are the accepted units:

1 . . . . . .atm/(mole-solute/mole-solvent), *not psia or Pa!*
2 . . . . . .cc/gm (cc at standard temperature, pressure)
3 . . . . . .cc/gram-mole (cc at standard temperature, pressure)
4 . . . . . .mass fraction
5 . . . . . .mole fraction

`HL` . . . . . The value of "Henry's constant" at TREF. (No default.)

`HTC` . . . . . Henry's constant temperature coefficient. (Default: 0.0)

`OC` . . . . . The value of the Ostwald coefficient. (No Default.)

`OTC` . . . . . Ostwald coefficient temperature coefficient. (Default: 0.0.)

`OPC` . . . . . Ostwald coefficient pressure coefficient. (Default: 0.0.)

`RL` . . . . . Flag indicating that Raoult's Law ($H=p_s$) is to be utilized for solubility data. Use of the Raoult's Law option requires that the Psat-Tsat data be supplied in the FPROP block for the solute.

* Actually, only UNITS=1 corresponds to a true Henry's constant. The others are treated as "Henry's constants" in this block, but are converted internally into true Henry's constants in units of pressure (psia or Pa), depending on the value of UID in the model) divided by (unitless) mass fraction.

Examples:

```
HEADER MIXTURE DATA, BIN, 8702, 717, RL
HEADER MIXTURE DATA, BIN, 8728, 9718, SI, 0.0, HL = 77039., UNITS = 1, PREF = 101325.0
```

When the identified fluid is not present in the model, the mixture data is ignored, and a user caution is printed.

Guidance: Zero solubility is a legal input. For Ostwald coefficients or other mass or molar fractions ("Henry's constant" when UNITS = 2,3,4,5), input zero. For zero solubility using the *true* Henry's constant (UNITS=1), specify 1.0E30. Removing the MIXTURE DATA block eliminates solubility of that solute/solvent pair globally. Infinite solubility is not allowed (Henry's constant = 0.0 with UNITS=1, or 1.0E30 for other coefficients), but very large solubilities are acceptable and may serve the same modeling need.

The MAPHEN routine (Section 7) tabulates the values of Henry's constants as input into MIXTURE DATA (or converted into Henry's constants in units of Pa or psia if input in some other units or format) as a function of temperature and pressure, ignoring the effects of multiple solvents.

Optional Array subblocks

A bivariate array can used to input the either the values of "Henry's constant" or the Ostwald coefficient when data is available as a function of *partial* pressure and temperature. As with FPROP data, the appearance of an AT subblock will override any reference style input.

The format of optional bivariate arrays is:

AT, type, n, p_1,p_2,...,p_n,
t_1, x_{11},x_{12},...,x_{1n},
t_2, x_{21},x_{22},...,x_{2n},
.
.
.
t_m, x_{m1},x_{m2},...,x_{mn}

where:

type	HL for Henry's constants OC for Ostwald coefficients
n	the number of pressure points that will be input
p_n	partial pressure input
t_n	temperature input
x_{mn}	solubility data (i.e., Henry's constant or the Ostwald coefficient) at (t_n, p_n)

This format can also be utilized to input data as a function only of temperature by inputting the data at a single (arbitrary) partial pressure. For example:

AT, type, 1, p,
t_1, x_1,
t_2, x_2,
.
.
.
t_m, x_m

where “p” can be any value since the above data will be applied at all pressures.

Caution—Solubility data is often scarce, and it can vary sharply with pressure. In order to avoid limiting a fluid flow solution to the range of available data, if a pressure or temperature value exceeds the range of input data for the arrays, then ***the endpoint value is used: no extrapolation is performed.*** In this aspect, AT arrays in MIXTURE DATA are unlike those in FPROP DATA. The user should include as wide a range of data as is available. Because no extrapolation is performed, if each input value is valid, then the valid range for each analysis will not be limited by available solubility data. However, the predictions produced outside the range of MIXTURE DATA inputs should be used with caution.

Note: As with any SINDA/FLUINT array input, the orderly arrangement presented above need not be obeyed literally by the user. While the inputs are order-dependent, as many values as needed can occur on any line. Thus, the following is equivalent to the above example:

AT, type, 1, p, t_1,x_1, t_2,x_2, ... t_m,x_m

An example of such a temperature-only MIXTURE DATA block is presented below:

```
HEADER MIXTURE DATA,BIN, 8000,9718,ENG,0.0
C
C  DATA FOR N2 IN WATER
C  REFERENCE COHEN, PG 109
C   UNITS ARE IN ATM/MOLE FRACTION
C   TEMPERATURE DEG R
C
        UNITS=1
        PREF=10000000.99
AT,HL, 1, 1.,
        500.0,102600.0,
        560.0,102600.0,
        610., 120900.0,
        660., 123100.0,
        710., 114500.0,
        760.,  99800.0,
        810.,  83000.0,
        860.,  66200.0,
        910.,  50700.0,
        960.,  37100.0,
        1010., 25600.0,
        1060.,  15800.0
```

3.23 Gas Dissolution/Evolution Modeling Options

By default, gases are assumed to have zero solubility into any liquids with which they are mixed.

The user can choose to model equilibrium and nonequilibrium dissolution and evolution of any noncondensible gas (8000 series user-defined fluid) into any fluid with a liquid phase. To perform such analyses

1) the user must provide additional fluid parameters as required within the FPROP blocks of user-defined fluids (Section 3.21). Library fluids already contain the required data internally.

2) for each solute and solvent pair, the user must provide a MIXTURE DATA block (Section 3.22) to define the equilibrium solubility, and perhaps rate data (the ZJ factor) for homogeneous nucleation modeling.

Only 8000 series gases may be solutes. However, any number of solvents and solutes may exist within a mixture. The solubility of multiple *solutes* within the same solvent will be treated independently with respect to the liquid phase, but the effect of multiple solutes will be taken into account when predicting if a vapor/gas phase can exist. In cases where multiple *solvents* are present, the amount in solution will be computed with an effective solubility constant computed as $\text{Log}(H_{eff}) = \Sigma(x_i * \text{Log}(H_i))$ for each solute. Refer to Section B.7 for more information on the methods and correlations involved in modeling of soluble substances.

The modeling of dissolved gases involves many variables and suboptions, and includes the ability to scale unknowns and assume control of various calculations. These variables may be used in the path, lump, and macro options described earlier. However, due to their large number and the specialized nature of their usage, these variables and modeling options are described separately in this section.

3.23.1 Overview of Modeling Methods

This section provides brief background information as well as an overview of the philosophy governing the user interface for dissolved gas modeling.

3.23.1.1 Basic Phenomena and Equations

When a soluble gas is introduced to a mixture containing a liquid solvent, the gas will very slowly begin to dissolve into the liquid until it either reaches its equilibrium concentration in each phase, or until the gas phase disappears (as long as all other gaseous species present also dissolve completely and any vapor phase condenses completely).* The solute dissolves into the solvent at a rate predicted using Fick's Law: the rate is proportional to the difference between the equilibrium concentration and the current concentration.

* If complete dissolution always occurs for the solute of interest, then significant computational savings can be made by describing the solute as a 9000 series fluid rather than as a soluble 8000 series fluid.

If the concentration of solute exceeds the equilibrium concentration (perhaps do to a decrease in pressure), *and if a void exists*, then the gas will slowly diffuse to the free surface where it will come out of solution. If on the other hand no void exists, then the gas will continue to stay dissolved well above the equilibrium concentration in a nonequilibrium state. Eventually, if the pressure drops below a critical pressure, the gas will form bubbles within the liquid spontaneously and explosively, nucleating homogeneously.

If any small voids are present in crevices in the wall, the gas will diffuse to those voids and evolve heterogeneously (such as when a supersaturated soda can is opened). Homogeneous nucleation (bulk boiling) also occurs in pure substances, although usually only under contrived conditions. It is much more common, however, with respect to evolution of solutes, and is often a violent event. In systems with high degrees of solubility (e.g., nitrogen in Halon™ 1301), when the pressure drops below the homogeneous nucleation pressure, the liquid almost instantly becomes a foam and the system pressure can rise rapidly above the original pressure.

Finally, when a volatile substance evaporates and it contains dissolved gases, then a certain amount of solute will be evolved via advection.

The equation defining the mass rate of dissolution is then:

$$\dot{m}_d = G_d \cdot A_{surf} \cdot (X_{eq} - X)$$

where G_d is the Fick's Law (mass transfer) constant, A_{surf} is the surface area of the interface, X_{eq} is the equilibrium mass fraction of solute in the solvent, and X is the current mass fraction.

For evolution, additional terms are added to cover the homogeneous nucleation rate (which when present is completely dominant) and the advection rate. Since flow rate is defined to be positive in FLUINT for dissolution, and since the Fick's Law constant governing evolution (G_e) may not be the same as that governing dissolution (G_d), the above equation becomes:

$$-\dot{m}_e = G_e \cdot A_{surf} \cdot (X_{eq} - X) + \dot{m}_h + \dot{m}_a$$

More details may be found in the Appendix in Section B.7.

3.23.1.2 Lumps and Associated Paths

Like evaporation and condensation, dissolution and evolution happen within lumps, and not paths. Also like evaporator and condensation, the rates at which these phenomena happen must be predicted using flow rate and shape information associated with tubes and STUBE connectors adjacent to the lumps if default correlations are to be used. Heat transfer ties (Section 3.6) such as HTN, HTNC, and HTNS are able to invoke correlations for condensation, evaporator, and single-phase heat transfer because they associate a lump with a path (as well as a node).

Dissolution and evolution modeling options by default use an analogy between heat and mass transfer to predict the Fick's Law constants described above. Thus, the attributes of paths (interfacial surface area, phase velocities, etc.) can be apportioned to up or downstream tanks and junctions. Unlike a tie, however, no node is involved since the mass transfer occurs between two phases of a single lump.

Attributes of a path include its shape, flow rate, interfacial surface area, and other data used to calculate the evolution/dissolution rate constant. For FLUINT to predict surface area of a path and the path's contribution to the rate constant, it must be provided with a flow regime using IPDC=6 (the default), -1, -2, -3, or -4 (Section 3.16).* Otherwise, the user may specify directly the surface area and rate constant contribution as described in later subsection.

Thus, the G*A term of a lump can be calculated by default using contributions of its adjacent paths. For lumps representing a section of a duct (e.g., that are part of a duct macro), these calculations may fully describe the rate of evolution or dissolution within a lump. However, the user can also use a user-defined rate constant and surface area to describe additional mass transfer within the lump, perhaps without requiring the association of any paths at all. This situation will occur when modeling the evolution/dissolution within a vessel, and requires the user to supply information regarding the surface area and rate constant calculation methods.

The total dissolution/evolution rate at a lump is therefore the sum of its (optional) contribution as well as the (optional) contributions of associated paths:

$$FRDx = FRHx + FRNCVx + (Xx - XFx) \cdot$$

$$(ASL \cdot GLx + \sum_k^{down} GDx_k \cdot FDAS_k \cdot ASPD_k + \sum_k^{up} GUx_k \cdot FUAS_k \cdot ASPU_k)$$

where

FRDx . . . . . mass flow rate of dissolution for species x within the lump
FRHx . . . . . mass flow rate of species x evolving homogeneously within the lump (negative if present)
FRNCVx. . . . . mass flow rate of species x evolving due to advection within the lump (negative if present)
Xx. . . . . the equilibrium mass concentration of species x within the lump
XFx. . . . . the current mass concentration of species x within the liquid in the lump
ASL. . . . . the interface surface area of the lump (in addition to any surface area attributed to it via associated paths)
GLx. . . . . the rate constant for dissolution/evolution within the lump
GDx_k. . . . . the rate constant for species x in the downstream part of path k (whose downstream portion has been attributed to the lump)
$FDAS_k$ . . . . . the fraction of the downstream end of path k attributed to the lump
$ASPD_k$ . . . . . the surface area of path k on its downstream end
GUx_k. . . . . the rate constant for species x in the upstream part of path k (whose upstream portion has been attributed to the lump)
$FUAS_k$ . . . . . the fraction of the upstream end of path k attributed to the lump
$ASPU_k$ . . . . . the surface area of path k on its upstream end

Note that the "upstream" and "downstream" designations do not change if flow rate reverses.

* For paths that are not members of duct macros, nonzero FUAS and FDAS must also be specified.

This superposition of path and lump contributions to the total mass transfer is depicted in Figure 3-42.

If LINE or HX macros are being used with the default of IPDC=6 (or negative IPDC) to model a duct, then the default system is almost entirely active and few user inputs are required. However, the user can alternately assume control of various parameters, perform scaling of various underlying correlations, or model specialized nonduct components. It is to support these uses that most of the complications in this section arise, including the large number of variables available.

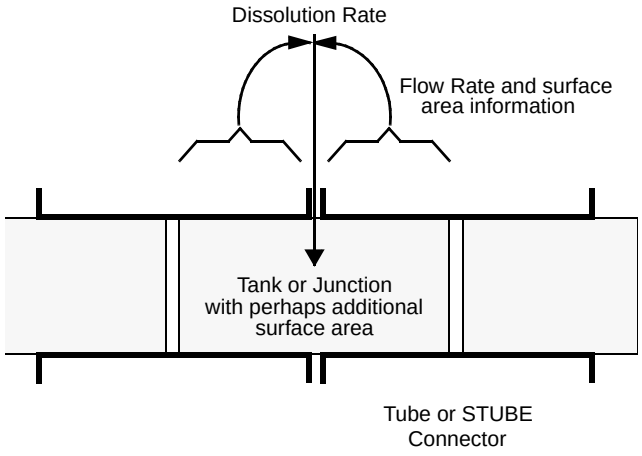


Figure 3-42 Lump plus Associated Paths for Modeling Dissolution/Evolution

3.23.1.3 Scaling and Correlation Variables

As with any two-phase system, there is a large range of uncertainty in the predictions of dissolution/evolution phenomena. Therefore, it is important for the user to be able to easily test the sensitivities of various inputs, and to be able to adjust the model to correlate to any available test data.

For this reason, most of the variables used for modeling solvents and solutes use a default system. If the value of the input parameter is nonnegative, it is used directly and internal estimation methods and correlations are ignored. If it is negative, its absolute value is used as a scaling factor. Thus, most of the input parameters have default values of -1.0. For example, GLx=1.0 means to use a value of 1.0 for GL of species x. GLx=-1.0 means to use the default system of calculations and correlations to estimate GLx. GLx=-2.0 means use twice the default calculation.

In addition to tabulation routines such as GASTAB, GASTB2, GASATB, and GASGTB (Section 7), the user can inspect results using parallel output parameters prefaced with a “C,” referring to them in input expressions or within logic blocks. For example, to refer to the current GLx factor for lump 10 and species R, the user would refer to “CGL10R” or “CGLR(10).” If GLx is zero or positive, of course, CGLx will equal GLx. Table 3-16 summarizes the relevant variables and their corresponding references as output variables (in the event the input values are negative and therefore used as scaling factors).

Table 3-16 Summary of Scalable Dissolution/Evolution Variables

Element	Variable Name	Output Reference	Description
tank or junction	ASL	CASL	surface area within the lump
	GLx	CGLx	Fick's Law constant for dissolution, species x
	UVI	CUVI	vapor to interface heat transfer coefficient (used for calculating GL)
	ULI	CULI	liquid to interface heat transfer coefficient (used for calculating GL)
tube or STUBE	ASPU	CASPU	additional surface area for upstream lump
	ASPD	CASPD	additional surface area for downstream lump
	GUx	CGUx	Fick's Law constant for dissolution, for upstream lump, species x
	GDx	CGDx	Fick's Law constant for dissolution, for downstream lump, species x
	UVPU	CUVPU	vapor to interface heat transfer coefficient for upstream lump (used for calculating GU)
	UVPD	CUVPD	vapor to interface heat transfer coefficient for downstream lump (used for calculating GD)
	ULPU	CULPU	liquid to interface heat transfer coefficient for upstream lump (used for calculating GU)
	ULPD	CULPD	liquid to interface heat transfer coefficient for downstream lump (used for calculating GD)

3.23.1.4 Considerations when using Junctions

Junctions are time-independent control "volumes." They react instantly to changes.

However, this does not prevent junctions from exhibiting finite rates of dissolution and evolution. For example, consider a duct that is modeled using a series of junctions. Vapor and gas are flowing into the duct, which is chilled enough to cause condensation of the vapor. Neglecting dissolution by perhaps leaving the solubility at the default of zero, the program would predict that all of the gas remained in the void phase along with a little vapor. Using equilibrium solubilities with infinite rate mass transfer, the program would predict perhaps that all of the gas had dissolved, and perhaps that the exit state was 100% liquid (XL=0.0). Reality is of course in between these two extremes, with the gas dissolving a little bit but the solute fraction at the exit of the duct would be far from equilibrium given the short transit time within the duct. Junctions can be used to predict such effects while themselves being completely time independent.

The volume (VOL) of a junction is normally ignored by the program, and is present mostly as a convenience for switching between tanks and junctions, and for calculating fluid masses etc. However, the VOL of a junction is used in the prediction of homogeneous nucleation rates, which are often correlated on a volumetric basis. Thus, ***if the volume of a junction is defaulted to zero, no homogeneous nucleation will be modeled in that junction.***

3.23.2 Additional Path Variables

The following are additional variables available for use with any tube or STUBE, including those generated using GENPA, LINE, or HX. Use of IPDC=6 (or negative) is almost mandatory: most values will otherwise default to zero.

In the following formats, the lower case "x" should be replaced by the letter identifier of the soluble gas ('A', 'B', etc.). The additional variables are:

```
[,ASPU=R][,ASPD=R][,GUx=R][,GDx=R]
[,IUAS=I][,IDAS=I][,FUAS=R][FDAS=R]
[,UVPU=R][,ULPU=R][,UVPD=R][,ULPD=R]
```

where:

ASPU. . . . . Path upstream interfacial surface area. (Default: -1.0, internally computed interfacial area applies; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.)

ASPD. . . . . Path downstream interfacial surface area. (Default: -1.0, internally computed interfacial area applies; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.)

GUx . . . . . path upstream lump mass transfer coefficient for soluble species "x." (Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.) CGUx contains the calculated value for references in logic or expressions.

GDx . . . . . path downstream lump mass transfer coefficient for soluble species "x." (Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.) CGDx contains the calculated value for references in logic or expressions.

IUAS. . . . . Upstream lump ID for interfacial surface area. (Default: the upstream lump)

IDAS. . . . . Downstream lump ID for interfacial surface area. (Default: the downstream lump)

FUAS. . . . . Fraction of interfacial surface area assigned to upstream lump (Default: 0.0 meaning no area is associated with upstream lump. If used within a duct macro, the FDAS and FUAS values will be defaulted to 0.0, 0.5, or 1.0 consistent with the discretization scheme. If NOT used within such a duct macro, no surface area will be included for this path unless a nonzero FUAS or FDAS is specified.)

FDAS. . . . . Fraction of interfacial surface area assigned to downstream lump (Default: 0.0 meaning no area is associated with upstream lump. If used within a duct macro, the FDAS and FUAS values will be defaulted to 0.0, 0.5, or 1.0 consistent with the discretization scheme. If NOT used within such a duct macro, no surface area will be included for this path unless a nonzero FUAS or FDAS is specified)

UVPU . . . . . vapor to interface heat transfer coefficient for upstream lump (used for calculating GUx. Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.). CUVPU contains the calculated value for references in logic or expressions.

UVPD.. . . . . vapor to interface heat transfer coefficient for downstream lump (used for calculating GDx. Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.) CUVPD contains the calculated value for references in logic or expressions.

ULPU . . . . . liquid to interface heat transfer coefficient for upstream lump (used for calculating GUx. Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.) CULPU contains the calculated value for references in logic or expressions.

ULPD . . . . . liquid to interface heat transfer coefficient for downstream lump (used for calculating GDx. Default: -1.0; must use IPDC=6, -1, -2, -3, or -4 for the default option else zero is returned.) CULPD contains the calculated value for references in logic or expressions.

Example:

```
PA TUBE,1,2,3, IPDC=6, UVPU=-2.0,UVPD=-2.0, GDB= 0.0
```

3.23.3 Additional Lump Variables

The following are additional variables available for use with any tank or junction, including those generated using GENLU, LINE, or HX.

In the following formats, the lower case “x” should be replaced by the letter identifier of the soluble gas (‘A’, ‘B’, etc.). Refer to Section B.7 for the equations governing homogeneous nucleation pressures and rates (ZRx, ZJx, PH, and FRHx). The additional variables are:

```
[,ASL=R][,GLx=R][,FRDx=R][,ZRx=R]
[,ZJ=R][,PH=R][,FRHx=R]
[,ULI=R][,UVI=R][,HENx=R][,IDGC=I]
```

where:

ASL . . . . . Lump interfacial surface area (Default: 0.0. Negative value scale with surface area of a spherical bubble.)

GLx . . . . . Lump gaseous mass transfer coefficient for species “x.” (Default: -1.0, will be calculated based on UVI and ULI if those variables are input.)

FRDx. . . . . Dissolution/evolution mass transfer rate for species “x.” (Default: 0.0, or input by the user using the IDGC constant as a flag)

ZRx . . . . . Rate constant for nucleation per unit volume for species “x.” (Default: calculated such that equilibrium concentrations are achieved in 10 milliseconds in the absence of other changes. Or ZR can be input by the user using the IDGC constant as a flag. *NOTE: if a junction is used, the VOL must be nonzero to use this option.*)

ZJ . . . . . Bubble nucleation rate. Z/J term in homogeneous nucleation pressure correlation (Section B.7.2). (Default: calculated internally, or input by the user using the IDGC constant as a flag)

PH . . . . . Homogeneous nucleation pressure (computed as described in Section B.7.2, or input by the user using the IDGC constant as a flag). May be negative, indicating tolerance of a supersaturated mixture.

FRHx. . . . . initial rate of production of species “x” from homogeneous nucleation (default for the initial rate: 0.0, and the current rate is calculated internally)

UVI . . . . . vapor to interface heat transfer coefficient (used for calculating GLx. Default: 0.0. Negative value scales with solid conduction.).

ULI . . . . . liquid to interface heat transfer coefficient (used for calculating GLx. Default: 0.0. Negative value scales with solid conduction.).

HENx. . . . . Effective Henry’s Constant for solute species “x.” “Effective” means the effects of multiple solvents has been included. (Default: 0.0, or calculated using MIXTURE DATA, or input on a lump-by-lump basis by the user using the IDGC constant as a flag). Units: psia or Pa, depending on UID (the CONTROL DATA, GLOBAL option). For zero solubility, set HENx to 1.0E30 if assuming control per the IDGC flag.

IDGC. . . . . Dissolved gas control constant for HEN, FRD, PH, ZR and ZJ calculations (Default: 11111. See explanation below.)

Example:

```
LU TANK,1, VOL=boxvol, IDGC=1111, HENF = 10230.0
```

Lumps usually inherit their surface area and other factors from paths that have pointed to them (via the path IUAS and IDAS parameters described in Section 3.23.2). However, user-input parameters for lump surface area (ASL) and either heat transfer coefficients (UVI and ULI) or mass transfer coefficients (GLx) can be specified either instead of or in addition to path-inherited mass transfer rates. These factors are useful for modeling nonduct components such as vessels (fuel tanks, storage cylinders, reservoirs or accumulators, etc.).

The ASL (interfacial surface area) defaults to zero. It if is input as a negative, then the actual surface area (CASL) is calculated assuming that ASL is a scaling factor times the surface area of the vapor volume assuming it is a spherical bubble or spherical droplet, whichever is smaller. (If a junction is used, VOL must be set for this default system to operate correctly.) In other words, for negative values of ASL, if AL is the current void fraction then:

$$CASL = -ASL*4*\pi* (3*VOL*MIN(AL,1-AL)/(4*\pi))^{2/3}$$

The ULI and UVI factors default to zero. If they are input as negatives, then the values used will be proportional to solid conduction -- they will be treated as a Nusselt number using volume over surface area as the characteristic dimension. In other words, if k_v is the conductivity of vapor, then for negative values of UVI:

$$CUVI = -UVI*k_v*(CASL/(AL*VOL))$$

Similarly, for negative values of ULI:

$$CULI = -ULI*k_l*(CASL/((1-AL)*VOL))$$

IDGC control constant usage: By default, most options governing dissolution and evolution mass transfer are calculated by the program, with the results available to the user for inspection or for use in logic or expressions. Alternatively, the user can assume control of various parameters. The IDGC constant (one per lump) governs whether the user or the program is calculating HENx, FRDx, PHx, ZRx, and/or ZJ.

IDGC is a 5 digit integer of the form “VWXYZ”, where:

V . . . . . controls the HENx value for all species
W . . . . . controls the FRDx value for all species
X . . . . . controls the PH value
Y . . . . . controls the ZRx value for all species
Z . . . . . controls the ZJ value

A value of “1” in any of the fields means the program will calculate the value. A value of zero (“0”) means the user has assumed responsibility for updating the value *for all soluble species*. Thus, the default of “IDGC=11111” means the program will calculate all of the required values. “If IDGC=10” on the other hand, the user has assumed responsibility for calculating all but the ZRx values. Because IDGC can vary from lump to lump, the user has extensive control over all calculations.

Caution regarding initial conditions (FLOW DATA): If a gaseous species is soluble, then *by default the initial amount of dissolved gas is zero.* If instead the user has directly specified the value of XF for the dissolved species, then that initial concentration will be used. However the user is cautioned against specifying a supersaturated initial condition since that state can lead to immediate nucleation. In the event of excessive concentrations, property calculation errors can occur.

There is no simple way to specify 100% saturated solute, nor any other fraction of dissolution saturation. Instead, estimate the desired XF for the solute, then make a preliminary run with a call to CNSTAB (Section 7.11.8) to find the saturation fraction.

3.23.4 Summary of Output Options

Four tabulation-style output routines are available for reporting the status of dissolved substances in the model (see Section 7 for more details). All have a single argument: the submodel name in single quotes, or 'ALL' for all fluid submodels.

GASTAB . . . . . For each tank and junction, and for each dissolved substance, GASTAB reports the values of PH, the mass transfer coefficients (GLx, CGLx, and the total G*A term including the GDx and GUx terms of attached paths), FRD, FRNCV, and HEN.

GASTB2 . . . . . For each tank and junction, and for each dissolved substance, GASTB2 reports the values of IDGC, PH, ZR, ZJ, FRH, ASL, and CASL.

GASATB . . . . . 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).

GASGTB . . . . . 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 (GUx, GDx, CGUx, CGDx).

HTLTAB . . . . . For each tank or junction, HTLTAB tabulates heat transfer coefficients (ULI, UVI, CULI, CUVI).

HTPTAB . . . . . For each tube and STUBE connector, HTPTAB tabulates heat transfer coefficients (ULPU, ULPD, UVPU, UVPD, etc.).

MAPHEN . . . . Tabulates solubility data input in MIXTURE DATA, converted into Henry's constants (in units of Pa or psia per *mass* fraction of solute), but neglecting effects of multiple solvents. To check the value at a given temperature or pressure, use the HENRY routine. Inspect the HEN constant if the effects of other solvents is needed.

HENRY . . . . . While not technically an output routine, this query routine may be added in user logic to find the current value of the Henry's constant (in units of Pa or psia per *mass* fraction of solute) for a particular mixture at a specific state.

3.24 Chemical Reactions (Reacting Flows)

If a mixture of several species has been defined as the working fluid for any particular fluid submodel, then any two or more of those constituents can be made to react within a control volume (tank) if their heats of formation have been defined (Section 3.21.2.5).*

To avoid tracking an excessive numbers of small species, one of the reactants can itself be an effective single-fluid representation of a collection of "hidden" species that continue to react at equilibrium: the internal composition changes from point to point or over time, but no extra mass conservation equations are required. Commonly, the effective replacement fluid represents hot products of combustion that continue to recombine as they cool. This collapsing of fluid species can be accomplished by using the variable molecular weight options for working fluids described in Section 3.21.7.5.

Reactions are invoked by specifying the rate of production or consumption of each species within one or more FLUINT tanks during a call to TRANSIENT (aka, "FWDBCK"), FORWRD, or STDSTL. Within STEADY (aka, "FASTIC"), these mass sink and source terms are ignored.

The mass rate of creation of a species is specified as its "XMDOT." For example, REACT.XMDOTG(100), or equivalently REACT.XMDOTG100, is the rate of creation (in kg/s or lb_m/hr, depending on UID) of species G in tank 100 of fluid submodel REACT. A positive XMDOT value means that this species is being created as the result of a reaction: it is a product. A negative XMDOT value means that this species is being consumed: it is a reactant. The XMDOTs for any one tank at any one time must sum to zero (or at least nearly so) in order to conserve mass.

XMDOTs can be specified (within logic or expressions) as functions of current temperature, pressure, partial pressures, etc. They can be functions of the amount of each species currently present in the tank, or the rate at which those species are currently flowing into the tank, or a combination of both. A utility routine, EQRATE (Section 7.11.9.9), is available to help calculate XMDOTs for certain common situations. See also CHGLMP_MIX: a method for instantaneously adjusting all constituents within a control volume.

Species with positive XMDOT will be generated at the current state (temperature, pressure, etc.) of the tank, and species with negative XMDOT will disappear at that same state. If a species current exists in two phases, it will be extracted or generated in proportion to that current phase distribution (weighted by mass). If instead the reaction should only be taking place in one phase but not the other, or if distinct reactions apply to each phase, perhaps twinned tanks (Section 3.25) should be used in combination with reaction options.†

---

* For library fluids, heat of formation is available for propane (290), ammonia (717), and propylene (1270) only. ***Other library fluids cannot be used as reacting species without generating a program abort.***

† Reactions are always confined to a single tank, even if that tank is a member of a set of active twins. Those reactions do *not* combine when the tanks homogenize: each tank's reactions are considered distinct. Therefore, special logic should be applied when using phase-specific reactions in the primary tank when the pair contains all vapor, or in the secondary tank when the pair contains all liquid.

Usually, a chemical equation is the basis of the relationship between each constituent's XMDOT values: their molar rates are related by their stoichiometric numbers. However, the values of XMDOT are in units of mass rate, not molar rate. Therefore, the molecular weights of each species (fetched perhaps by a call to VMOLW or VMOLWPT, or stored as a register value) must be used to convert into mass rates. For example, assume that hydrogen (H_2) is being combusted with oxygen (O_2) to produce water (H_2O):

$$H_2 + ½O_2 \rightarrow H_2O$$

or equivalently:

$$2H_2 + O_2 \rightarrow 2H_2O$$

Assume that oxygen is fluid 8732 (species "O") and hydrogen is fluid 8702 (species "H"), and that water is species "W" in this example. If the reaction is being limited by the rate of oxygen flowing into the tank at the rate of *froxy* (a register whose units are mass flow rate), then for tank #1 the following might be supplied in FLOGIC 0:

```
xmdotO(1) = -froxy
xmdotH(1) = 2.0*xmdotO(1)*vmolw(fi8702)/vmolw(fi8732)
xmdotW(1) = -(xmdotH(1)+xmdotO(1))
```

In the above logic, the rate of hydrogen consumption (xmdotH) is calculated based on the ratio of stoichiometric numbers and molecular weights. The rate of water production (xmdotW) is positive: the negative of the sum of two negative values. It could have been specified based on the stoichiometric numbers and molecular weights as well, but it is simpler and even more accurate to instead set xmdotW equal to the negative of the sum of xmdotO and xmdotH to conserve mass.

In fact, the code requires that Σ(XMDOT) = 0.0 for all species within any control volume at any time. If this criterion is not met within a specified tolerance,* the program will abort. If the values do not perfectly sum to zero, but are within the allowed tolerance, the supplied XMDOT values will be normalized such that they *do* sum to zero in order to conserve mass.

If the XMDOT for one species is negative, but that species is not *currently* present in the tank (even if it is currently flowing into it), then the XMDOT for that species will be set to zero. If that zeroing then causes the mass imbalance to exceed tolerances, the program will abort.

Because of the need for normalization and for zeroing absent reactant consumption rates, the user's imposed XMDOT values may not be exactly equal to those applied within the code during the next time step. Another cause of such differences (between input and effective XMDOT) is discussed in Section 3.24.2: FLUINT automatically reduces the reaction rates in proportion to the concentration of the limiting reactant in order to produce a stable solution.

If there are multiple chemical equations being obeyed, and if any one species is involved in more than one such equation within the same tank, then the final XMDOT value for that species should reflect the sum of all reactions in which it is involved.

* The maximum imbalance allowed is $0.1*|XMDOT|_{max}$.

For example, consider two reactions involving 5 species, for which the forward reaction rate at the current temperature and pressure has been calculated as registers *rate1* and *rate2*:

$CH_4 + H_2O \leftrightarrow CO + 3H_2$ (reaction 1, forward molar rate: “rate1”)

$CO + H_2O \leftrightarrow CO_2 + H_2$ (reaction 2, forward molar rate: “rate2”)

Assuming that the molecular weights of each species had been stored or precalculated as a register (e.g., “mw_ch4” is the molecular weight of CH4), and assuming that the following letter designations H, W, G, M, and C were used as species identifiers:

$H=H_2,\ W=H_20,\ G=CO,\ M=CH_4,\ C=CO_2$

then the logic might appear in FLOGIC 0* as:

```
c reaction 1
      xmdotm100   = -rate1*mw_ch4
      xmdotw100   = -rate1*mw_h2o
      xmdotg100   = rate1*mw_co
      xmdoth100   = 3.0*rate1*mw_h2
      xmdotc100   = 0.0             $ zero since not involved!
c reaction 2 (water-gas shift reaction: WGS)
      xmdotg100   = xmdotg100 - rate2*mw_co
      xmdotw100   = xmdotw100 - rate2*mw_h2o
      xmdotc100   = xmdotc100 + rate2*mw_co2
      xmdoth100   = xmdoth100 + rate2*mw_h2
```

In general, when multiple reactions are involved, the user should consider zeroing the rates initially, then summing into each species for each reaction in which it is involved.

As will be explained next in Section 3.24.1, FLUINT calculates the net heating rate for the reaction based on the properties of the reactants and products (HFORM and HSTP, per Section 3.21.2.5).

3.24.1 Heat of Reaction Correction (QCHEM)

If the user has specified the XMDOTs correctly, then FLUINT will automatically calculate the net heat rate of reaction (the negative of the power that would be required to keep the mixture isothermal). Summing over the a^{th} species:

$$\text{net heat rate of reaction} = -\Sigma\{ XMDOT_a * H_a(P_a,T) \} + QCHEM$$

* Using the *network logic* feature of FloCAD or Sinaps is especially useful, with the syntax “xmdotg100” being replaced by “xmdotg#this,” for example.

H_a is the a^{th} constituent's enthalpy at its current temperature and partial pressure P_a (or total pressure, for a liquid). As one fluid appears, it brings its enthalpy to the mixture. As another one disappears, it takes its enthalpy with it. If there are no chemical reactions, the basis (or zero point) of any species' enthalpy function is irrelevant since only differences matter. But as can be seen in the above equation, if one species used a very cold temperature as its enthalpy basis, and another used a very high temperature basis, then the latter species would appear to add or subtract a disproportionate amount of energy to the reaction. If the basis of each constituent's enthalpy function were such that its enthalpy H_a were equal to its heat of formation ($HFORM_a$) at the standard temperature and pressure (STP),* then QCHEM would be identically zero: no correction is necessary. Since each fluid is defined independently, a correction (QCHEM) is necessary to adjust for different enthalpy bases for each fluid based on the HFORM (and perhaps HSTP) inputs contained within each species' FPROP DATA block.

QCHEM is a lump output parameter summed (along with QL, QTM, etc.) into the tank's total heat rate, QDOT. QCHEM defaults to zero, and is available for inspection or use in logic or expressions, but it cannot be specified within FLOW DATA: it is an output parameter only.

QCHEM is calculated as follows:

$$QCHEM = -\Sigma\{ XMDOT_a * (HFORM_a - HSTP_a) \}$$

HFORM and HSTP (if it was not calculated by the program) are the heat of formation and the enthalpy at STP for that species, as supplied in its FPROP DATA block (if it is not a library fluid) as explained in Section 3.21.2.5.

There is another equivalent way to interpret the above calculations. If the reaction is taking place at high temperature, then net heat of reaction is calculated as if reactants were cooled to STP, as if heat were then released or absorbed at that condition (STP) based on the enthalpies of formation of each species, and as if the products were then heated back up to the current temperature and pressure.

$$\text{net heat rate of reaction} = -\Sigma\{ XMDOT_a * [HFORM_a + (H_a(P_a,T) - HSTP_a)] \}$$

As noted in other sections, the apparent XMDOT values used by the code may be different from the user input. These differences will affect the actual reaction rate applied over the time interval, which may vary slightly from the QCHEM value output for each lump.

3.24.2 Vanishing Reactants

FLUINT does not know whether the user's XMDOT values are based on quantities of species present in a tank, or on their rate of ingress. As noted above, a reactant must be currently present within the tank, and not just currently flowing into it. In other words, XMDOTa cannot be negative if XFa=XGa=0.0 for any one species "a."

* This is, in fact, the approach taken by some chemical equilibrium (Gibb's Free Energy minimization) software: HFORM = HSTP = $H_a(P_a,T)$ for fluid properties extracted from such programs.

However, consider a reactant present in only minor concentrations, such that it is the limiting reagent (the one that will be consumed first if the reaction were to continue). There are various mathematical problems that result from this situation, and they are dealt with automatically by the code, though they can also be dealt with by users in the form of carefully considered XMDOT logic. First, the types of mathematical problems that result will be explained by example, then the ways in which FLUINT automatically avoids such problems will be described.

To continue the above example from the previous subsections, a small (reaction-limiting) amount of oxygen is introduced into a stream of hydrogen. What if it were incorrectly assumed instead that hydrogen were the limiting reactant, flowing in at a rate of *frfuel*? The logic might appear as follows:

```
xmdotH(1) = -frfuel
xmdotO(1) = 0.5*xmdotH(1)*vmolw(fi8732)/vmolw(fi8702)
xmdotW(1) = -(xmdotH(1)+xmdotO(1))
```

The amount of oxygen (XGO, PPGO) would then be driven to a very small value, and if the code took no other measures, it would eventually hit zero and cause a program abort per the rules defined above. Perhaps the user could modify the above equations to make the rates drop off as either hydrogen or oxygen disappeared from the control volume. For example, the following logic drops the reaction rate in proportion to the mole fraction (expressed as partial pressure fraction) below 1%, whether oxygen or hydrogen is the limiting quantity:

```
xmdotH(1) = -min(1.0,100.0*ppgH1,100.0*ppgO1)*frfuel
xmdotO(1) = 0.5*xmdotH(1)*vmolw(fi8732)/vmolw(fi8702)
xmdotW(1) = -(xmdotH(1)+xmdotO(1))
```

Unfortunately, if the code took no other measures to avoid problems, the above logic would cause the oxygen concentration to oscillate wildly at a very small value since the XMDOT for each species is held constant over each interval. This would cause the code to restrict the time step in an attempt to control changes happening over each interval.

By writing the XMDOT logic as a function of the amount of in-flowing oxygen per the logic in the previous section, some of these problems would be *temporarily* avoided:

```
xmdotO(1) = -froxy
xmdotH(1) = 2.0*xmdoto(1)*vmolw(fi8702)/vmolw(fi8732)
xmdotW(1) = -(xmdotH(1)+xmdotO(1))
```

But in the above case, any small amount of oxygen that existed initially in that control volume would still disappear over time, since any incoming oxygen is completely combusted: nothing replenishes the original amount oxygen, so it is swept out of the tank as time progresses. Eventually,

XGO would be driven to zero. An alternative would be to assign a "combustion efficiency" to the process: a fraction of completeness of reaction.* For example, assuming 97% reaction efficiency leaves a nonzero level of residual oxygen:

```
xmdotO(1) = -froxy*0.97
xmdotH(1) = 2.0*xmdoto(1)*vmolw(fi8702)/vmolw(fi8732)
xmdotW(1) = -(xmdotH(1)+xmdotO(1))
```

Even a small amount of residual reactant helps: a reaction efficiency of 0.999, for example, is sometimes enough.

While careful consideration of the above methods is well worth the user's time (especially the use of finite reaction efficiencies if XMDOT rates are based on inflow), FLUINT performs corrections by default in order to help avoid such problems. It implements automatic control over reactions based on the limiting reactant in case the XMDOT logic has not been so carefully considered. Smoother executions and larger time steps result, but at a cost of correcting (effectively damping) user-supplied reaction rates.

In a nutshell, **FLUINT automatically scales the *effective* XMDOT and QCHEM values in proportion to the mass concentration of the limiting reactant.** For example, if oxygen were the limiting reactant, then if the XGO halves over the next time step, the *effective* reaction rate is also halved automatically ... as if the user-specified XMDOT and the program-calculated QCHEM values had all been multiplied by 0.5, *even though those values will not appear to have been adjusted if inspected in output, logic, or expressions*. (If XGO doubles, so will the effective ratio rate.)

The large magnitude of the numbers used in that last example were chosen for clarity, but may cause undo concern. The automatic stabilization is, in fact, not often even noticeable in results. One reason for its transparency is that actual changes to the concentrations of limiting reactants is normally very small per time step,† such that the stability correction is also normally very small.

The other reason that the user should not be unduly concerned is that most reaction kinetics follow a similar trend: it is as if FLUINT is implicitly anticipating an upcoming change to XMDOTs in the subsequent time step. The generalized reaction rate between substances A and B often takes the following form:‡

$$\textit{rate (proportional to XMDOT)} = -k[A]^a[B]^b = -(Ce^{-E/RT})[A]^a[B]^b$$

In other words, the reaction rate is often proportional to the concentrations of species A and B (represented by [A] and [B]), the rate constant k (which is proportional to an Arrhenius term containing the activation energy E, the gas constant or other constant R, and the absolute temperature

* See the EQRATE utility, Section 7.11.9.9, for help modeling reaction efficiencies.

† In fact, a maximum concentration change limit for each species is a criterion of the time step selection itself, virtually guaranteeing that stability adjustments to the reaction rates will be small.

‡ This equation is *not* applied by FLUINT, though it could be the basis of the user's XMDOT values. This equation is only presented as rationale for the reaction-limiting methods that FLUINT will apply to user-supplied (or EQRATE-calculated) XMDOT values. If there is a temperature-dependence on the rate constant, or non-unit exponents on the concentrations, then it is the user's responsibility to include those effects in the calculation of XMDOT values.

T). The exponents *a* and *b* are usually near unity, *which is what FLUINT assumes for the implicit solution of the equation set during each time step.* By default, FLUINT adjusts of the reaction rates as a function of either [A] or [B] (whichever is limiting).

The automatic adjustment of the reaction rates described above can be eliminated by calling (from any location):

```
CALL CHEMDAMP(0.0)
```

To restore them to their full level of activity:

```
CALL CHEMDAMP(1.0)
```

Intermediate values of the CHEMDAMP argument (say, 0.5) will achieve intermediate results. For example, a value of 0.5 means to reduce the reaction rate by 25% given a 50% reduction in the concentration of the limiting reactant. For the general case, where F is the damping factor listed in the CHEMDAMP calls above, X_{lim} is the mass fraction of the limiting species, and ΔX_{lim} is the change over the time interval:

$$QCHEM_{eff} = QCHEM_{calc} * (1 + F * \Delta X_{lim} / X_{lim})$$

$$XMDOTa_{eff} = XMDOTa_{input,\ normalized} * (1 + F * \Delta X_{lim} / X_{lim}) \qquad \text{for each species a}$$

3.25 Modeling Phasic Nonequilibrium using Twinned Tanks

A fundamental FLUINT assumption is that any lump is characterized by a single equilibrium thermodynamic state. In fact, that is the *definition* of a lump, just as a SINDA node is defined as a point with a single temperature. The assumption is equivalent to one of perfect mixing within each lump, meaning that the heat transfer and mixing processes are assumed to occur over a negligibly small time scale compared to the system solution. While that assumption is normally justified and greatly enhances solution speed, it can be inappropriate in some instances.

Within a FLUINT tank containing both liquid and either gas or vapor (e.g., $0 < XL < 1$), the perfect mixing assumption means that any changes or additions to either phase *immediately* affect the lump pressure—the heat and mass transfer rates between phases are assumed to be infinite. (If mass transfer due to phase change is low in multiple constituent flows, these effects are less noticeable.) In Sample Problem C (the pressure cooker), this assumption causes the pressure in the cooker to drop exponentially with time because even a little subcooled liquid drastically reduces the equilibrium saturation pressure. In many single-constituent (pure two-phase fluid) cases, such perturbations are unstable. Slight injection of liquid can lead to catastrophic collapse of the vapor since the liquid influx will increase as the control volume pressure drops. In a duct macro built with tanks, internally applied spatial acceleration terms usually prevent such behavior even though minor oscillations occur: sharp flow rate gradients are "discouraged" since they cause self-correcting pressure gradients. Also, the large degree of mixing within duct flow prevents the phases from having different temperatures: modeling phasic nonequilibrium is usually unnecessary in evaporators and condensers.

In fact, with a few important exceptions listed below, the difference in temperature between liquids and vapors need only be modeled in quasi-stagnant or one-port (non flow-through) control volumes such as storage vessels, accumulators. Typically, modeling nonequilibrium becomes important in line flow only when

1. transient pressure wave propagation through a two-phase line is important, or
2. when stratified flow and huge temperature gradients exist, such as large diameter horizontal lines partially filled with cryogenic fluids (or at least room temperature fluids exposed to a comparatively hot environment),* or
3. in axially stratified fill and purge models (Section 3.27).

This subsection describes methods for simulating the lack of thermal equilibrium between liquid and vapor in the same control volume (whether an isolated control volume or one within a duct macro) using a pair of twinned tanks.

* Horizontally stratified line modeling creates two additional challenges: circumferential heat transfer gradients and quenching as a liquid from progresses. See Section 3.25.8.1.

3.25.1 Overview

Each path has a single characteristic flow rate, and each lump has a single characteristic thermodynamic state.

To model slip flow, in which vapor travels at one flow rate and liquid at another within a passage, tubes or STUBE connectors can be *twinned* (Section 3.17), a process in which a pair of paths (one for liquid, one for vapor) are created and manipulated together, sometimes even treated as a single effective path. When two-phase flow is present, the primary path carries liquid and the secondary carries gas and/or vapor, but if the flow is single-phase (or nearly so), then the secondary path disappears temporarily and is ignored.

To model phasic nonequilibrium, in which the gas and liquid within a two-phase control volume may not be at the same temperature (and in some instances not the same pressure), twinned tanks are required. They follow a similar approach to twinned paths: the primary tank mostly contains the liquid, and a secondary tank mostly contains the gas/vapor. When the tanks are single-phase liquid or vapor (or nearly so), then the secondary tank disappears temporarily and is ignored.

Twinned tanks are linked together automatically with a FLAT iface (Section 3.8.3) and a CONSTQ ftie (Section 3.7). The two tanks may optionally exchange mass and constituents with each other using a *superpath*: a collection of specialized mass transfer *subpaths* described later. Each subpath in a superpath is used internally by the program to model vaporization, condensation, bulk fluid movement, dissolution, and evolution.

Additional independent flow paths may be added as needed by the user. These additional paths can be attached to either of the twinned tanks individually, but to connect to *both* tanks simultaneously requires a twinned path to be specified. Analogously, additional ties and fties may be connected to either tank using single elements, or to both tanks simultaneously using twinned ties and twinned fties.

General rules of operation for twinned tanks are as follows:

1. The VDOT and COMP values on the secondary tanks are always ignored (analogously, the DH, TLEN, etc. of secondary tubes or STUBEs are also ignored). Internally, the code sets them to be the same values as those of the primary tank. Because the two tanks are interconnected via an iface, *changes to the VDOT or COMP value of the primary tank will automatically affect both tanks.* Therefore, if the COMP value is a function of volume, then the value applied to the primary tank should be based not on *its volume*, but on the *total volume* of the pair of twinned tanks.

2. Additional ifaces may be applied to the primary tank or the secondary tank. If they are applied to the vapor (secondary) tank and only vapor or gas exists, they will be temporarily moved to the primary tank. Subvolume overdetermination rules (Section 3.8.1) apply.

3. Individual (untwinned, or singlet) paths may be applied to the primary tank or the secondary tank. If they are applied to the vapor (secondary) tank and only vapor or gas exists, they will be temporarily moved to the primary tank.

4. Twinned paths attached to the primary tank are understood to apply to both tanks simultaneously. These paths are almost always tubes, since STUBE connectors are often unstable because of their instantaneous reaction speed.

5. Analogously, individual (untwinned or singlet) ties and fties may be applied to the primary tank or the secondary tank. If they are applied to the vapor (secondary) tank, then if the control volume contains only vapor they will be temporarily moved to the primary tank.

6. Twinned ties and twinned fties attached to the primary tank are understood to apply to both tanks, albeit with different QTIEs and perhaps different UAs, depending on the type. Refer to tie apportionment rules described in Section 3.6.

There are several important differences between isolated (individually input) pairs of twinned tanks and those generated along with a duct macro:

1. When twinned tanks are used, the macro paths must be twinned as well (including in flat front fill/purge models).

2. In HX macros, the generated HTN, HTNC, or HTNS ties must also be twinned if tanks are twinned. These twinned ties will use the flow regime map to determine apportionment of heat flow between each phase and the wall (Section 3.6).

3. Regime mapping must be selected: either the default of IPDC=6 or negative IPDC must be used in duct macros that employ twinned tanks.

4. Additional paths and ties and fties and ifaces may be added to a duct macro as needed.

3.25.2 Defining the Interface

Note: Section 3.23 is a prerequisite for the following discussion. The user should note that ***in duct macros, all of the factors are largely taken care of by an extensive default system***, and that the following discussion is relevant only to advanced duct macro usage or to manipulations of an isolated (nonmacro) pair of twinned tanks, where data regarding the interface and heat and mass transfer coefficients must be supplied either directly and/or by association with twinned paths.

When soluble species are absent, then only *one* liquid/vapor interface is relevant: the nonisothermal one between the two tanks. This surface area is composed of a user-supplied component (AST, the analog of ASL described in Section 3.23.3 for any tank independent of whether or not it is twinned) as well as the flow area inherited from twinned paths that are attached to the twinned tanks (per the rules governing ASPD, ASPU, FUAS, FDAS, etc. described in Section 3.23.2). The heat transfer coefficients that apply to this main interface are similarly composed of a user-supplied component (UVT, ULT) as well as the coefficients inherited from twinned paths that are attached to the twinned tanks (per the rules governing ULPD, UVPU, etc. described in Section 3.23.2).

With soluble species present, some additional rules apply. When both tanks in a twinned pair are active, there can exist up to *three* liquid/vapor interfaces, each with its own mass transfer rates (Figure 3-43).

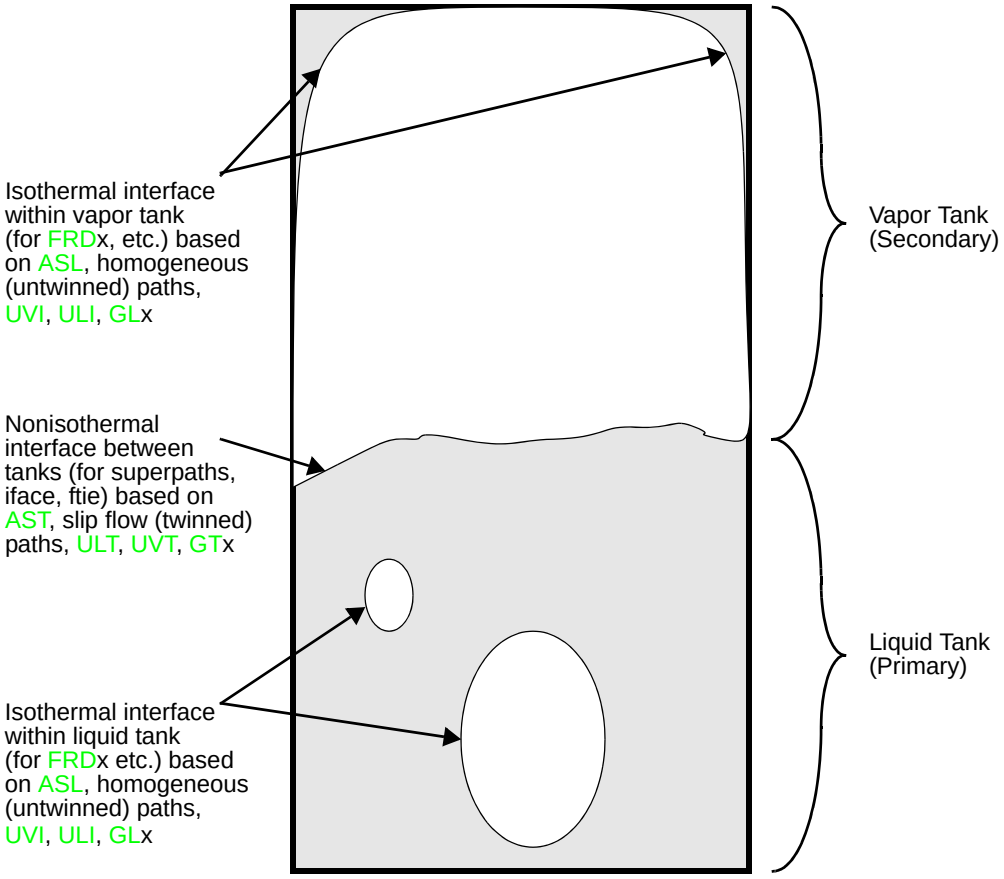


Figure 3-43 Twinned (Nonequilibrium) Tanks

Within each tank, an isothermal user-specified surface area (ASL), interface heat transfer coefficients (UVI and ULI), and mass transfer coefficients (GTx, where x is the species letter identifier) exist as they do for any tank (Section 3.23.3). Species flow rates within each are designated FRDx. (Because they occur *within* a tank and not *between* them, these flow rates are not represented by paths, and can be addressed in logic or expressions using the tank ID number as described in Section 3.23.3). These terms can be used to model bubbles within the liquid and/or droplets within the vapor. In other words, finite-rate mass transfer can occur even when the phasic temperatures are equal *within* a tank (or junction, for that matter).

In addition, mass transfer can occur across the “main” (nonisothermal) interface between the two tanks (i.e., the one influenced by AST and the twinned paths that are attached to the tanks). Once again, this mass transfer is governed by a user-specified surface area (AST), interface heat transfer coefficients (UVT and ULT), and mass transfer coefficients (GTx, where x is the species letter identifier). All of these parameters function analogously to the parameters ASL, CASL, UVI, CUVI, etc. *All are attributes of the primary tank (i.e., can be addressed in logic or expressions using*

the primary tank ID number). Refer to Table 3-17, and compare with the top part of Table 3-16.

Table 3-17 Summary of Scalable Twinned Tank Interface Variables

Element	Variable Name	Output Reference	Description
primary tank	AST	CAST	surface area at the interface (between the lumps)
	GTx	CGTx	Fick's Law constant for dissolution, species x
	UVT	CUVT	vapor to interface heat transfer coefficient (used for calculating GTx as well as interface vaporization and condensation)
	ULT	CULT	liquid to interface heat transfer coefficient (used for calculating GTx as well as interface vaporization and condensation)

If the vapor tank contains both a condensible/volatile species and one or more noncondensible species, then the vaporization and condensation of the condensible/volatile species can be slowed due to a mass transfer limit: diffusion of the condensible/volatile species through the noncondensible species must be taken into account. This is accomplished automatically. (The internal routine used is HTFDIF, documented in Section 7.11.4.) For either vaporization or condensation, the effective CULT will be somewhat reduced due to this effect. For example, if ULT is positive, then CULT < ULT. Specifically, CULT = f*ULT where f is number smaller than unity which accounts for diffusion-limited heat transfer in the vapor side (despite being applied to the liquid side).

When a path (almost always a tube) is linked to a tank via the IUAS and IDAS options (which are automatically generated by duct macros), then path's contributions to the tank's heat and mass transfer (surface area, heat transfer rates, etc. as predicted by the path flow regime) are included, usually eliminating any need to add additional ASL/GLx etc. terms. When a twinned tube is linked to a twinned tank, these "contributions" *apply to the main interface,* but not to the isothermal interfaces within the primary tank and within the secondary tank, which are left available for user manipulations. In other words, by default (and neglecting any vaporization carry-over of solutes into the vapor space) the FRDx terms for both the primary and secondary tank will be zero since ASL defaults to zero. However, when an untwinned tube is applied to either of these tanks, however, its contributions (per the IUAS and IDAS options) will be made to the isothermal interface within the tank to which it is attached, and not to the nonisothermal interface between the twinned tanks. Mass transfer between twinned tanks must be handled by paths, not by internal tank terms such as FRDx. To handle such effects, a special concept called a *superpath* exists. A superpath may be thought of as a set of parallel *unnamed* MFRSET-like paths, each transporting a different species or perhaps a different phase. Since there are potentially many such paths (2+S+V, where S is the number of soluble gases and V is the number of volatile species), the user need not name each one. Refer to Section 3.25.4 below.

Because the main interface is not isothermal, heat transfer can occur across it because of the automatically generated ftie; otherwise, the interface would have been adiabatic. Condensation and vaporization can occur *within* each lump as a result of other boundary conditions (QL, and the effect of any ties).

Even without condensation/vaporization, and without evolution/dissolution, phases can move across the boundary between the two tanks. The primary tank variables VDRP and VBUB control this movement. VDRP is the velocity of droplets in the vapor (secondary) tank towards the interface, and VBUB is the velocity of the bubbles in the liquid (primary) tank towards the interface.* These default to 3600 ft/hr (1 ft/s, 0.3048 m/s), but can be changed by the user.† The intent of these variables is to keep the phases separated. They cause a flow rate in the superpaths in proportion to the void fractions in each tank and to the total interface area of the nonisothermal (main) interface between the two tanks. This flow rate diminishes to zero as either the droplets and/or bubbles disappear, or as the interface disappears. See also TLIQ and TVAP paths within superpaths (Section 3.25.4).

Figure 3-44 illustrates a typical network surrounding twinned tanks.

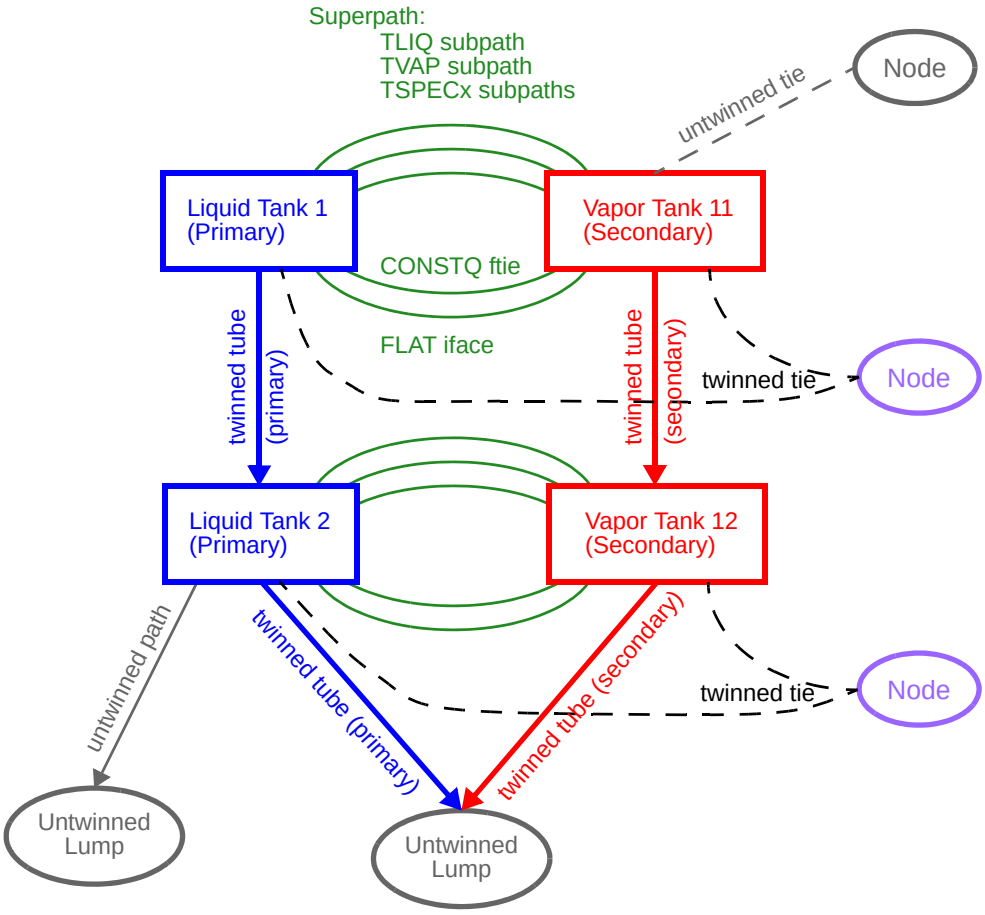


Figure 3-44 A Typical Twinned Tank Network

* Negative VDRP and VBUB inputs are a signaled use of an alternate meaning: keep the tank from being too wet (VDRP) or too dry (VBUB), as described later. VDRP = -0.999, for example, signals the program to maintain no less than AL=0.999 in the presence of liquid injection or continuous condensation.

† The default value for VBUB is unrealistically high for most applications, and a lower value of bubble rise velocity should be considered. The use of the alternative input of VBUB = -0.001 is also encouraged.

For stand-alone twinned tanks (i.e., those that are not part of a duct macro representing flow in a pipe, but rather are used to represent large two-phase vessels such as dewars, boilers, reservoirs, etc.), the elevation of the liquid/vapor surface should be set when body forces are present. Path end locations (Section 3.13.4), and twinned HTP ties with the DEPTH parameter varied (Section 3.6.3.3) are also relevant.

3.25.3 Input Formats

Additional inputs/variables within the lump or macro subblock. Note that the inclusion of "LTWIN = ..." is what causes a tank to be twinned and for the other parameters to have meaning:

```
[,AST = R] [,UVT = R] [,ULT = R]
[,VDRP = R] [,VBUB = R]
[,LTWIN = I] [,TLV = R]
[,IDFACE = I] [,IDPATH = I] [,IDFTIE = I]
[,GTx = R]
```

where:

AST . . . . . Interface surface area (in addition to that inherited from adjacent tubes or STUBE connectors). Defaults to 0.0, meaning no additional heat or mass transfer will occur across the main interface (other than that portion inherited from adjacent paths). Specify -1.0 to use a spherical bubble approximation (see ASL), or a negative multiplying factor. Refer to CAST in logic or expressions as the calculated value. Affects heat and mass transfer as well as bubble and droplet movement between the two tanks.

UVT* . . . . . Vapor to interface heat transfer coefficient (applies to the main interface). Defaults to -1.0, meaning essentially solid conduction. Specify a negative multiplying factor or a positive value to use directly. Refer to CUVT in logic or expressions as the calculated value. Affects the flow rate in the TSPECx part of the superpath, where x is a volatile species. If not part of a duct macro, UVT should be corrected to include diffusion-limited condensation and vaporization, perhaps using the HTFDIF utility (page 7-297).

ULT . . . . . Liquid to interface heat transfer coefficient (applies to the main interface). Defaults to -1.0, meaning essentially solid conduction. Specify a negative multiplying factor or a positive value to use directly. Refer to CULT in logic or expressions as the calculated value. Affects the flow rate in the TSPECx part of the superpath, where x is a volatile species.

* Despite describing vapor-side factors, references to VDRP, UVT, and CUVT in logic or expressions must use the primary (liquid) tank ID and not the secondary (vapor) tank ID to be consistent with all other common twinned tank variables. This same scheme applies to twinned paths: use the primary element identifier.

VDRP . . . . . Velocity of droplets in the vapor tank towards the main (nonisothermal) interface, with droplets assumed to be uniformly distributed throughout the volume. Default: 3600 ft/hr (1 ft/s) or 0.3048 m/s. VDRP affects the flow rate in the TLIQ part of the superpath. If VDRP is essentially infinite (>1.0e10), then the program switches to attempting to maximize the void fraction of the secondary gas/vapor tank, as explained next.

If the VDRP value is negative, this signals an alternative meaning: maximize the void fraction (AL) in the secondary tank in the presence of liquid injection or continuous condensation, but don't allow a void fraction of less than |VDRP|. For example, setting VDRP = -0.999 means the void fraction will not drop below 99.9%, and the TLIQ subpath's flow rate will be calculated as required. Use of negative VDRP is strongly recommended in cases where twinned tanks really shouldn't be used due to excessive condensation, but are chosen anyway.

If the liquid is not volatile, then a minimum liquid volume fraction of 0.001 must exist, so negative values of |VDRP| greater than 0.999 such as VDRP = -0.9999 (or VDRP > 1.0E10) will be treated as if VDRP = -0.999 had been input.

VBUB . . . . . Velocity of bubbles in the liquid tank towards the main (nonisothermal) interface, with bubbles assumed to be uniformly distributed throughout the volume. Default: 3600 ft/hr (1 ft/s) or 0.3048 m/s. VBUB affects the flow rate in the TVAP part of the superpath. If VBUB is essentially infinite (>1.0e10), then the program switches to attempting to minimize the void fraction of the primary/liquid tank, as explained next.

If the VBUB value is negative, this signals an alternative meaning: the minimize the void fraction (AL) in the primary tank in the presence of vapor/gas injection or continuous boiling, but don't allow a void fraction greater than |VBUB|. For example, setting VBUB = -0.01 means the void fraction will not rise above 1%, and the TVAP subpath's flow rate will be calculated as required. Use of negative VBUB is strongly recommended in cases where twinned tanks really shouldn't be used due to excessive boiling, but are chosen anyway.*

If the liquid is not volatile, then a minimum gas volume fraction of 0.01 must exist, so negative values of |VBUB| smaller than 0.01 such as VBUB = -0.001 (or VBUB>1.0E10) will be treated as if VBUB = -0.01 had been input.

If instead the liquid is contains a thermodynamically compressible liquid (6000 series COMPLIQ fluid), then a minimum gas/vapor volume fraction of 0.001 must exist, so negative values of |VBUB| smaller than 0.001 such as VBUB = -0.0001 (or VBUB>1.0E10) will be treated as if VBUB = -0.001 had been input.

In other words, use of an incompressible yet volatile liquid is the only case in which zero void fraction can be maintained in the primary tank instead

* When boiling or gas injection is extreme, a "TWIN TANK PHASE SWEEP LIMIT" on the time step may be experienced. This is an indication that the twinned tank concept is being over-exerted, and that a single perfectly-mixed tank (or a combined twin using the NOTWIN routine) would be a better choice.

of a minimum void.

LTWIN . . . . . The ID of the secondary tank. The presence of “LTWIN” causes the tank to be twinned, and the above factors (GTx, AST, etc.) to have meaning.

TLV . . . . . The initial temperature of the vapor/ullage (secondary) tank. Default: TL. If this parameter is not specified, then the twinned tanks will be initially in the combined (equilibrium) mode. If it is specified (usually TLV>TL), then the primary and secondary tanks will be initialized separately, and the first solution is expected to be a transient (and not STEADY). Further guidance is provided in Section 3.25.6.2. **Use of both AL and PL! are strongly recommended when using the TLV input option.**

IDFACE . . . . . The ID of the FLAT iface to be placed between the primary and secondary tanks. Required if LTWIN is input. Otherwise, input zero if some other type of IFACE is required that will be input separately. In used within a duct macro or if twinned paths are attached to this lump, then only FLAT ifaces can be used, in which case IDFACE cannot be zero (or at least, if it is zero a FLAT iface must be generated separately or an error will occur).

IDPATH . . . . . The ID of the superpath to create between the primary and secondary tanks. Defaults to primary lump ID if LTWIN is input.

IDFTIE . . . . . The ID of the CONSTQ ftie to create between the primary and secondary tanks. Defaults to primary lump ID if LTWIN is input. Input zero (no ftie generated) to neglect inter-phase heat transfer along with any associated vaporization and condensation.

GTx . . . . . Mass transfer conductance for soluble species x across the main interface. Defaults to -1.0, meaning calculated automatically using UVT and ULT. Specify a positive value to override the default calculation. Specify -10.0 to augment the default calculation by a factor of 10, etc. Refer to CGTx in logic or expressions as the calculated value. Affects the flow rate in the TSPECx part of the superpath, where x is a soluble species.

Additional inputs within GENLU, LINE, or HX macro subblocks:

```
[,LTINC = I]
[,IFINC = I]
[,IDPINC = I]
[,IDFTNC = I]
```

where:

LTINC . . . . . LTWIN increment (default: LUINC)

IFINC . . . . . IDFACE increment (default: LUINC)

IDPINC . . . . . IDPATH increment (default: LUINC)

IDFTNC . . . . . IDFTIE increment (default: LUINC)

Each pair of a twinned tank must exist at the same level of duplication. In other words, there can exist no difference in duplication factors *between* a pair of tanks. Thus, the DUP factors of any superpaths, ifaces, and fties generated between a primary and secondary tank must be unity at all times.

3.25.4 Mass Transfer Superpaths

Superpaths consist of a set of two or more "unnamed" MFRSET-like *subpaths* that occur only between twinned tanks. One numeric user ID is used to generate all of the underlying subpaths.

As a minimum, two subpath connectors of type TLIQ and TVAP are created which will be used by the program to model bulk fluid motion between the two tanks. These two paths distinguish between phases, but not between constituents. The TLIQ subpath transports excess liquid from the vapor twin to the liquid twin according to VDRP, the velocity of droplets toward the main interface (or minimum void fraction of the secondary tank). The TVAP subpath transports excess vapor (and any gases) from the liquid twin to the vapor twin according to VBUB, the velocity of bubbles toward the main interface (or maximum void fraction of the primary tank void).

If a volatile species (library, 6000, or 7000 series) exists and a CONSTQ ftie has been generated, then an additional subpath of type TSPECx will be created, where x is the letter identifier of the volatile constituent. This subpath will be used by the program to model vaporization or condensation at the interface. If only a single (pure) condensible/volatile species is present and only "FID=" has been used to specify this substance, then the letter identifier is "A" by default such that the vaporization and condensation flow rates can be found in the TSPECA subpath.

For each soluble species, an additional path of type TSPECx will be created to model dissolution or evolution rates, where x is the letter identifier of the soluble species. The flow rates of these particular subpaths correspond to the FRD factors describing dissolution rates within a lump. However, since these flows occur *between* two lumps (and not *within* a single lump as do the FRD terms), they must be represented by separate paths or in this case, subpaths. For TSPECx paths representing dissolution/evolution (but *not* condensation/vaporization), the STAT flag will appear in tabulations as "XFR," which is a special internal phase suction flag intended to handle the transfer of gas directly into liquid and vice versa.

Flow rates of subpaths are always positive from the secondary (vapor) tank to the primary (liquid) tank (maintaining the convention for FRD, which is positive for dissolution). This means condensation will be positive, and vaporization negative. The user can refer to the flow rates within each of these subpaths in logic or expressions as:

FLIQ(m) . . . . . the bulk liquid flow rate (TLIQ subpath) in superpath m
FVAP(m). . . . . the bulk vapor flow rate (TVAP subpath) in superpath m
FSPx(m) . . . . . the flow rate of species x in superpath m

For example, by creating a superpath named "103," the user is implicitly creating several subpaths that can be referred to only indirectly using the ID of the superpath: FLIQ103, FSPW103, etc. These parameters are available for inspection of calculations only: they cannot be set directly by the user. (Of course, additional paths can be placed between primary and secondary twins.)

The user *cannot* refer to traditional path values such as FR(m), GK(m), etc. for subpaths, nor can the routine INTPAT be used for superpaths. To look up the internal address of a subpath within a superpath in logic, use INTPTS('smn',m,ARG) where ARG = 'x' (where x is the letter identifier of the desired species), 'VAP', or 'LIQ.' The returned value may then be applied to the FR value of the unnamed subpaths, which is where FLIQ, FVAP, and FSPx are actually stored.

For example, below are two ways to refer to the liquid flow rate in superpath 103 in submodel "subname:"

```
        XTEST   = subname.FLIQ103      $ or subname.FLIQ(103)
F       XTEST   = FR(INTPTS('subname',103,'LIQ'))
```

In EZXY and in postprocessing options within either Thermal Desktop or Sinaps, the ID of each path is listed, not the type. Therefore, the internal (and normally hidden) subpath ID that is automatically generated by SINDA/FLUINT is not only exposed to the user, it becomes the *only* way to refer to that path. As an aid in finding the right path inside of a postprocessing selection, the actual/hidden subpath ID system is documented below.

Whatever the fluid mixture, one TLIQ and one TVAP subpath are created. The TLIQ subpath has an ID of "980000000 + m," where m is the superpath ID. Similarly, the TVAP subpath is named "990000000 + m."

If, in addition, a volatile species "x" exists, then a TSPECx subpath is created. Additional TSPECx subpaths are created for each soluble gas species as well. The number of each species' letter designator in the English alphabet is used to create the subpath ID (A=1, B=2, C=3... X=24, Y=25, Z=26). If a volatile species V exists in the mix, then for superpath m, the TSPECv subpath ID is "220000000 + m" since V is the 22nd letter in the English alphabet.

Using m=103 as an example superpath ID, generated subpath IDs might be as shown below (depending on the species defined):

```
 10000103 ..... TSPEC subpath ID for species A (FSPA103)
 20000103 ..... TSPEC subpath ID for species B (FSPB103)
     ...
150000103 ..... TSPEC subpath ID for species O (FSPO103)
     ...
980000103 ..... TLIQ subpath ID (FLIQ103)
990000103 ..... TVAP subpath ID (FVAP103)
```

As a reminder, the above IDs are normally hidden and *cannot* be used to access the path flow rate in logic or expressions. Use "FRSB103" (and *not* "FR20000103") to refer to the flow rate in the subpath for species B in superpath 103.

3.25.5 Execution Control Options

The routines NOTWIN, GOTWIN, and SPLIT_TWIN (Section 7.11.1.3) are available to enable the user to force one or all pairs of twinned tanks to stay homogeneous. However, there is some interplay with the path control routines GOSLIP and NOSLIP. See also Section 7.11.1.3.

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 or SPLIT_TWIN to 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 which are not attached to outflowing twinned paths may need to be separately homogenized using a call to NOTWIN.) Subsequently, to reenable just slip flow modeling, call GOSLIP. To reenable both slip flow and nonequilibrium modeling, also (*subsequently*) call GOTWIN. Calling GOTWIN may be enough to reenable 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.

To see if a pair of twinned tanks is currently in the combined (homogeneous) mode or the separated (nonequilibrium) mode, check the volume of the secondary tank. If it is zero, the tanks are in the combined (homogeneous) mode. If it is positive, they are separated (nonequilibrium).

GOTWIN and SPLIT_TWIN both return one or more tanks to nonequilibrium mode. The difference between the two is that GOTWIN *enables* future splitting (perhaps as soon as the next time step or solution step), whereas SPLIT_TWIN causes an immediate split into two tanks, even changing the void fraction if the primary tank is too wet or too dry to otherwise permit splitting. SPLIT_TWIN is primarily used to initialize twinned tanks in OPERATIONS, as will be described next.

3.25.6 Initializing Twinned Tanks

Before OPERATIONS begins, the initial state of a twinned tank is assumed homogeneous if TLV has not been specified, and the TL, PL, XL, AL, etc. of the primary tank determines its initial state as with any lump. This is suitable if the first solution routine called is STEADY (“FASTIC”), which is usually true.

If instead a transient (or STDSTL) is to be called first, *and* if the initial conditions of the control volume should *not* be at equilibrium (if, for example, the gas/vapor phase should start at a warmer temperature than that of the liquid), then either the TLV (vapor/ullage temperature) input option should be used, or the SPLIT_TWIN utility (Section 7.11.1.3) should be called.

If neither the TLV input option is used nor the SPLIT_TWIN utility is called, *and* if the initial void fraction is within limits (1% to 99.99%,* as determined by the XL or AL of the primary tank), then the tanks will be *later* split (nonequilibrium) mode at the start of the next transient time step. Because of the void fraction limit, a tank that is single phase or nearly so will start in the combined (equilibrium) mode. If the user requires that the tanks can separate in a later transient solution, but does not need the gas or vapor phase to start at a warmer temperature than the liquid phase, he or she should assure that AL is not too high or too low so as not to exclude that mode.

3.25.6.1 Summary of Initialization Utilities

If the user needs to perform initializations starting with combined (homogenized) tanks, and is unsure of its status, the NOTWIN routine can be used to combine the twins. Since NOTWIN will also prevent such twins from later splitting, a subsequent call to GOTWIN should be considered after initializations. GOTWIN will not split the tanks (unlike SPLIT_TWIN), rather it will *reenable* them to be split later during network solutions.

In some cases (especially when using twinned tanks to model vessels), starting a transient with tanks that are already split is required, with the vapor/gas at a different temperature than the liquid. In this case, either use the TLV input option (described below), or call SPLIT_TWIN *then* call CHGLMP and CHGVOL for each of the primary and secondary tanks as needed to set initial conditions as required.

In summary:

NOTWIN. . . . . .*disable* twinning, combining tanks if they are already split
GOTWIN. . . . . .*enable* later twinning, without immediately splitting tanks
SPLIT_TWIN . . . . . .*force* splitting to occur now, changing void fraction if needed

3.25.6.2 TLV (Vapor/gas Temperature) Input Option

If the first solution that will be invoked is a transient, and if the temperatures of the phases shouldn't be equal (e.g., the vapor/gas tank should be initialized as superheated and/or the liquid tank should be initialized as subcooled), or if the volatile species fraction should be less than 100% relative humidity in the vapor/gas tank, use the TLV input option.

For a pure substance, TLV should be greater than or equal to TL (e.g., superheated). By using the TLV option, TL can be less than saturation (i.e., subcooled liquid can be used as an initial condition). If noncondensible gases are present as well, then usually TLV is warmer than TL, but it might be cooler as well.

* Once in the separated mode, the void fraction is allowed to be within a larger range: about 0.1% to 99.9999%: hysteresis is used to prevent tanks from toggling back and forth between homogeneous and nonequilibrium modes.

The presence of TLV within a twin tank (or GENLU, LINE, or HX) subblock invokes several changes:

1. Each phase (each tank within the twin) will be initialized separately, permitting them to be out of thermal and mass transfer equilibrium initially.

   The gas or vapor phase will normally be warmer than saturation: say, TLV>TL or TLV>T_{sat}(PL). However, if noncondensible gases are present then the ullage can also be cooler than the liquid. The liquid might be saturated or subcooled, though it can be warmer if no volatile species are present.

   Even if TLV=TL, since the tanks are now initialized separately, their temperatures may not be the same (liquid can't be warmer than saturation nor vapor cooler than saturation). More importantly, for mixtures the vapor/gas (ullage) is now initialized without having to comply with the need for XG of the condensible species to represent a fully saturated state: the tanks can now be in mass transfer nonequilibrium, with the gas phase over or under saturated.

   These initial states can lead to severe initial transient (e.g., violent initial vaporization), and if inadvertently mixed before being solved, can result in property errors or extreme temperatures or pressures (e.g., a superheated noncondensible gas with a subsaturated condensible species, mixed with a volatile liquid). The intent of this option is to help avoid the need for user logic (CHGLMP, CHGVOL, SPLIT_TWIN etc. as noted above), but it still demands careful attention to the exact initial condition desired for a transient.

2. The homogenization that is part of BUILDF is suspended: if TLV was used as an input for *any* twinned tank, NSOL=2 will be assumed initially.

   This means *all* twinned tanks, whether or not they used TLV as an input, will be in the separated mode before the first solution is called. If TLV has been used for *any* twinned tank, the first solution is expected to be TRANSIENT (but may also be STDSTL or FORWRD), such that if STEADY is called first instead, a warning will be produced since severe states may result, as mentioned above.

There is no default for TLV (other than that specified in an LU DEF subblock): if it is not input, the tanks are initialized together in the combined mode using TL, PL or PL!, and XL or AL.

When TLV is detected as an input, the program will use the supplied values for TL, PL or PL!, and XG and XF, but it will apply them all to each tank separately. In other words, it will set XL=AL=0 to the primary tank, and XL=AL=1 to the secondary tank, initialize the thermodynamic states, then adjust the volumes as needed to comply with the input value of XL or AL. (When TLV is not specified, the tanks are initialized to the same two-phase thermodynamic state.)

The program is not always able to successfully initialize a nonequilibrium state. First, it requires that XL=0 for the primary tank and XL=1 for the secondary tank be achieved. Normally this requirement is easy to meet, but it precludes a supercritical state (which is not allowed for twinned tanks in any event).

Second, it requires that the initial state satisfy the minimum liquid and minimum vapor requirements. A twinned tank will homogenize during a run when the void fraction drops below 0.0001 (minimum vapor requirement) or when the void fraction rises about 0.999999 (minimum liquid requirement), so it will not allow a tank to be initialized outside of this range. If AL has been specified and it is outside of the range 0.0001 < AL < 0.999999, the tank will be initialized in the homogeneous mode without any error message. If AL is not supplied as an input, and if XL is not 0 or 1 yet the resulting AL is outside of the range, an error is produced. T**he use of AL is recommended for use with any twinned tank, but it should be considered almost mandatory when twinned tanks are initialized using TLV.**

Third, the program requires that the pressures in the tanks be initially equal. **This means that use of PL! (pressure priority) option *is almost mandatory* if a volatile/condensible species is present**, since it is otherwise very difficult to assure that the pressures will be initially equal. Similarly, TLV should normally be greater than T_{sat}(PL!) in such a case.

Examples:

```
LU DEF, AL=0.1, TLV = Norm_ullage

C with a condensible present:
LU TANK,1, AL=0.5,  VOL = V_High_Tank
      PL! =1.0e5,   TLV = Hot_ullage
      TL = 1.0e30       $ liquid side will be overridden with Tsat(PL!)
      XGH = 0.9, XGP = 0.1    $ these will be used in the vapor/gas
                              $ side, even if super or subsaturated.
      LTWIN = 1001,  AST=width*length


C if mixture is nonvolatile, noncondensible (e.g., oil and air)
LU TANK, 33, VOL = 1.0, AL=0.99
      LTWIN = 1033, AST = 0.1
      PL = P_atmos  $ PL! not needed since no phase change involved
      TL = 300.0    $ hot oil
      TLV = 200.0   $ cool air
```

3.25.7 Anticipation of Phase Change

Assume a twinned tank is in the combined mode full of vapor and/or gas, and a little cold liquid is injected. If FLUINT waited until a liquid phase had established itself via perfect mixing rules, then the pressure would plummet unrealistically, negating most of the benefits of nonequilibrium modeling: the key event would have passed before nonequilibrium methods were even invoked. A similar problem exists when vapor is injected into a liquid-filled tank.

To help overcome this difficulty, FLUINT anticipates when such a tank will transition due to injection.* (Phase change by heating or cooling does not cause a similar problem since it heats or cools the bulk fluid before the transition occurs.) When it estimates that such a transition will occur during the next time step, it splits the tanks into separate twins in expectation of the event, such that nonequilibrium methods are applied from the start.

However, this anticipation temporarily violates a strict conservation of mass and energy, since a bubble or a droplet of minimum size must be artificially created in anticipation. This slight error in conservation is accumulated and overcome in the course of the next few time steps, as controlled by the control constants RMFRAC and RMRATE (Section 4.4.2).

The user can also control the time step size over which this “anticipation logic” is applied using the control constant RMSPLT (Section 4.4.2). Such control is important since preserving tiny droplets and especially tiny bubbles is very difficult numerically. In the latter case, the thermodynamics of a tiny bubble dominate the response of a much more massive liquid-filled tank and therefore perhaps an entire network’s response.

If a transient simulation of a filling a vessel starts with a dry initial condition, or if the analysis of a draining tank starts with a completely full initial condition, the user is strongly encouraged to adjust the initial conditions to start with a small amount of the opposite phase. In other words, start a filling analysis with a small amount of liquid already present (AL<0.9999), and start a draining analysis with a bubble already present (AL>0.01). Taking this care in initial conditions avoids a very time-consuming and usually inconsequential simulation of the formation of first droplet or bubble. Refer to the prior subsection for information on specifying initial conditions for twinned tanks.

* See Section 3.25.8.1 for limitations when modeling quenching in stratified lines.

3.25.8 Example: Duct Macro

When a duct macro is used for dissolution (Section 3.23) and/or nonequilibrium twinned tank models of pipe flow, the seemingly complex input options disappear via the default system, as do most needs for special initializations. Paths attribute their surface areas and heat and mass transfer coefficients to the appropriate lump automatically. As long as the user does not desire to name the superpaths, FLAT ifaces, and CONSTQ fties (leaving them to default to the lump ID*), then all the user need add to the LINE or HX macro is the identifier of the first twinned tank. For example:

```
LTWIN = 11
```

This simplicity is contrasted with individually input pairs of twinned tanks, which are normally used in stagnant or quasi-stagnant control volumes (Section 3.25.9). However, recall that only FLAT ifaces are applicable between paired tanks in duct macros (because of the implicit use of twinned paths), whereas other types of IFACEs may be used in nonduct situations, permitting pressure differences between the liquid and vapor tanks.

Use of centered discretization ("C") options is strongly recommended. The one exception to this statement is flat front modeling of the filling or purging of lines (Section 3.27), which works well with twinned tanks.

3.25.8.1 Cautions Regarding Modeling Horizontal, Stratified Lines

When volatile liquid in a horizontal (or nearly horizontal) line stratifies, cool liquid will collect on the bottom and warm vapor/gas will collect near the top. This section describes issues† that only arise with stratified flow regimes; other regimes completely wet the pipe walls.

Unfortunately, a single tank (or twinned tank pair) does not have a knowledge of which nodes are above or below the liquid/vapor level. The user could conceivably employ the ITAC=1 option for twinned ties and add such information themselves, but this would be onerous. If the flow is somewhat stagnant, and if the user could create an expression defining the liquid depth, FloCAD's node-locating feature for HTP ties could also serve as a work-around.

Otherwise, if no circumferential thermal resolution is used ... if a single node represents the periphery, then that node's temperature will represent an "average" of what should be a warmer temperature on top and a cooler temperature on bottom. In this situation, the liquid will be exposed to a warmer environment than actual, and vapor will be adjacent to a cooler wall than it should.

Modeling the quenching of a warm wall by an inflowing liquid can be especially problematic, since not only are the above issues of either losing or tracking circumferential gradients present, but additional issues arise as well.

---

* Many users apply the same sequence of IDs to both the lumps and the main (axial) tubes or STUBEs. This usage unfortunately conflicts with the default of applying the lump ID as the superpath ID. Either rename the main tubes or STUBEs, or use the IDPATH option to rename the superpaths to avoid naming conflicts.

† Future versions are expected to offer more advanced tools for this class of application.

First, the progression of liquid from one end of a segment to another creates a problem parallel to that of the circumferential heat transfer issue. If the *axial* resolution of the nodes is the same as that for the fluid volumes (i.e., one node per lump), then axial temperature gradients that should exist will be lost: liquid entering such a segment will experience a warmer wall than it should, and it will similarly experience a cooler wall than it should as it leaves that segment. On the other hand, if the axial resolution of nodes exceeds that of the fluid control volumes, then such a tank is also unable to discern which nodes that its liquid encounters first unless the user again takes over with ITAC=1 and knowledge of the orientation and location of each node. As with the circumferential resolution issue, the user has the choice of either neglecting the gradients or working to distinguish wetted nodes from dry nodes, but both choices are problematic.

Second, when twinned tanks are used to model the flow, before the liquid can enter the control volume downstream, it first causes a collapse of the temperature and pressure downstream as if it were sprayed into that segment. The anticipation method introduced in Section 3.25.7 is often no help in this instance, because if twinned tanks *were* formed downstream to prevent the collapse, the first tiny tank of liquid that appears would experience a too-warm wall both axially and circumferentially, so it quickly disappears. Choosing to set RMSPLT=0.0 to avoid this transition struggle, and accepting the false periodic response as each tank quenches may be required.

The only current resolution is increased axial discretization of the line to diminish these effects, though at the expense of increased run times. Increased axial resolution does not address the issue with either neglecting or including circumferential gradients, but it might make them easier to handle in short segments that use a single ring of nodes at each axial segment.

3.25.9 Using Stand-alone (nonmacro) Twinned Pairs

Nonequilibrium two-phase control volumes may be required in stagnant or quasi-stagnant vessels: portions of the flow model in which mixing is comparatively poor and/or temperature differences between liquid and vapor can be expected either due to wall heat transfer or perhaps compression of the vapor space. (Such situations also often demand the most care in setting initial conditions, as discussed in Section 3.25.6. These types of models often make use of HTP ties, which are documented in Section 3.6.3.3. See especially the DEPTH parameter of those ties. Path end location (Section 3.13.4) are also relevant.

See TankCalc60 in Section 7.11.9.10 for extra assistance in common tank geometries. Also note that stand-alone twinned tanks are a main tool of FloCAD Compartments, as described in the Thermal Desktop User's Manual. When used with Compartments, the AST and coordinate locations (CX, CY, CZ) of the twinned tank are updated automatically.

Unlike twinned tanks within macros, the code cannot assume anything about the type of flow, the shape or size of the interface, wall heat transfer apportionment etc. Therefore, the user assumes responsibility for describing the situation, although a partial default system is still available to help estimate parameters or at least to provide a basis for either sensitivity studies or correlation. The main purpose of this section is to provide the user with guidelines regarding defining the heat and mass transfer parameters within stand-alone tanks.

The user can still have the twinned tanks acquire characteristics (surface area, heat transfer coefficients) from adjacent paths. Single paths attached to one of the twinned pair add their contributions to the isothermal interface (Section 3.23) within the attached tank (to be added with ASL, affecting perhaps UVI, ULI, GLx). Twinned paths add their contributions to the nonisothermal interface (Section 3.25.2) between the twinned tanks (to be added with AST, affecting perhaps UVT, ULT, GTx).

All of these manipulations are available within macro paths as well, but those are automatically calculated by the default options. Only one key difference, in fact, exists between stand-alone twinned pairs and those generated as part of a duct macro: if no twinned paths are applied to a stand-alone pair, the user can apply an iface type other than FLAT, permitting the pressure of the liquid phase to differ from the pressure of the vapor/gas phase. This difference is sometimes caused by gravity gradients, but also because of interface curvature (as modeled by SPHERE, SPRING, or WICK ifaces).

3.25.9.1 Default Values and Modeling Guidance

Perhaps the biggest obstacle facing the user is the choice of the values AST, ULT, and UVT. (By default, GTx values will be estimated from those three inputs.) These values are dependent on the geometry of the control volume, its orientation relative to body forces, the movement of the fluid in each phase relative to the other phase (which is influenced by ingress and egress if any). These values can rarely be determined analytically: the user must rely on parametric analysis, judgment, and calibration with test data if available. The following discussion is intended to help establish good engineering judgment, and to explain the usage of the default options.

To use the default calculations, simply input -1.0. To use twice the default value: -2.0, half the default value: -0.5, etc. Otherwise, nonnegative values will be used directly. The intent is to allow the user to parameterize the problem in terms of ratios of the default values if no other information is available.

The default value for AST is zero: no heat or mass transfer will occur unless some twinned paths contribute to the main interface via their ASPU/ASPD designations. A positive value of AST is used directly as CAST (the calculated AST value available in logic or expressions). A negative value of AST denotes a fraction of a spherical interface based on the current secondary (vapor/gas) tank volume. In other words, AST=-1.0 means use one sphere's surface area, and AST=-1.5 means add fifty percent more (i.e., an oblong bubble or two disconnected bubbles, etc.).

The parameter with the least uncertainty is AST, the interfacial surface area. As noted above, the default is based on a spherical vapor or gas bubble, which is valid in the limit of small void fractions for *any* shaped control volume as long as all the vapor is assumed to exist in one bubble. Otherwise, the value of AST may be estimated based on the shape of the control volume, any imposed body forces, and the fill level. For example, assume a cylindrical control volume of length L and diameter D. For a vertical cylinder in a gravity field, the surface area is a disk of diameter D. In the absence of gravity or if D is very small, the surface area is a hemisphere of diameter D (up to a certain fill level). In tilted cylinder, the surface area is either an elliptical disk (in gravity) or an

ellipsoid (low gravity or small D). The user should not be overly concerned with calculating an exact surface area because this term is always used in combination with ULT and UVT, and the uncertainty in those parameters is relatively large, as will be discussed below.

The defaults for ULT and UVT are -1.0. A negative value for these parameters invokes a calculation based on the input or calculated value of AST assuming solid conduction: KA/L, where K is the conductivity of the appropriate phase, A is AST (whether input or defaulted), and L is a characteristic conduction path length based on the current volume of the appropriate phase and AST: L = VOL/(2*AST). The validity of this approximation varies with control volume and surface area shape, but it provides a reasonable estimate of the *lowest possible heat transfer coefficient* in all cases.* It is roughly equivalent to a Nusselt number of unity. Surprisingly, *this lowest estimate of solid conduction is also often the best estimate*. The reasons for this statement are as follows.

First, there is usually little boundary layer shear at the interface: the vapor is often just dragged along with the liquid surface motion, if any. In one experimental study, a cup containing vapor was inverted in a spinning pool of subcooled liquid. The heat transfer rate in the vapor was very low despite the relative motion of the liquid; apparently solid body rotation formed quickly in the vapor. The moral is that, unlike heat transfer between a fluid and a wall, the velocity profile at a liquid to vapor interface does not contain much of the shear layer that normally increases heat transfer beyond pure conduction.

Second, stratification or poor mixing in the liquid volume is common if there is only one inlet or outlet to that volume (as in the case of a liquid reservoir or accumulator, or a fuel tank), if the control volume is vertical and has a long aspect ratio (e.g., a vertical or slightly tilted cylinder), or if some sort of baffle or disrupter exists at the inlet to disperse incoming fluid and prevent jets from forming. Even without the above conditions, a solid conduction value may be warranted if the liquid is not disturbed and if the flow *exits* from the liquid control volume: such conditions do not promote mixing of the liquid. Conditions that *do* promote mixing and hence warrant a higher ULT include ingress of undisrupted liquid into a disturbed or short aspect ratio control volume, and cases where liquid flows through the control volume. In the former case the formation of liquid jets is promoted: incoming cold liquid is able to reach the interface easily in the form of a jet, and the resulting rapid condensation (for cases with phase change) at the interface drops the control volume pressure, increasing the jet flow rate.

Third, in multiple-constituent flows, phase change and hence interface mass transfer is zero if no condensible species is present, or may be very low if the concentration of the condensible species is small. This means even less movement of fluid, and hence lower mixing rates within each phase. For these reasons, the adequacy of the thermal equilibrium (perfect mixing) assumption becomes less likely for multiple-constituent two-phase flows than single constituent flows, and the use of the twinned tanks should be more frequent with mixtures.

* While this statement is true, the lowest *effective* coefficient can be *less* than unity when condensation or vaporization is diffusion-limited. When a noncondensible gas is present, mass transfer is corrected (“degraded”) to include diffusion-limited condensation and vaporization. Internally, the HTFDIF utility (page 7-297) is applied.

In summary, with no vapor ingress or degradation due to diffusion, the default conduction value should be used for UVT, and only minimal increases should be considered even if there is flow into or through the vapor portion. For the liquid coefficient ULT, the choices are less clear, although a solid conduction value is appropriate for most cases in which mixing in the liquid is not promoted. As a rule of thumb, relying on the facts that AST and UVT can be reasonably estimated, the user should concentrate on ULT for calibration against test and for parameterizing system response, preferably as a ratio of the solid conduction default (see the example below). Remember that this ULT ratio may change during execution as conditions (e.g., the inflow rate) change.

Noting that the pressure in the control volume is governed by the saturation condition of the vapor volume for single-constituent cases, and by the gas temperature for multiple-constituent cases, *the user should not neglect or oversimplify heat transfer to the container wall nor neglect the mass of the wall in transient analyses where the pressure response of the control volume is critical*. These wall terms may include natural convection and film condensation in the vapor space, and may include subcooled and nucleate or pool boiling in the liquid space. Otherwise, solid conduction or single-phase convection terms should be considered as a minimum for wall heat transfer.

3.25.9.2 Two-phase (Separated Mode) Behavior

To illustrate typical behavior of a nonequilibrium two-phase single-constituent control volume, consider a cylindrical, vertical, and adiabatic reservoir that is plumbed to the rest of the system via a single port at the (liquid covered) bottom. This scenario is used as the basis of the sample presented in Section 3.25.10.

Now consider cold (subcooled) liquid flowing into the reservoir. If the influx is more rapid than the condensation rate at the interface, the vapor will compress and heat up, driving the saturation pressure up temporarily and tending to discourage the influx.* Eventually the vapor will cool down as the influx stops, and will begin to condense (not just at the interface), lowering the pressure and thus permitting more influx. If the influx were more gradual, the control volume may never experience an increase in pressure. Note that equilibrium conditions dictate that the pressure will eventually be less than before unless the wall heater can keep up with the rate of cold liquid influx, but that the *process* in which this liquid is injected determines the *route* by which it approaches this equilibrium value.

In the reverse process, if liquid leaves gradually (less than the evaporation rate at the interface), then the pressure will only drop slightly if at all. However, if the liquid is quickly drawn out of the reservoir, the pressure will drop quickly as the vapor is expanded, perhaps to the point where the bulk of the liquid begins to boil or flash. At that point, further pressure reductions of any significance do not occur: the vapor volume grows from the coalescence of bubbles from the liquid side.

In the first instance, the rate of condensation at the interface is *almost strictly a function of the heat transfer and mixing in the liquid side: ULT.* Thus, the distinction between “rapid” vs. “slow” influx is dependent on the value of ULT. This fact, combined with the facts that the heat transfer in a vapor bubble is almost entirely due to conduction and that the pressure in the control volume is a function of the vapor side, causes analytic difficulties in the limits of a nearly hard-filled reservoir.

* Since the condensation rate is low or zero for multiple-constituent models, compression heating of the gas void almost always takes place.

The vapor bubble is persistent. Increasing the liquid influx slightly causes the vapor to superheat quickly, and yet the state of this small bubble dominates the pressure of the larger liquid volume. At some point in the analysis, the nonequilibrium methods must be abandoned, reverting to single-phase (combined) methods.

3.25.10 Example: Stand-alone (vs. Duct Macro) Tanks

The nonequilibrium behavior of a cylindrical, vertical, adiabatic, reservoir (Section 3.25.9.2) filled with pure, volatile (two-phase), ammonia is considered under simplified boundary conditions. The reservoir is assumed to be a vertical cylinder 5 cm. in diameter and 50 cm long, and is being used in an ammonia system at 20°C. It is initially $^1/_4$ full of liquid by volume. At time zero, fluid at 0°C is injected at a constant mass flow rate such that the reservoir becomes $^3/_4$ full in 60 seconds. The reservoir is then held for two minutes more, at which time liquid is extracted at the same rate as before for one minute, returning the fill level to $^1/_4$, where it is held for the final two minutes of the simulation. UVT is held at -1.0 (solid conduction in the vapor) and AST corresponds to a disk-shaped interface. For comparison purposes, ULT will be parameterized from -1.0 (the minimum) to -1000.0 in order-of-magnitude increments. An analysis of a single perfectly mixed tank (corresponding to infinite ULT and UVT) is also run.

Figure 3-45 shows the complete thermal response of the cylinder in the ULT=-1.0 case. Figure 3-46 shows the overall pressure response of all cases. The complete input file is included at the end of this section. In the ULT=-1.0 case, the heat transfer between the phases is so poor that little change is evident when the reservoir is locked up. Clearly, the high temperatures ranges experienced in this case would increase the importance of the wall model in a real analysis. As the heat transfer rate (or degree of mixing in the liquid) increases, the peak pressure is reduced, and the reservoir approaches equilibrium faster.

The extremes of behavior in this example were purposeful. A long aspect ratio cylinder was chosen along with minimal heat transfer area, an adiabatic wall, and extremely fast fill and empty rates to exaggerate the nonequilibrium behavior. The user should not assume such extremes are typical. In normal pipe flow, the perfect-mixing assumption is usually acceptable.

Sample Problem F contains another example of nonequilibrium simulation.

The complete input listing for this example is listed in Table 3-18.

Table 3-18 Sample of Stand-alone Twinned Tanks

```
HEADER OPTIONS DATA
TITLE NONEQUILIBRIUM RESERVOIR - SECTION 3 EXAMPLE
       OUTPUT         = non.out
       USER1          = non.usr
C
HEADER REGISTER DATA
       diam           = 0.10
       length         = 0.10
       uliq           = -1.0                        $ INITIAL LIQ SIDE HTC (ULT)
       filrat         = 0.0                         $ FILL RATE (AND EMPTY RATE)
       voltot         = 0.25*pi*diam^2*length
```

```
C
HEADER FLOW DATA,RESERV,FID=717
LU DEF, TL=20.0
PA DEF, FR=0.0, DH     = 0.10,        TLEN           = 0.10
C
C LIQUID SIDE (AND WHOLE TANK DURING PERFECT MIXING RUN)
C INITIALIZE TO SATURATION WITH FAKE XL, CHANGE VOLUMES LATER
C
LU TANK,1, PL=0.0, VOL=voltot
       LTWIN           = 2                          $ make it twinned
       AST             = 0.25*pi*diam^2
       ULT             = uliq
       AL              = 0.75                       $ 1/4th full of liquid
C
LU PLEN,300,XL=0.0, PL=0.0, TL = 0.0                $ 0 C LIQUID SUPPLY
PA CONN,300,300,1,STAT = DLS,                       $ INJECTOR/EXTRACTOR
       DEV             = MFRSET
HEADER CONTROL DATA, GLOBAL
       UID             = SI                         $ metric
       ABSZRO          = -273.15                    $ deg C
       OUTPUT          = 1.0E10
       OUTPTF          = 6.0                        $ OUTPUT 6 SEC INTERVAL
       NLOOPS          = 50
       DTMAXF          = 2.0
C
HEADER OUTPUT CALLS, RESERV
       CALL LMPTAB
       CALL LMXTAB
       CALL PTHTAB
       WRITE(NUSER1,11)TIMEN,TL1,TL2,VTS(PL1-PATMOS,RESERV.FI)+ABSZRO
     .                ,XL1,XL2,PL1/1.0D5
11     FORMAT(1X,1P,6(G14.5,',',),G14.5)
HEADER OPERATIONS
BUILDF NONEQ,RESERV
DEFMOD RESERV
C
       CALL SVPART('-R',NTEST)                      $ save all but registers
C
C TRIPLE CURRENT LIQUID AMOUNT OVER 1 MIN
       FILRAT          = 2.0*DL1*VOL1/60.0
C
       DO 1 ITEST      = 1,5
C
C PULL BACK INITIAL CONDITIONS AND WRITE HEADER TO USER FILE
                       CALL RESPAR(NTEST)
                       WRITE(NUSER1,10)ULIQ
C
C IF SPECIAL FIFTH RUN (PERFECT MIXING) RESET TANK AND UNTWIN
                       IF(ITEST .EQ. 5)THEN
                                  CALL NOTWIN('RESERV',1)
                       ENDIF
C      FILL
                       TIMEO           = 0.0
                       TIMEND          = 60.0
                       SMFR300         = FILRAT
                       CALL TRANSIENT
C      HOLD
                       SMFR300         = 0.0
                       TIMEND          = TIMEND + 120.0
                       CALL TRANSIENT
C      EMPTY
                       SMFR300         = -FILRAT
```

```
                      TIMEND          = TIMEND + 60.0
                      CALL TRANSIENT
C       HOLD
                      SMFR300         = 0.0
                      TIMEND          = TIMEND + 120.0
                      CALL TRANSIENT
                      ULIQ            = ULIQ*10.0
1       CONTINUE
C
10      FORMAT('1 RESULTS WHEN ULIQ = ',1pG13.5/
     .  /5X,'TIMEN',T25,'TL',T40,'TV',T55,'TS',T70,'XL',T85
     .  ,'XV',T100,'PL'/)
C
END OF DATA
```

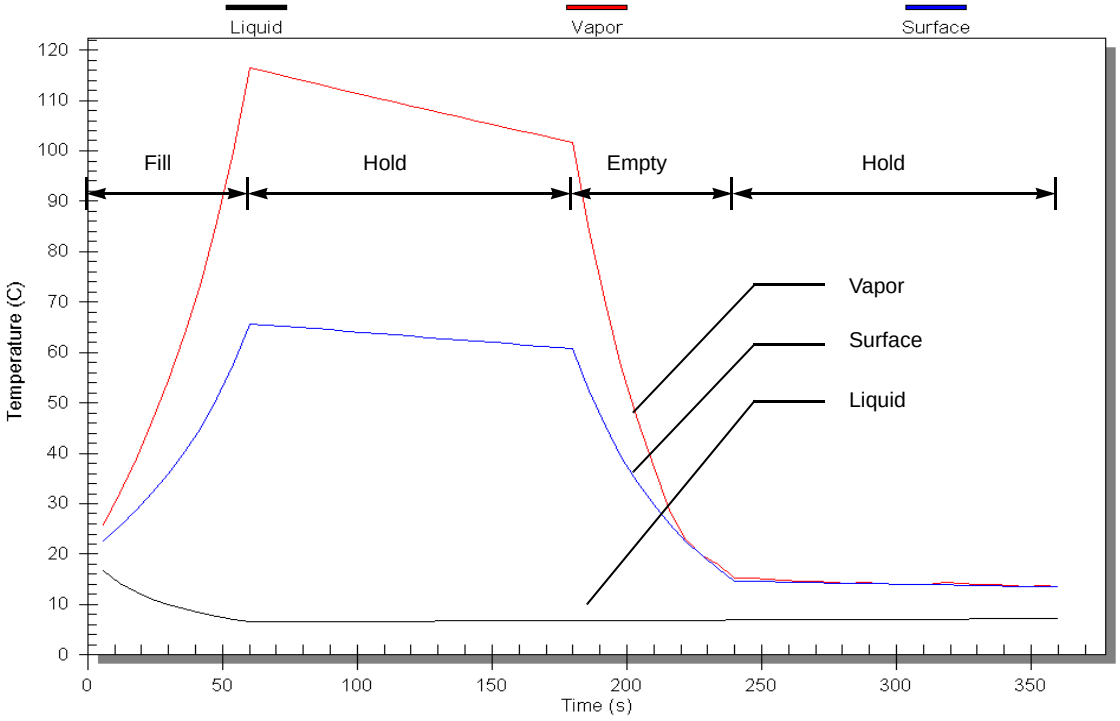


Figure 3-45 Temperature Response Given ULT = -1.0

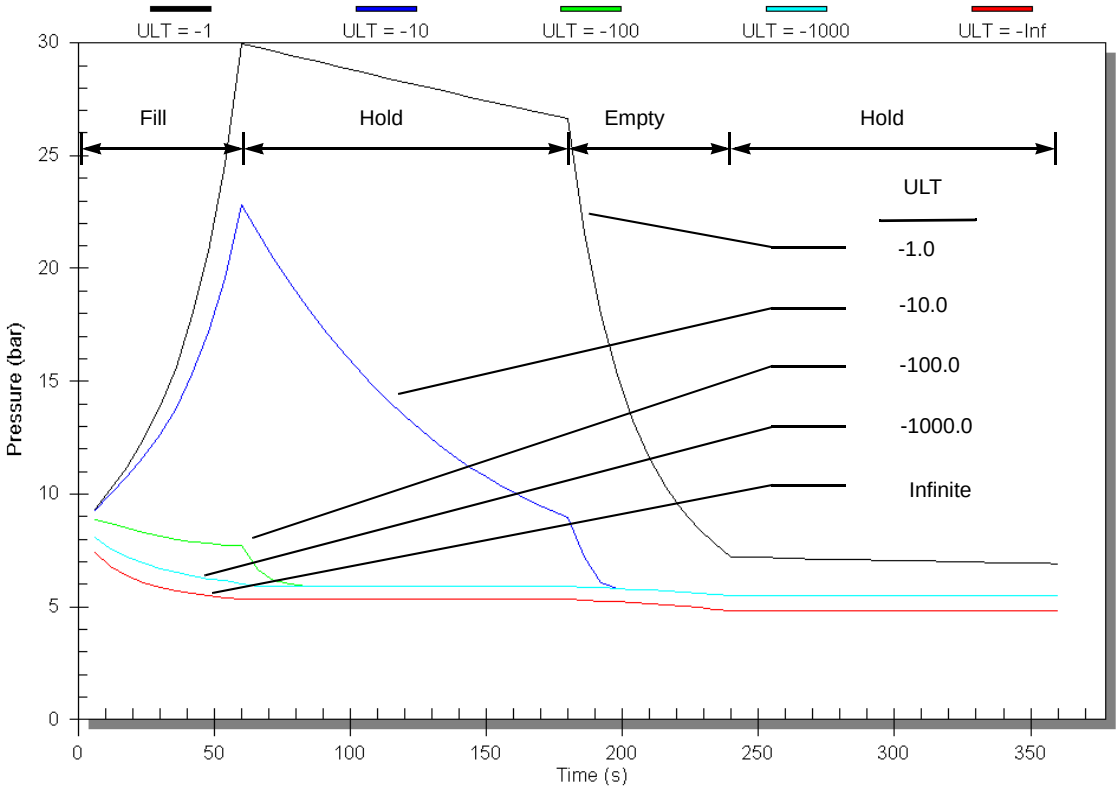


Figure 3-46 Pressure Response as a Function of ULT

3.26 Rotating Flow Passages

Examples of rotating fluid passages include rotary lawn sprinklers, coolant channels within turbine blades, and oil channels within piston rods. Additional examples in which the walls have relative motion (i.e., do not co-rotate) include secondary flows within seals and bearings: in gaps between rotors and stators. Generally, rotating passages occur within turbomachinery, engine, and transmission systems. As usual, SINDA/FLUINT makes little presumption about the intended application, and provides general-purpose modeling tools instead.

Paths are by default stationary. They can optionally be assigned a rotational (spin) rate along with a beginning and an end radius (distance from the spin axis). “Beginning” refers to the defined upstream (i^{th}) end of the path, and “end” refers to the defined downstream (j^{th}) end; these ends do not switch when flow rates reverse. The spin rate is given by the path parameter ROTR, which uniquely has units of revolutions *per minute* (rpm) *independent of the model units* (SI vs. ENG). The end point radii are specified as RADI and RADJ, in user units of length (ft or m, depending on UID). All three of these variables default to zero.

The user will normally need to further specify the beginning and ending angles of the path with respect to the rotational axis and the plane containing the flow passage. These velocity angles (VAI, VAJ, VXI, and VXJ, in degrees) are defined in Section 3.26.1. The default for these angles corresponds to a perfectly radial tube: VAI=VAJ=VXI=VXJ=90 degrees.

The main effect of specifying rotation is to impose an additional pressure force upon the path. If the inlet of the spinning path is stagnant (LSTAT=STAG), and if the walls are co-rotating (e.g., a spinning tube or a passage within a solid), then an additional accelerational pressure loss may be applied to yield static pressures downstream of that entrance path.* For high-speed flows, including high rotational rates, changes to the rotation of the fluid (especially at the entrance and exit of a rotating section) will be reflected in the tally of kinetic energies into the end-point lumps. Another side effect is to impose a rough estimation acceleration perpendicular to the path, which is relevant only for two-phase flow regime determination (Section 3.16.3). *Normally, no other enhancement of frictional pressure drop or heat transfer is made: if such effects are required they should be added separately* (see for example the path FCLM and FCTM multipliers, and the analogous heat transfer tie multipliers).

The walls of the fluid passage need not co-rotate: there can be relative or shearing motion. This is indicated by the rotational velocity ratio RVR, the use of which is described in detail in Section 3.26.2. RVR=1.0 by default, which means co-rotating walls: the fluid is rotating at the same rate as the walls. RVR<1 means that one wall is not rotating as fast as the other. RVR=0.5, for example, might be used to model a passage between concentric cylinders or a gap between a rotating risk and stationary disk: it implies that the average fluid velocity is half of that of the rotating wall. For most

* For co-rotating walls or solid rotational motion (RVR=1.0), this additional acceleration is above and beyond the “equivalent K-factor of one” as described in Section 3.3.1, Section 3.3.3, and Section 3.15.2.5. The fluid must be accelerated to the total velocity of the rotating path: the velocity of the passage (based on its flow rate) added vectorially with the rotational velocity (based on its ROTR and radius, etc.).

connector devices, this simply results in a decrease of the rotational force. For tubes and STUBEs, however, the election of RVR<1 has a profound effect on heat transfer and pressure drop, as described in Section 3.26.4.

When walls are slipping relative to each other, entrance accelerations are not augmented nor are kinetic energies of the absolute velocity included. Rather, the flow equations are solved “along the FR direction” and not along a streamline. This important distinction is clarified later. For now, note that there is a significant difference between RVR=1.0 (co-rotating walls) and any RVR even slightly less than 1.0 (indicating relative wall motion).

Whether walls are co-rotating or moving independently, energy is being added or subtracted by mechanical motion. By default (INTORQ=NO), this energy (and the corresponding hydraulic torque, TORQ) is calculated in a first-order manner: by assuming energy is added like a pump or subtracted like a hydraulic turbine using an “efficiency” (EFFP, which defaults to unity). For flow within gaps and seals (usually RVR<1), a torque should instead be input by the user (INTORQ=YES). In either case, the resulting flow work is calculated as QTMK, which is then summed into the outlet lump’s QTM value (which is in turn summed into the lump’s total heat load, QDOT). Section 3.26.3 describes the torque and energy options in more detail.

Note that the specification of rotational velocities and radii is a function of paths and not lumps: the endpoint lumps might be stationary or might themselves be rotating (such a distinction is not made by the software, as long as path forces are appropriate). Normally, the effects of rotation will overwhelm body force or elevation changes, but if not the user should consider assigning an average position (CX, CY, CZ coordinate location) to a rotating lump.

In FLOW DATA, the additional input formats in any path subblock are:

```
[ROTR=R][,RADI=R][,RADJ=R][,VAI=R][,VAJ=R]
[,VXI=R][,VXJ=R][,RVR=R]
[,EFFP=R][,TCF=R][,TORQ=R,INTORQ=C]
```

These parameters, their meaning, and their implications on the simulation will be described in subsequent subsections.

For users of text input files, note that in HX or LINE (duct macros, Section 3.9.2), special rules apply as they do for DUPI and DUPJ: *these values apply to the whole duct and not to individual paths within that duct.* RADI and VAI are only applied to the inlet path, while RADJ and VAJ are only applied to the outlet path.* Values for internal paths are interpolated between the inlet and outlet values. For this reason, if one end is specified, the other must be too. For example, either *both* RADI and RADJ must be specified within the HX or LINE block, or *neither* should be. The same rule applies to VAI and VAJ.

The above default usage is intended for a spinning tube or passage with co-rotating walls (RVR=1.0). For passages with relative wall motion that must be subdivided axially or radially using a duct macro, then the RADI/RADJ and perhaps VAI and VAJ or each path within the macro must

* Center-discretization is the preferred method for this and many other reasons. Upstream- and downstream-discretization are accepted but should be avoided.

be specified separately and perhaps individually. For users of Sinaps or FloCAD *pipes*, rotation options are not available at the top level forms and must be specified as part of the tube or STUBE definition, usually by editing all pipe paths at once (independent of whether RVR=1.0 or RVR<1.0).

Finally, note that a special output routine, ROTTAB, is available to tabulate data relating to path rotation, including the effective pressure force due to the effects of spinning.

3.26.1 Velocity Components

The components of flow velocity at both the inlet and outlet of each path must be defined relative to the axis of rotation. Two angles are employed at the inlet (VAI, VXI) and two at the outlet (VAJ, VXJ), and all have units of degrees and default to 90 degrees. The VAI and VAJ angles have the greatest effect: they are used within the calculation of the body force in parallel with the flow. The VXI and VXJ angles, on the other hand, have no direct effect on the forces *along* the path; they affect only *perpendicular* forces that can in turn affect flow regimes.

The VAI and VAJ angles are defined in the plane containing both the rotational tangent vectors and the path flow vectors, as shown in Figure 3-47. VAI is the angle from the flow path at the inlet to the tangent vector (the instantaneous velocity of a particle in the moving wall). VAJ is the same angle at the defined outlet. The defaults of VAI=VAJ=90, therefore, correspond to a flow passage that is perpendicular to the tangent vector, but this does not imply either radial or axial flow.

In fact, the angles VAI and VAJ do not contain information about how the flow-tangent plane is located relative to the axis of rotation. That information is the purpose of the VXI and VXJ angles: they describe the angles between the inlet plane and the axis of rotation, and between the outlet plane and the same axis, as illustrated in Figure 3-48. Specifically, the VXI and VXJ axial angles are defined as the angles between the planes containing VAI and VAJ and the axis of rotation.

If the flow is purely radial, then Figure 3-47 applies entirely (and the axial angles VXI and VXJ are both left at the default value of 90 degrees from the axis of rotation).

For a path representing the primary flow through a centrifugal pump (and for some reason a PUMP connector was not chosen instead), then VXI=0 and VXJ=90. For flow through a radial gap, such as a flow-through journal bearing as depicted in Figure 3-49, VXI=VXJ=0.

Note that in Figure 3-49, the planes containing the angles VAI and VAJ have been rotated 90 degrees from the configuration in Figure 3-47: from purely radial to purely axial. However, the meanings of VAI and VAJ have not changed: they still represent the angles from the tangent vectors to the path flow rate vectors. In the case depicted in Figure 3-49, no net pumping would occur in an incompressible fluid since VAI=VAJ=90 and RADI=RADJ, but in reality VAJ is slightly less than 90 degrees due to residual swirl at the exit of the cylindrical gap, which would cause slight pumping. VXI and VXJ do not contribute to pumping (acceleration along the path). Rather, they enable the code to calculate the acceleration perpendicular to the path, which is currently applied only to two-phase flow regime determination estimations.

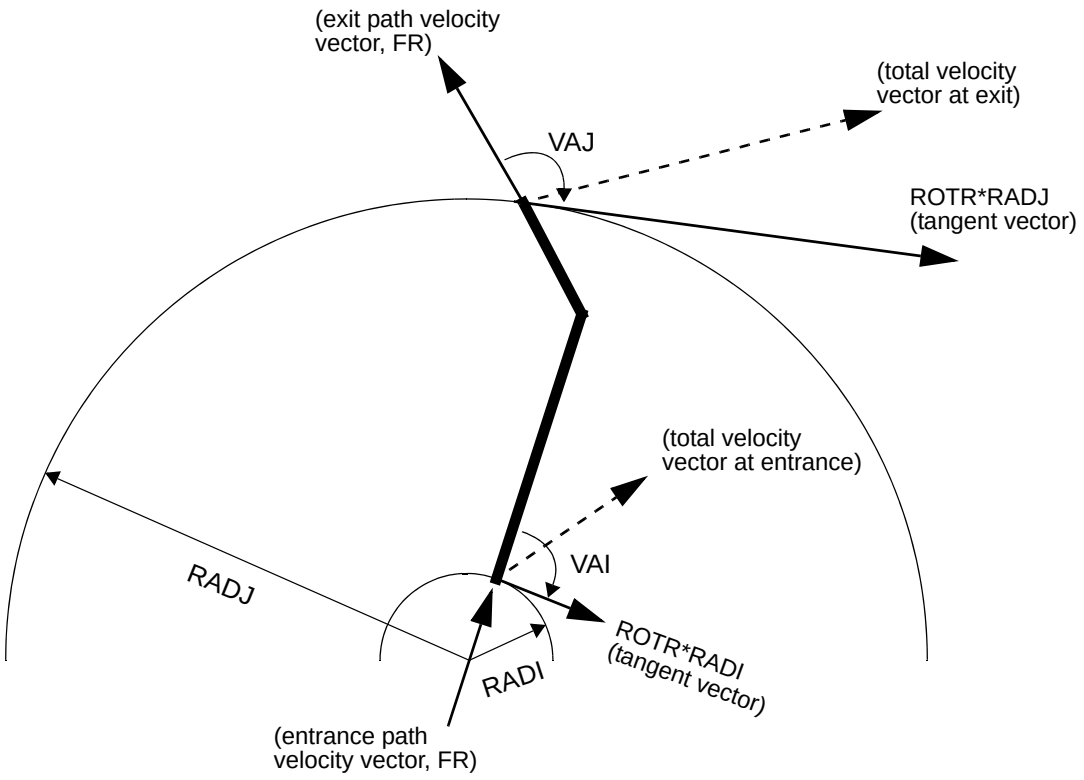


Figure 3-47 Velocity Angles within the Flow-Tangent Plane (and Radial Flow Example)

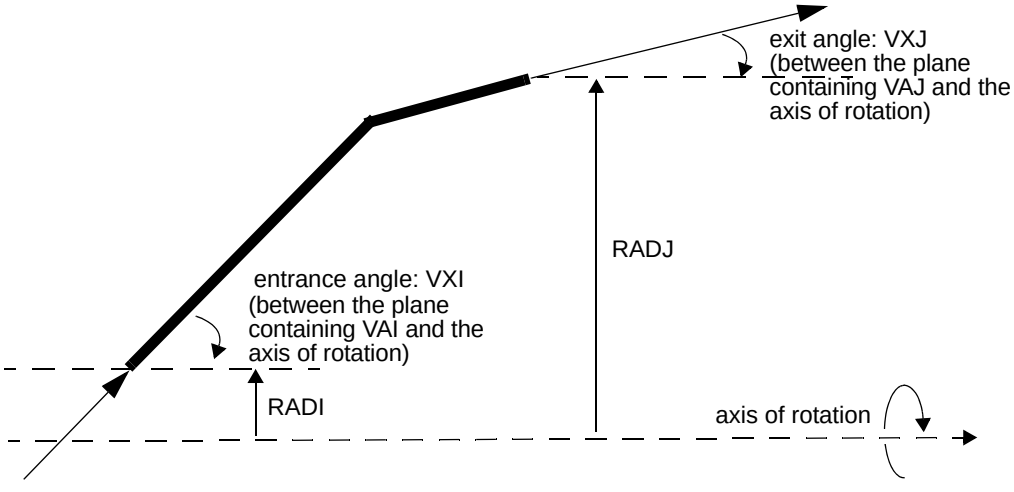


Figure 3-48 Velocity Angles for the Axial Component

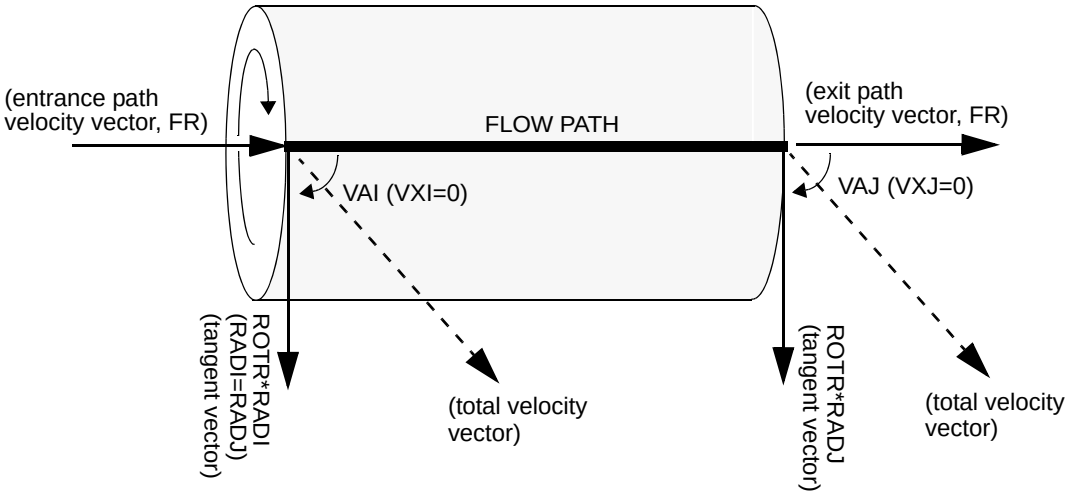


Figure 3-49 Velocity Angle Example: Flow Through a Journal Bearing (Top/Internal View)

Note that the total velocity vector (shown dashed in the figures) is a vector sum, and can be larger or smaller than the path velocity based on FR. It is changes to this total vector from inlet to outlet that determine the corresponding pressure forces on the path. A query routine, VELPATH (Section 7.11.1.10) is available for calculating the various velocity components including the total vector.

In summary, the path variables defining angles are as follows:

`VAI` ..... Angle (in degrees) at the defined inlet from the relative path vector to the tangent vector (ROTR*RADI).

`VAJ` ..... Angle (in degrees) at the defined outlet from the relative path vector to the tangent vector (ROTR*RADJ).

`VXI` ..... Angle (in degrees) at the defined inlet from the plane containing VAI to the axis of rotation. VXI defaults to 90 (radial flow), whereas VXI=0 represents axial flow.

`VXJ` ..... Angle (in degrees) at the defined outlet from the plane containing VAJ to the axis of rotation. VXJ defaults to 90 (radial flow), whereas VXJ=0 represents axial flow.

3.26.2 Rotational Velocity Ratio (Relative Wall Motion)

In Figure 3-49, the inner cylinder (perhaps a shaft or rotor) is rotating but the outer cylinder (not shown) is not. In other words, the assumption that the walls rotate with the fluid (solid body rotation) has been violated: the walls are instead moving relative to each other. The fluid within the gap is

sheared, with the effective velocity somewhere between that of each wall. In fact, lacking better data, the average of the two wall velocities is often a good estimate. If one wall is smooth and the other is very rough (or has vanes or ribs or knurls), then the average velocity would be skewed toward the rough or vaned wall.

This effect can be included via the path parameter RVR:

RVR . . . . . The velocity ratio* for relative wall motion. RVR=1.0 by default, meaning solid body rotation (e.g., a rotary lawn sprinkler impeller, turbine blade coolant passages, engine oil within hollow cam shafts). RVR=0.5 is a good estimate for most cases with similar walls if one wall is stationary and the other is rotating at ROTR (e.g., a journal bearing or face seal). If the rotating wall is rougher or has vanes, a value higher than 0.5 may be appropriate. If instead the rotating wall is smooth and the stationary wall is not, then an RVR less than 0.5 may be appropriate.†

RVR effectively reduces the fluid rotation: $ROTR_{eff}$ = RVR*ROTR. Since the pressure force associated with spinning is proportional to ROTR squared, a value of RVR=0.5 reduces the spinning force to one fourth of that of solid body rotation: $\Delta P_s \sim RVR^2 ROTR^2 (RADJ^2 - RADI^2)$ for the simple case where VAI=VAJ=90.

RVR even *slightly* less than unity signals shearing flow rather than solid body rotation, and this choice has several important repercussions:

First, the default methods for calculating torque and power input are likely inappropriate and should instead be replaced with a user-supplied mechanical torque (Section 3.26.3.2). This is controlled by the INTORQ parameter as explained in the next subsection.

Second, for tubes and STUBEs, rotary and effective total Reynolds numbers (REW and REX, respectively) become active and both friction and heat transfer will be affected (as described below in Section 3.26.4).

Third, FLUINT no longer solves along the flow streamline when RVR<1. The mass flow rate (FR) is actually a vector, but when RVR<1 that vector is not the same as the flow stream vector. For example, in Figure 3-49 the entrance path (horizontal) vector is the same as FR, but the streamline is along the total velocity vector, and is larger in magnitude. In other words, if RVR<1 then FLUINT is solving along a *component* of the streamline velocity (parallel to the axis of a spinning cylinder, or parallel to the radius of a spinning disk). There are various effects of this treatment:

1. If more than one path is used in series to represent such a passage (e.g., if the flow passage had been discretized, perhaps using a duct macro), then the pressures and temperatures for interior lumps with LSTAT=NORM won't represent a true static state. Since the actual total velocity is higher, the actual static temperature and pressure would be lower.

* Also referred to by various references as the "pumping factor," "core-swirl ratio," etc. Usually the variable K or β is used in equations when referring to this parameter.

† In this case, INTORQ should be selected to be YES, with an applied torque specified (perhaps using the correlations described in Section 7.11.7).

If a single path were used to represent such a passage (in other words, with “stationary” inlet and outlet lumps), then there exists a virtual state between those two lumps somewhat analogous to a path throat state (i.e., PLTH, TLTH) with a lower static temperature and pressure than perhaps either of the endpoint lumps.

This means derived properties such as density and viscosity will be somewhat in error, though perhaps the greatest danger is that a volatile liquid might flash within such a passage but that flashing would be missed by the FLUINT solution.

The routine STATICTPA (Section 7.11.1.10) can be used to estimate the static temperature and pressure along the streamline. However, that check is not invoked by default: it must be performed by the user if this error term is a concern.

2. The choking calculations in SINDA/FLUINT (Section 3.18) consider only the flow rate FR; they neglect effective rotational velocities (e.g., choking induced by very high values of RVR*ROTR*RAD for shearing flows with RVR<1). In other words, it is conceivable that choking might be missed for very high rotational speeds. A query routine, VELPATH (Section 7.11.1.10) is available for calculating the various velocity components including the magnitude of the total velocity, and this value can be compared against the sound speed if this is a potential concern.

For the Very Advanced User: In theory, it is possible to force FLUINT to solve along the streamline instead of along the axial or radial component of velocity. This is effected using VAI and/or VAJ less than or equal to -1000.0 as a signal. For example, if two paths were used in series to represent stream-wise flow though a journal bearing, both the VAJ of the first path and the VAI of the second path would be set to -1000.0, in which case the lump between those paths represents the true static state, and the FR through it represents the total velocity vector. However, many complications arise. The lengths and projected flow areas must be adjusted continually as a function of both ROTR and the current flow rate itself, a pumping term is required since the rotary shear is no longer perpendicular to the solved velocity vector, and built-in correlations for effective friction (see Section 3.26.4.1) are no longer applicable. No further guidance is available for such modeling: the use of the VAI/VAJ signal is merely available to support possible future developments either by CRTech or by the user community. Note that this VAI/VAJ stream-wise solution signal is only relevant for RVR<1.

3.26.3 Torque, Efficiency, and Power

Rotary motion imparts or extracts energy from the fluid network. The energy required to spin the fluid to a higher pressure (or conversely the energy extracted from the fluid as it drops to a lower pressure) is calculated by the program and is available as the translatable path output variable QTMK (which has units of either W or BTU/hr, depending on the value of UID selected). The power input QTMK will be added to the QTM (a translatable lump output variable) of the lump *currently* downstream of the rotating path. QTM is one component (along with QL and the sums of tie QTIEs and ftie QFs) of the total lump power input QDOT.

There are two modes available for calculating QTMK:

1. The default method (INTORQ=NO) assumes that the path acts like a simple pump or hydraulic turbine operating on incompressible fluid. This mode is most appropriate for solid body rotation, RVR=1, *even though it is applied by default regardless of the value of RVR.* The corresponding hydraulic torque (TORQ) is calculated as an output variable in this mode. These calculations can be influenced by a user-supplied hydraulic efficiency (EFFP) that defaults to unity.

2. An alternative method is available in which torque is specified (INTORQ=YES). This mode is more appropriate for relative wall motions (RVR<1) as occur in dissipative passages such as cavities, gaps, and seals. When INTORQ=YES, the program calculates QTMK on the basis of a user-specified TORQ. Section 7.11.7 describes correlations that are intended to assist in this specification of TORQ.

These two modes will be further described in subsections that follow. For both modes, a summary of the new parameters available as inputs for any path include:

EFFP. . . . . Overall efficiency (real, greater than 0.0 and less than or equal to 1.0). This is an input variable by default (INTORQ=NO) and an output variable if INTORQ=YES. EFFP defaults to 1.0.
Note that this EFFP parameter is distinct from the parallel variable for a PUMP, TURBINE, COMPRESS, or COMPPD path.

TORQ. . . . . Mechanical torque, output by default. If on the other hand, by electing INTORQ=YES, TORQ should be input in units consistent with TCF as described below. TORQ should be positive for shaft power added to the network, and negative for power extracted from the network.
Note that this TORQ parameter is perhaps distinct from the parallel variable for a PUMP, TURBINE, COMPRESS, or COMPPD path.

TCF . . . . . Correction factor for the path output variable TORQ, such that TORQ = TCF*FR*ΔP_s/(EFFP*DL_{up}*|ROTR|) has the desired units, where ΔP_s is the pressure force due to rotation (positive for a pressure rise), and DL_{up} is the effective suction density of the current inlet.
If UID=SI, TCF defaults to $60/(2\pi)$ such that TORQ is in N-m. If UID=ENG, TCF defaults to $144/(60*2\pi)$ such that TORQ is in lb_f-ft.
Note that this TCF is perhaps distinct from the parallel variable for a PUMP, TURBINE, COMPRESS, or COMPPD path.

3.26.3.1 Default Mode: Effective Turbomachine (user-defined EFFP)

By default (INTORQ=NO), path power QTMK is proportional to the volumetric flow rate times the pressure rise due to spin (FR*ΔP_s/DL_{up}), and is positive for such a pressure rise from inlet to outlet given a positive flow rate. ΔP_s is the pressure force due to rotation (positive for a pressure rise), and DL_{up} is the effective suction density of the current inlet (the compressibility of the fluid is neglected).

Similarly, the hydraulic torque TORQ required to impose the spinning (or extracted as a result of spinning) will be calculated by the program by default. This variable can be used to calculate shaft speeds or rotational accelerations (using perhaps the concurrently solved ODEs), and can be output in any desired units via the user-specified torque conversion factor TCF.

An optional unitless efficiency may be specified: EFFP, which defaults to unity. If EFFP*FR*ΔP_s is positive (energy added), then the result is divided by EFFP (as if the path were a quasi-pump). If EFFP*FR*ΔP_s is negative (energy extracted) then the result is multiplied by EFFP (as if the path were a quasi-hydraulic turbine).

Thus, for positive EFFP*FR*ΔP_s (energy added to network), both QTMK and TORQ will be positive, and EFFP appears in the denominator (effective pumping):

QTMK = c*FR*ΔP_s/(EFFP*DL_{up}) (c converts units for UID=ENG)
TORQ = TCF*FR*ΔP_s/(EFFP*DL_{up}*|ROTR|)

For negative EFFP*FR*ΔP_s (energy extracted from network), both QTMK and TORQ will be negative and so EFFP appears in the numerator (as if the path were a hydraulic turbine):

QTMK = c*FR*ΔP_s*EFFP/DL_{up} (c converts units for UID=ENG)
TORQ = TCF*FR*ΔP_s*EFFP/(DL_{up}*|ROTR|)

Notice that the sign of EFFP itself is used to determine whether energy should be added or subtracted from the network. For example, if FR is positive but ΔP_s is negative, positive EFFP is interpreted as a hydraulic turbine with energy *extracted* from the network. But if EFFP is negative in that same situation, then it is interpreted as a pump in which the flow is backwards and energy is to be *added* to the network. Unlike many other SINDA/FLUINT options, the positive direction of FR has meaning for rotating paths beyond that of a simple sign designation.

This mode (EFFP as an input) is most appropriate for solid body rotation (RVR=1). However, *it is applied by default regardless of the value of RVR unless INTORQ=YES as described below.*

3.26.3.2 Alternative Mode: Dissipative Element (user-defined TORQ)

The above relationships are followed by default (INTORQ=NO): TORQ and QTMK are calculated as functions of EFFP, which defaults to 1.0. For dissipative elements such as axial gaps, radial gaps, and seals (i.e, RVR<1), the user should usually instead specify TORQ (usually positive in this case) and have QTMK calculated as a function of this input TORQ. Use INTORQ=YES to elect TORQ as an input (rather than output) parameter,* in which case the following relationship is obeyed:

QTMK = c*TORQ*|ROTR|/TCF (c is unit conversions for UID=ENG)

The efficiency EFFP is also calculated in this mode for reference, consistent with the equations in the prior subsection. EFFP is forced to be a positive value by using the absolute value of FR*ΔP_s but may exceed unity since its meaning is often nonphysical in this case. The program will use EFFP= |FR*ΔP_s/QTMK| if TORQ is positive, and EFFP= |QTMK/(FR*ΔP_s)| if TORQ is negative.

* The correlations described in Section 7.11.7 are available to assist in this calculation of TORQ.

3.26.3.3 Notes on Rotating Turbomachinery Connectors

While the application of path rotation options to PUMP, TURBINE, COMPRESS, or COMPPD connectors is of dubious modeling value (i.e., “a pump that itself spins about an axis”), such use is not prohibited. Generally, rotating paths might be used to represent certain pumps (e.g., alternatives to PUMP connectors) but are intended instead help represent passages *within* such pumps.

Because of these differences, when the path is a PUMP, TURBINE, COMPRESS, or COMPPD connector, the TORQ and QTMK values are *not* calculated based on spinning (ROTR), but rather on the pump definition itself (SPD, HPMP, GPMP, etc.). Therefore, the EFFP and TCF values for PUMP connectors (as an example) are distinct from those for a generic rotating path. In fact the PUMP TCF factor will often have different units from the path rotation TCF value since ROTR may not be in the same units as is the pump SPD.

If a PUMP, TURBINE, COMPRESS, or COMPPD connector is assigned a nonzero ROTR value, an additional pressure force will result but the QTMK, TORQ, EFFP, and TCF values will be applied based on the turbomachinery connector relationships and not on the relationships described in Section 3.26.3.

Currently, INTORQ=YES is not supported for pumps, compressors, and turbines.*

3.26.4 Use with Tubes and STUBE Connectors

When tubes and STUBE connectors are rotated (nonzero ROTR), additional calculations are invoked, as detailed in this subsection.

When RVR=1 (no relative wall motion), the effects on tubes and STUBEs are similar to those of other paths: a rotary pressure force is superimposed upon the path, and heating (QTMK) and hydraulic torque (TORQ) are also calculated. Additionally, an acceleration perpendicular to the path is calculated that may influence two-phase flow regime prediction.

When RVR is less than unity (RVR<1), shear across the fluid is indicated due to relative wall motion. This affects both pressure drop and (if the tube or STUBE is a member of any HTN, HTNC, or HTNS ties) heat transfer, as described below.

3.26.4.1 Effect on Duct Pressure Drop (RVR<1)

When ROTR is nonzero but the walls are co-rotating (RVR=1), pumping and work terms will be applied to a passage but the frictional resistance of the path is assumed to be unaffected since the wall appears stationary from the viewpoint of the fluid.

When ROTR is nonzero but RVR<1, indicating shear, then the duct (tube and STUBE connector) friction calculations are modified to take into account the effective velocity of the fluid, with friction augmented in the turbulent regime to take into account the increased friction associated with rota-

* This will be enabled in future versions to support cases where turbomachine torque is known, but efficiency is not (e.g., data from a test stand).

tional shear. These methods should be viewed as reasonable estimates for generic situations; they should be modified or replaced with more specific models or data if available, using perhaps path friction modification factors such as FCTM.*

The rotational Reynolds number, REW (a translatable tube and STUBE connector variable†) is defined as $\omega RH\rho/\mu$, where ω is the rotational speed, R is the radius (maximum radius for disks), H is the gap size (either radial or axial), μ is dynamic viscosity, and ρ is the density. In SINDA/FLUINT terms, with the gap size assumed to be DH/2,‡ and ω calculated from ROTR based on UID:**

$$\text{REW} = \omega * \max(\text{RADI},\text{RADJ}) * (\text{DH}/2) * \rho / \mu$$

The effective Reynolds number, REX, is then calculated as:††

$$\text{REX} = \text{SQRT}\left((\text{RVR}*\text{REW})^2 + \text{REY}^2 \right)$$

where REY retains its meaning as the net flow Reynolds number based on the flow rate FR: REY = |FR|*DEFF/μ*AF.

The pressure drop calculations use the effective Reynolds number REX rather than REY both for determining laminar/turbulent transition, and as the starting point for calculating turbulent friction factors. (The dynamic head is still based on the velocity associated with the net flow rate FR, which is a component of the actual velocity as described in Section 3.26.2) In the laminar regime (REX<2000), on the other hand, the friction factor is based on REY instead of REX (e.g., f = 64/REY): the net flow and rotational velocity profiles are assumed linearly superimposable.

For example, if REY=1500 (laminar flow) but REW=4000 and RVR=0.5, then REX=2500 and a transitional friction factor will apply. In other words, rotation will cause the flow to transition to turbulence faster than REY will indicate.

In the turbulent regime (REX>2000), the total friction factor is calculated on the basis of the internal curve fit to the Moody chart using the effective Reynolds number REX, but that initial factor is then augmented to include the effects of rotational shear:‡‡

$$f_{eff,turb} = f(\text{REX},\text{WRF}) * \{ (1 + \text{RVR}^2/Y^2)^{3/8} + (1 + (1-\text{RVR})^2/Y^2)^{3/8} \}/2$$

---

* For example, for eccentric seals, a modification of both the RVR value and the effective gap size (DH) should be considered as needed to match test data or other references.

† If RVR=1, then REW=0 and REX=REY.

‡ This formula implicitly assumes that DH has been input by the user as the correct DH=2H. Note that the REY calculation itself actually employs DEFF, and DEFF=2DH/3=4H/3 for thin gaps. This distinction (a factor of 2/3 vs. 1 in the REY calculation) represents a slight departure from the effective velocity method, but produces a more accurate answer for laminar flow.

** If UID=SI, $\omega = (2\pi/60)*|\text{ROTR}|$ (radians/sec), and if UID=ENG, $\omega = (2\pi*60)*|\text{ROTR}|$ (radians/hr).

†† This formula represents a generalization of the effective velocity method: Eqn 17 in Carl Gazley Jr., "Heat Transfer Characteristics of the Rotational and Axial Flow Between Concentric Cylinders," Transactions of ASME, 1985.

‡‡ This correction is generalized from the lambda factor applied to turbulent flow in annular gaps by Kurokawa et al: eqn. 7 in "Axial Thrust Behavior ...," Journal of Propulsion and Power, pp 244-250, Vol. 10, No. 2, 1994. (In that reference, the uncorrected f is a function of REY, not REX.) The scaling of flow resistance with $(\omega R)^{3/4}$, which itself is based on a scaling of shear stress with $(\omega R)^{7/4}$, is also contained in a model of turbulent radial flow with "inertia" (pumping) in an axial gap, per eqn. 11 of NASA SP-8121, 1978. Reasonable comparisons have been made with the methods suggested by that document for flow in face seals.

where Y = 4*REY/(7*REW)

For example, for axial flow in a annular gap (RADI=RADJ), increased rotational speed results in increased resistance to axial flow. Such a situation might arise for axial flow through a journal bearing, where turbulence itself might be induced in an otherwise laminar flow by high rotational speeds. A similar effect (increased resistance with increased rotational speed) exists for radial flow in an axial gap (e.g., flow between a stationary and a rotating disk), though that situation is complicated by the pumping terms since RADI is not equal to RADJ.

For radial flow within axial gaps, REW and therefore REX will change along the radius even for incompressible flows because max(RADI,RADJ) will change. Therefore, this might require subdivision of the passage to capture these changes along the flow path.

For thin gaps and seals, extra discretization should also be considered to capture the heating effects of dissipative torques (Section 3.26.3) since heating can affect fluid properties and velocities along the flow stream, which in turn affect resistance and pumping terms. A duct macro might therefore be considered instead of a single tube or STUBE connector.

For flow in hot gas seals, the FCRAREF correction (Section 7.11.6.2) might be required if the gap size is small enough to be on the order of the mean free path for the gas molecules.

Application of rotational shear to two-phase friction factors is legal, and is performed for consistency and continuity and will certainly represent a good first estimate. However, those extrapolations do not represent a validated approach. The use of twinned (slip flow) paths is similarly legal, but is discouraged. (The accelerations associated with rotation will likely result in a dispersed flow regime with minimal slip in any case.)

3.26.4.2 Effect on Convection Ties (RVR<1)

When a forced convection tie (HTN, HTNC, or HTNS) is made to a lump, one or more paths are named to describe the wetted area, velocity, and other information required to complete the invocation of internal correlations for forced convection heat transfer. For single-phase flow, these correlations consist of constant isothermal Nusselt numbers for laminar flow, and the Dittus-Boelter correlation for turbulent flow. All of these parameters have defaults that can be overridden tie by tie, and scaling parameters exist as well. Refer to the description of parameters such as XNUL, CDB, and UER in Section 3.6.6.

For high speed flows, the tie effective fluid temperature TEF begins to deviate from the lump temperature TL; TEF represents the adiabatic surface temperature (see Section 3.6.5). In other words, at high speeds the fluid appears hotter than the static temperature because it must be slowed within the boundary layer (though not quite to the stagnation point).

For paths with co-rotating walls (RVR=1, or “solid body” rotation), heat transfer calculations are assumed to be unaffected by the rotation itself. This approximation is good for small gaps and for axial flows within radial gaps. For large axial gaps (radial flow or even in the absence of net flow), the default methods will underestimate heat transfer since they neglect secondary flows due to the buoyancy effect. To assess the order of magnitude error, consider applying the natural convection routines (e.g., NCVFPT in Section 7.11.4.8) with artificially high “gravity” (via the NCSET

routine) set to the rotational acceleration: GEEF = $R\omega^2/g$ (where g is the acceleration of gravity, R is the radius, and w is the rotation rate in radians/time).

The remainder of this section describes heat transfer corrections invoked for rotating, shearing flows: convection ties made to tubes or STUBE connectors for which ROTR is nonzero but RVR<1.

Effective fluid Temperature (TEF): Even for low net flow rates (FR), the TEF of a tie may be elevated if ROTR is high enough. In other words, the fluid appears hotter to the wall due to the velocity of rotation (as indicated by the Reynolds number REW multiplied by RVR) in addition to the net velocity (as indicated by the Reynolds number REY).

Using RVR*max(RADI,RADJ)*ω as the effective core velocity is technically true only from the perspective of the stationary wall (stator or housing). From the point of view of the moving wall (rotor or shaft), the fluid's velocity is (1-RVR)*max(RADI,RADJ)*ω. However, FLUINT makes no distinction between rotating and stationary walls: the heat transfer is assumed symmetric and is applied to any wall, rotating or not. Fortunately, RVR is usually not far from 0.5 when it is not unity, so this caution is need only be heeded in rare situations such as vaned gaps or seals (with RVR on the order of 0.9 perhaps) combined with high rotation speeds where heat transfer is also important. In such extreme cases, the default heat transfer methods themselves (as described next) are not likely to be applicable without modification.

Heat Transfer Coefficient Calculations: Generalized methods for heat transfer within rotating and shearing flows do not exist, often because heat transfer is a secondary or even tertiary consideration behind prediction of leakage flow rates, pressures, and torques. SINDA/FLUINT calculations are therefore simple but consistent: the effective Reynolds number REX is substituted for the net flow Reynolds number REY in the standard heat transfer correlation set. Principally, this invokes the Dittus-Boelter correlation for single-phase turbulent flow, with the Nusselt number proportional to the Reynolds number to the 0.8 power (UER=0.8 by default). No distinction is made between radial or axial flows, nor between radial in-flow and radial out-flow. As was mentioned above, no distinction is made between heat transfer to the rotating wall versus the stationary wall.

The resulting SINDA/FLUINT predictions can be considered only approximate. However, they are consistent with the few methods that *are* available, such as those listed in Owen (1989),* which also employ similar scaling with velocity in the limits of very high rotation rates or very high net flow rates. SINDA/FLUINT results should be calibrated to available test data or at least uncertainties tested using variations of the default values of tie parameters such as XNUL, XNLM, XNTM, CDB, UER, etc.

The application of these default methods to two-phase (and rotating, shearing) flows is done for consistency (e.g., survival during steady-state excursions) but has no basis in any test data, and so should any predictions for two-phase flow not be used except for order-of-magnitude estimations, or as the basis for calibrating to available test data.

* 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. Some of the correlations contained within that reference are available as user callable subroutines, which can be applied either to HTU/HTUS ties or as a check on the order of magnitude error in the default SINDA/FLUINT methods for HTN/HTNC/HTNS ties.

3.27 Flat-front Two-phase Flow Modeling

Consider the simulation of an empty (gas-only) line being filled from one end with a liquid. Or, consider the case of a pressurized gas used to purge a line that is initially full of liquid. The general-purpose default two-phase methods in FLUINT would spread the liquid-gas interface over several lumps, in effect spraying the liquid into the gas line or bubbling the gas into the liquid line. This section describes an alternative approach: the user can instead assume that the liquid and gas experience little mixing at the interface, and that the liquid-gas interface travels as a planar surface ("front") through the piping system: the liquid and gas can be separated *axially.*

For certain simple problems, this *flat-front* assumption can best be modeled using compliant tanks or perhaps ifaces (Section 3.8). For example, a filling line might be represented as a tube whose length grows in proportion to the volume of the very compliant (COMP >> 0) tank into which it empties: TLEN = VOL#down/AF#this. (The tank pressure represents the pressure of the liquid-gas interface.) Logic or expressions can be used to detect a full line, perhaps reducing the compliance of the downstream tank to that of the liquid and/or wall: COMP = (TLEN302 > Lmax)? CompLW: CompBig. Such *unfurling methods* are cumbersome if more than one path is required to model the filling or purging duct, however. Worse, they are also only appropriate if there is very low pressure gas or vacuum initially in the line (except for trapped gas volumes within dead end lines), and if the liquid is nonvolatile.

The ability to model two-phase flows as a propagation of a flat front can be met via the combined use of two otherwise independent options: a directional phase-specific suction option applicable to all paths (Section 3.27.2), and a user-specified flow regime for "axially stratified" flows (Section 3.27.3). While these features can be used separately for other purposes, *they can be used together when modeling propagation of a flat liquid-gas interface through ducts.*

Paths and duct macros may be defined using the shortcuts "FF=FILL" (for liquid filling a line in the positive flow direction) or "FF=PURGE" (for gas purging liquid from a line, with the gas flowing in the positive flow direction). These shortcuts set the regime/correlation identifier (IPDC), the phase suction status (STAT), and the upstream fraction (UPF) factors to values that are appropriate for flat-front modeling, as will be described below.

When flat-front options are used for tubes, the code will automatically include capillary forces at the interface assuming full wetting and a capillary radius of half the hydraulic diameter: $\Delta P_{cap} = 4*\sigma/DH$ where σ is the surface tension. This meniscus term is normally negligible, but it may be important in studies of filling or purging capillaries and other very small-diameter flow passages.

Flat-front modeling is *not* a general-purpose option to be used with any class of two-phase flow. Rather, it is intended to handle a specific "degenerate" class of problems for which the extreme assumptions implicit in the flat-front approach can be made. Significant limitations therefore apply:

1. **Transients Only.** *Flat-front modeling is intended for transient flow only.* A flat front either sweeps through a system (and hence it has already filled or emptied the system at steady-state), or else the flow rate is zero as an initial condition. In the latter case (e.g., a stagnant vertical pipe with liquid at the bottom and gas at the top), a STEADY analysis is likely to

be unsuccessful since zero flow in a junction leaves it in an undefined state. Therefore, special care may be needed to initialize a model in FloCAD, Sinaps, or via logic in OPERATIONS before beginning a transient.

2. **Nonvolatile (or at least low-volatility) Liquids.** *Flat-front modeling is not an appropriate assumption to make if significant phase change occurs*: if the liquid boils and/or the vapor condenses. It is not appropriate for quenching a hot line with cryogenic liquid, for example. 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). A single liquid-gas front is not required,* but any zone containing two-phases will be assumed to have an axial division between the phases: a liquid slug or a large gas/vapor bubble.

   This limitation can be partially overcome using the DUCT_MODE routine (Section 7.11.1.4), 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 still be a single distinct phase change front within the duct.

3. **Tanks and Tubes.** Tanks are essentially *required* when modeling ducts in the flat-front approach; junctions may be used for limited purposes, but should be avoided if possible. Tubes are also *strongly* recommended instead of STUBE connectors. Within this section, the term *tube* will therefore be used, even though under rare circumstances an STUBE might be substituted.

   Ideally, one would like to use tubes for the liquid portions of the model, and STUBE connectors for the gas portions (gas volumetric flow rates will be the same as liquid volumetric flow rates as a front moves through a system, and this means a very small gas mass flow rate). Unfortunately, path types cannot be changed dynamically (during a run) within SINDA/FLUINT. Furthermore, the instantaneous nature of an STUBE connector can result in instabilities given the strong (essentially infinite) gradients that result from a flat-front approach. Therefore, small time steps often result due to flow rate change limits within the gas tubes, especially near the interfaces.

   If small time steps become a problem due to such causes, consider a combination of raising DTTUBF while reducing DTMAXF. Also, since the fill/purge events typically modeled with a flat-front approach can be violent, either use a thermodynamically compressible liquid, or consider use of the COMPLQ routine to avoid an assumption of incompressible liquid. Also, consider in addition the WAVLIM routine to track pressure waves if appropriate.†

---

* See limitations on trapped slugs and bubbles (with multiple fronts) that result from discretization requirements, as described in Section 3.27.1. Also, only a single front is permitted with DUCT_MODE.

† Flat-front methods have yet to be validated against test data, and computational noise exists in the pressure solutions (Section 3.27.5.3) which further reduces its applicability to estimation of flow-induced loads. Flat-front modeling should be used with extreme caution for pressure estimations, and is more appropriate for design cases such as estimating how long it will take to fill or purge a line or network.

4. **Twinned Tanks.** *Twinned tanks are strongly recommended for flat-front modeling.* When using twinned tanks in duct macros, tubes must similarly be twinned.* However, when axially stratified (IPDC=-5, as defined below in Section 3.27.3) is specified, the tubes will remain homogeneous even if they are twinned, since slip flow is intrinsically impossible with a flat front assumption.

   Therefore, unlike other cases, such twinned tubes might be relocated by the program to the secondary tanks to affect the desired phase suction action. *When using twinned tanks connected to tubes with IPDC=-5, use of either STAT=LRV or STAT=VRL (see below) is mandatory.*

   When using twinned tanks for flat-front modeling, consider using negative values of VDRP and VBUB to maintain a separation of phases. This usage helps satisfy the requirement in item #2 above.

3.27.1 Discretization Requirements

A FLUINT tank represents both the pressure at a *point* in space, as well as the pressure within any compressible gas or vapor *averaged over the length* of the line segment represented by that tank. As the liquid or gas front passes through a tank, its pressure can change abruptly, causing a conflict between the above two meanings of tank static pressure.

To preserve the meaning of a tank pressure as both “the pressure at this location” and “the average gas or saturation pressure over a finite distance” without requiring excessive spatial resolution, flat-front modeling requires that a specific discretization scheme be followed. Unlike other two-phase flows, for which centered discretization (the “C” option in duct macros) is preferred, for flat-front models either an up- or downstream scheme *must* be employed† as depicted in Figure 3-50.

Conveniently, an FF=FILL model is symmetric to an FF=PURGE model with respect to flow rate direction: if the flow reverses direction in an FF=FILL model, the conditions for a purge model in the reverse direction have been met. Therefore, a model of a reversing or even oscillating liquid-gas front can easily be built.

When a tube is defined using these flat-front options, the tube is associated with the downstream tank (for FF=FILL) or the upstream tank (for FF=PURGE) to form a “flat-front segment.” In Figure 3-50, three such segments are shown for each line. The program internally recognizes such a segment and may apply special considerations to such a flat-front tank/tube pair.

This unique discretization requirement, combined with the lack of a steady-state and the need for twinned tanks and tubes, means that a general-purpose model cannot be built for a single system that will meets all analysis objectives; *a special-purpose model of a system must be built if flat-front fill or purge analyses are required.*‡

---

* This is not a restriction specific to flat-front modeling: it applies to any duct macro with twinned tanks.

† Centered discretization can result in numerical instabilities if the liquid column has strong pressure gradients; while it is not illegal when using flat-front modeling options, it should be avoided unless pressure gradients are always very small.

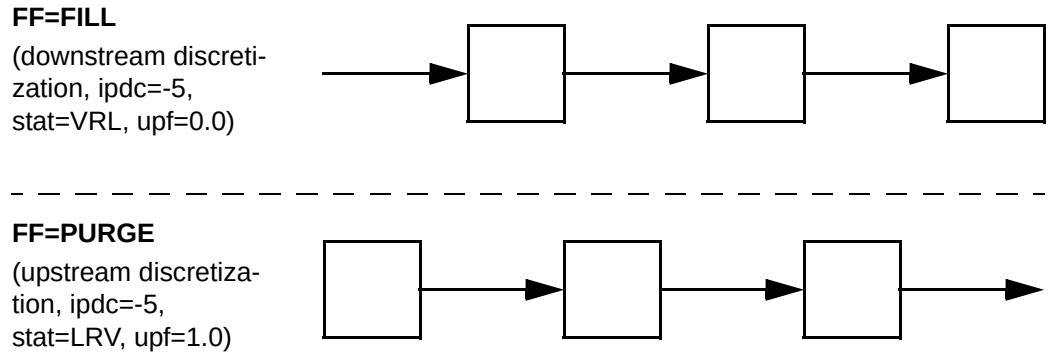


Figure 3-50 Flat-front (Fill or Purge) Discretization Schemes

The discretization requirements for flat-front modeling impose further restrictions on possible applications:

1. **Traveling liquid plugs or long gas bubbles are precluded.** For fill modeling, liquid must travel as a front in the positive direction (or gas as a front through the opposite direction). A plug of liquid cannot be pushed through a network by gas pressure, since this would mean conflicting discretization would be required: FF=PURGE in the upstream end of the liquid plug, and FF=FILL in the downstream end. Similarly, a length of gas bubble cannot be tracked through the system using flat-front methods.*

2. **Trapped liquid plugs or gas bubbles require special treatment.** In complex networks, such as manifolded parallel lines, it is possible (especially with the added effects of capillarity or gravity) to have lines in which gas becomes trapped during a fill event (with liquid at both ends of the line), or in which liquid becomes trapped during a purge event (with gas at both ends of the line). When this happens, the discretization and phase-specific suction at the far end of the line is likely of the opposite of what it needs to be to suitably represent the front.

   This problem is somewhat like the traveling plug/bubble problem noted above in that two flat fronts are present in the same line. However, since the plug or bubble doesn't travel far, the zone in which each front can move is "known" (or at least presumed). Therefore, a solution is possible: use one discretization method on one end of the line and the opposite

‡ This is not meant to imply that the entire model must use flat-front assumptions and discretization. For example, consider a horizontal line that is being filled, with a smaller vertical line draining it from the middle. The horizontal line might be modeled using flat-front (FF=FILL) methods, but the vertical line will not contain a front and therefore can be modeled using normal center-discretized, IPDC=6, etc. methods. In fact, this vertical line might contain counter-current flow (liquid falling, vapor rising), if so it *can't* use flat-front approximations. (If this were the case, the horizontal line could use twinned tanks, and the vertical line would use twinned tubes but perhaps probably homogeneous, untwinned tanks.) See also DUCT_MODE (Section 7.11.1.4)

* A possible exception can be made for very slow moving flows, with pressure gradients small enough such that centered discretization does not become unstable. In this case, the spatial resolution must be smaller than the expected plug or bubble length, and user logic would be needed to detect the trailing edge of the plug or bubble, and reverse the original phase suction (using CHGSUC calls to change from STAT=VRL on the leading edge of a liquid plug to STAT=LRV on the trailing edge, or the reverse for a long gas bubble).

discretization on the other end of the line. For example, for a trapped bubble, use the FF=FILL (top half of Figure 3-50) at the inlet, and the FF=PURGE discretization (bottom half of Figure 3-50) at the outlet where liquid might enter in the negative flow rate direction.

This “opposing discretization” method of course precludes modeling the entire line with a single duct macro, and forces the user to assume some middle point which will always be liquid (for a trapped plug) or that will always be gas (for a trapped bubble). Such knowledge might only come from preliminary runs that employed guessed locations at which bubbles or liquid might become trapped.

3.27.2 Flat-front Phase Suction Options

Two phase suction options are available to assist with flat-front modeling in combination with IPDC = -5 (Section 3.27.3). However, they can be used for other purposes than flat-front modeling, and they can be applied to any non-capillary path.

These two options are:

STAT = VRL: gas/vapor in forward direction, liquid in reverse direction. This option is essentially a combination of the options VS and RLS. This option should be used for a *liquid front flowing in the positive direction*, or a gas front flowing in the reverse (negative FR) direction. STAT = VRL is automatically set if FF=FILL is specified. For example, use this option to model the filling of a gas line with liquid introduced in the normal (positive) flow direction. For a homogeneous (untwinned) tube with a twinned tank upstream, define the upstream connection to the secondary tank to preferentially extract gas in the nonequilibrium mode. For a homogeneous (untwinned) tube with a twinned tank *downstream*, define the downstream connection to the primary tank to preferentially extract liquid in the nonequilibrium mode when flow reverses.

STAT = LRV: liquid in forward direction, gas/vapor in reverse direction. This option is essentially a combination of the options LS and RVS. This option should be used for a *gas front flowing in the positive direction*, or a liquid front flowing in the reverse (negative FR) direction. STAT = LRV is automatically set if FF=PURGE is specified. For example, use this option to model the purging of a liquid line with gas introduced in the normal (positive) flow direction. For a homogeneous (untwinned) tube with a twinned tank upstream, define the upstream connection to the primary tank to preferentially extract liquid in the nonequilibrium mode. For a homogeneous (untwinned) tube with a twinned tank *downstream*, define the downstream connection to the secondary tank to preferentially extract gas in the nonequilibrium mode when flow reverses.

Unlike any other phase suction options (other than STAT=NORM), STAT=LRV and STAT=VRL *are* allowed within duct macros (LINE and HX, as described in Section 3.9.2) and will apply to every tube generated in those macros.

3.27.3 Axially Stratified Flow Regime

A special user-specified flow regime, “axially stratified,” can be invoked by selecting IPDC = -5 for any tube or STUBE connector.* This regime is *never* chosen by the built-in regime predictor (IPDC=6): it only exists as an adjunct for flat-front two-phase flow modeling. The axially stratified regime will be selected if either FF=FILL or FF=PURGE is specified.

When this regime is selected, a single “front” (such as is depicted in Figure 3-51) is not specifically assumed. Rather, any liquid and gas within the tube is assumed to completely fill the tube diametrically: phases are assumed to subtend the full diameter of the duct such that any interface between phases is flat. This implicitly assumes equal phasic velocities, and therefore slip flow is disabled in twinned tubes if IPDC = -5 is selected.

However, absence of slip flow does not eliminate the possibility of using twinned tanks in duct macros, and in fact such a selection is often even more important in flat-front flow due to compressive heating and cooling of the gas and the small size of the liquid-gas interface. When using IPDC = -5 with twinned tanks in duct macros, use of either STAT=VRL or STAT=LRV (or the corresponding FF=FILL or FF=PURGE shortcuts) is mandatory, and may cause the end-points of the tubes to be moved by the program as the front progresses through the line. Twinned tubes are required with twinned tanks in duct macros, even though they will remain homogeneous during flat-front simulations.

As with any two-phase flow regime, selection of this specialized regime affects friction and inertia calculations, body force calculations, axial FTIE conductances, as well as heat transfer calculations for nonvolatile/noncondensible mixtures. However selection of this regime does *not* affect heat transfer calculations in the limit of volatile/condensible two-phase flows (in which case the assumption of a flat-front is questionable, as was noted above).

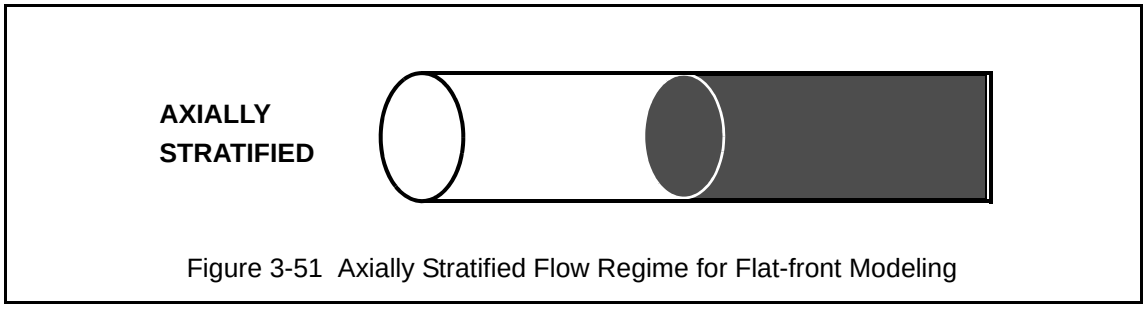


Figure 3-51 Axially Stratified Flow Regime for Flat-front Modeling

To visualize the numerical methods used for flat-front modeling, imagine that a liquid front has progressed half-way through the three-tank model represented by the upper part of Figure 3-50. The first tank will be full (if twinned, it will be collapsed into the homogeneous mode with only the primary tank remaining). The first tube will contain 100% liquid and will exhibit the friction and inertia corresponding to such a column of liquid. The last tank will be empty (containing only gas),

* Tubes are strongly recommended instead of STUBE connectors for flat-front modeling, since STUBE connectors can become unstable as the front passes through them. The term *tube* will therefore be used almost exclusively within this section.

so the last tube will exhibit the low frictional resistance (due to the low velocity) and low inertia (due to the low density) of the gas flow. If homogeneous (untwinned), the middle tank will be half full (AL=0.5). If twinned, then both the liquid and gas/vapor twins will have the same volume (namely, half of the total), and the upstream (liquid) tube will connect to the primary (liquid) twin, and the downstream (gas) tube will connect to the secondary (gas/vapor) twin. The middle tube, whose behavior is dependent on this half-full control volume (because UPF=0.0), will exhibit the frictional resistance and inertia of half a length of liquid column (neglecting gas density and velocity in this description, even though they actually aren't neglected in the program itself).

3.27.4 Brief Input Examples of Flat-front Modeling in Duct Macros

For clarity, some input examples are provided for HX macros using flat-front modeling methods. (LINE macros work analogously.)

The following defines a downstream HX macro, initially full of gas, that is to be filled from the defined inlet (lump #201) with liquid:

```
m hx,1,d,1,1,1,wall.1,201, nseg = 10
      FF=FILL $ equivalent to ipdc= -5, stat=VRL, upf=0., pa=tube, lu=tank
      dhs = diam, tlent = length, xl = 1.0
```

The following is the same as the above, but uses twinned tanks to allow the gas that is next to the liquid to heat, perhaps, if it is compressed by the liquid. Notice that the paths must be twinned too to satisfy general network construction rules, even though they will actually remain homogeneous:

```
m hx,2,d,1,1,1,wall.1,201, nseg = 10
      FF=FILL $ equivalent to ipdc= -5, stat=VRL, upf=0., pa=tube, lu=tank
      ltwin = 1001, ttwin = 1001, twin = 1001
      dhs = diam, tlent = length, xl = 1.0
```

The above models would also be suitable for the purging of a line in the reverse (negative flow rate) direction.

The following example is the same as the above, except that the line is initially full of liquid and gas is to be injected from the defined inlet (which is not contained within this example; lump #301 is the defined outlet):

```
m hx,3,u,101,101,101,wall.101,301, nseg = 10
      FF=PURGE $ equivalent to ipdc= -5, stat=LRV, upf=1., pa=tube, lu=tank
      ltwin = 1201, ttwin = 1201, twin = 1201
      dhs = diam, tlent = length, xl = 0.0
```

3.27.5 Notes on Flat-front Modeling

3.27.5.1 Nonduct Devices

Systems through which liquid-gas fronts pass often contain orifices, pumps, nozzles, tees, and valves as well as tubes. In fact, FF=FILL or FF=PURGE can be set in path defaults (PA DEF subblock), and appropriate portions of those shortcuts will apply to subsequent paths of any type.

If connecting to a twinned tank that is part of a flat-front segment:

> If FF=FILL, to be consistent with STAT=VRL, *connect to the secondary (vapor/gas) twin on upstream tanks*, and to the primary (liquid) twin on downstream tanks. Consider also using a path end location ("port", see 3.27.5.4).
>
> If FF=PURGE, to be consistent with STAT=LRV, connect to the primary (liquid) twin on upstream tanks, and *to the secondary (vapor/gas) twin on downstream tanks*. Consider also using a path end location ("port", see 3.27.5.4).

However, unlike a tube, there is no way to define "percent full" for a valve or orifice. An ORIFICE, for example, is a simple representation of a real device, having a single state as an inlet. When a liquid-gas interface reaches that inlet, it will transition instantaneously from 100% liquid to 100% gas (or the reverse). Such an oversimplified representation can cause sudden and largely artificial pressure surges.

The user should watch for problems, ignoring certain responses and/or alleviating them by ending phase-specific suction before the upstream tank has completely filled or emptied. For example, a call to CHGSUC could set STAT=NORM for a device being flooded when the upstream void fraction drops below (say) 1%, helping to eliminate the more extreme discontinuity that will otherwise occur when all gas has disappeared in the upstream tank.

For example, if path 2030 is flowing from (i.e., is downstream of) tank 20, then the following logic might be used to terminate vapor-only suction when the upstream is near to transitioning (in effect, turning the flat front into more of a spray). This logic is only executed once (per the *mtest* check) and has been generalized for the case where tank 20 has a vapor twin (tank 21 in this case):

```
header flogic 0, flow
      if(mtest .eq. 0)then
            if(vol21 .ne. 0.0)then
                  if(vol21/(vol20+vol21) .le. 0.01)then
                        call notwin('flow',20)
                        call chgsuc('flow',2030,'norm')
                        mtest = 1
                  endif
            elseif(al20 .le. 0.01)then
                  call chgsuc('flow',2030,'norm')
                  mtest = 1
            endif
      endif
```

Turbomachinery (PUMP, TURBINE, etc.) and TABULAR devices are inappropriate to use in a flat-front model without taking further measures. For TABULAR devices, perhaps the “gradual filling” of this device could be accomplished if multiple performance curves were specified as a function of the contents of the upstream tank (for purging) or the downstream tank (for filling), with OFAC a function of the up- or downstream void fraction or quality.

3.27.5.2 Flat-front at Tees

When liquid arrives at a tee (flow split), it will tend to continue in the same direction as the incoming vector. This inertial “carry-over” is only crudely represented by the default tee resistance correlations, since those were never intended for two-phase flows. Until more complete methods are available, applying a K-factor loss of 1.0 to side-flow branches* is somewhat representative of the need to accelerate the fluid from zero velocity (from the perspective of the side flow branch).

When a gas front arrives at a tee, the inertial carry-over is not as much of a concern, but the upstream discretization needed for flat-front modeling of a purging network can be problematic. Since the split point itself will likely be modeled as a tank (or pair of twinned tanks) representing line segments *downstream* of it, the question then arises: *which* downstream leg does that volume represent? One solution is to insert a LOSS element representing an estimate of tee losses for the branching leg, and let the volume of the split point to represent the continuing flow duct only and not the branching leg. In the following example, LINE macro #1 branches at the middle (#5) tank to LINE macro #2:

```
pa def, ff=purge, dh=diam
m line,1,U,1,1,991,     nseg=10, dhs=diam, tlent=length
      ltwin = 1001, twin = 1001
pa conn,200,5,1201, dev=loss, fk=1.0
m line,2,U,201,201,992, nseg=5, dhs=diam, tlent=length2
      ltwin = 1201, twin = 1201, pa=tube, lu=tank
```

Notice that the branching loss path (#200, leaving the middle of duct macro #1 and entering duct macro #2) connects the primary twin (tank #5) on the upstream end to the secondary twin (tank #1201) on the downstream end.

The use of wye-tee correlations such as YTKALG or YTKALI (Section 7.11.9.7) is normally appropriate only for homogeneous models since the wye-tee routines cannot handle twinned tanks much less two-phase flow at the merge/split point. Since twinned tanks are preferable to homogeneous tanks for flat-front modeling, this means that the wye-tee correlations are largely unavailable for flat-front modeling unless logic is applied to disable their use while the tank at the merge/split point is in a twinned state (i.e., while the volume of the secondary twin is nonzero). Such an action

* K-factor losses in a tube are applied based on upstream densities, independent of UPF. Thus, once the tank representing the tee is full, the K-factor loss will be based on liquid flow even if the tube (whose content is based on the lump downstream of it) is still mostly empty.

disables these correlations for a condition under which they are not appropriate anyway. An example of such usage for an FF=FILL problem is provided below, in which upstream tube #3 fills tank #3 (with twin #1003), tube #4 is the continuing path, and tube #201 is the branching path:

```
c          ^
c          | 201
c          |
c -----> 3,1003 ----->
c   3             4
c
      if(vol1003 .eq. 0.0 .and. (xl3 .eq. 0.0 .or. xl3 .eq. 1.0))then
            call ytkalg('fill',201,4,3,ftest,gtest,90.0,0.0)
            call ytconv('fill',201,4,3,ftest,gtest,fk201,fk4)
      else
            fk4   = 0.2       $ guessed/temporary values
            fk201 = 0.2
      endif
```

3.27.5.3 Computational Noise at Transitions

Flat-front modeling is based on an extreme assumption: that liquid and gas are perfectly separated, creating a perfect discontinuity in properties that travels through the fluid network. Even though this situation might seem to represent a simpler problem to solve than that posed by most two-phase flows, it creates special problems for a general-purpose solver like FLUINT that is based on finite differences and finite volumes. Traditional spatial gradients do not exist at the front: no amount of increased spatial resolution will resolve a step function in properties, which is what the flat front represents.

The situation is further complicated by the state transitions that are experienced in both tanks and tubes as a front passes through them. If the flow rate in a tube is plotted versus time, a step function will be evident in the mass flow rate of the tube as the front passes through it, in part because the program attempts to preserve continuity of velocity through the event. At the exact same time, another discontinuity occurs. Consider a purging case with untwinned tanks.* The tank just upstream of the aforementioned tube is undergoing a transition between a “hard-filled” (all liquid) state and a compressible state containing a tiny gas and/or vapor bubble that dominates the tank’s hydrodynamic response. This transition is problematic for any tank (Section 3.19.3.5), but is especially so in a flat-front model because the tube is transitioning at the same time and because no amount of increased resolution can help to better resolve the discrete event.

Spikes or notches in the tank pressure can therefore occur as the front passes through, due to purely numerical causes: they are artificial and should be disregarded. Most plots will not reveal these extremely high frequency and low amplitude events, though in rare cases they are of sufficient amplitude to trigger property warning messages. The principle concern is that these spikes and notches can invalidate any logic (usually placed in FLOGIC 2) that seeks to capture maximum or

* Twinned tanks use a special method that tends to smooth the transitions more, as discussed later, so transitions are often more severe with single, homogeneous (untwinned) tanks. This is one of several reasons why twinned tanks are strongly recommended when using flat front methods.

minimum pressures after every time step. Such pressures might be of interest for flow-induced loads or for detecting auto-ignition of monopropellants, or for estimateing flow-induced loads as the front reaches a fitting.

Unless measures are taken to filter out these high-frequency artificial pressure responses, *the pressure responses cannot be used directly.* Similarly, the pressure response as the front moves through a nozzle, valve, or orifice is largely artificial, as was discussed in Section 3.27.5.1.

Pressure responses *can* be better trusted if the front terminates in a dead-end line containing a gas, especially if it contains a noncondensible species. Otherwise, flat-front modeling is appropriate for calculating the time it takes to fill or purge a network and identifying stranded (unfilled or unpurged) sections.

Flat-front models could also be used for calculating the fluid velocity *right before* the front strikes a valve, nozzle, or evacuated dead-end. This velocity could be used for hand-calculations (or side calculations in logic) of the maximum pressures that could occur, thus avoiding the numerical noise in the software's predictions of the impending event.

Transitions with Twinned Tanks—When the liquid/vapor interface moves from one pair of twinned tanks to a downstream pair *within a duct macro,* special methods are employed to (1) keep the upstream pair of tanks from collapsing to a homogeneous (well-stirred) condition as the void fraction limit is hit, (2) keep the downstream pair from starting in a homogeneous condition before accruing enough of the incoming phase to split into the separated mode. For example, if a warm purge gas were displacing a volatile liquid, these special methods preserve the warm temperature of the gas by not artificially mixing it with the cooler liquid (or warm this liquid and cause it to boil by mixing it with the warm gas). To achieve this effect, the front will appear to jump slightly at the transition (between 0.1 and 0.2% of the length of a path) at the transition, as the liquid/vapor front disappears from the upstream pair of tanks and re-appears in the downstream pair instantly.

The sudden jump might cause a little oscillation in the results if subsequent time steps are very small. Such oscillations are caused by the reintroduction of the small mass and energy errors that were accrued in the jump, and they can be reduced (spread over a longer correction period) using control constants such as RMFRAC and RMRATE. Normally however, these effects are not noticeable.

3.27.5.4 Flat Front Interactions with Path Ports

"Ports," or path end locations that are distinct from their end point lumps (Section 3.13.4), are often used in conjunction with flat-front modeling methods, especially when gravity or accelerations cause the liquid/vapor interface to flatten in both vessels and attached piping. However, interactions between these two options can be complicated.

To understand these interactions, it helps to recognize that the tank downstream of an FF=FILL tube have a close relationship: together they model the fluid within an axial *segment* of a line. The tube represents the shape and velocity of the segment, and the tank represents the volume, pressure (at the liquid/vapor interface), and temperature of the volume within that same segment. It would be rare for the volume of the tank to be different from the flow area of the tube times its length (AF*TLEN). This relationship is true whether the tanks and tubes are twinned or homogeneous.

For an FF=PURGE tube, the same relationship is true of the upstream tank.

Figure 3-52 presents a diagram of such a generic segment, whether for a purge or fill model, and whether the tanks and tubes are twinned or not (even though this figure depicts twinned tanks and tubes).

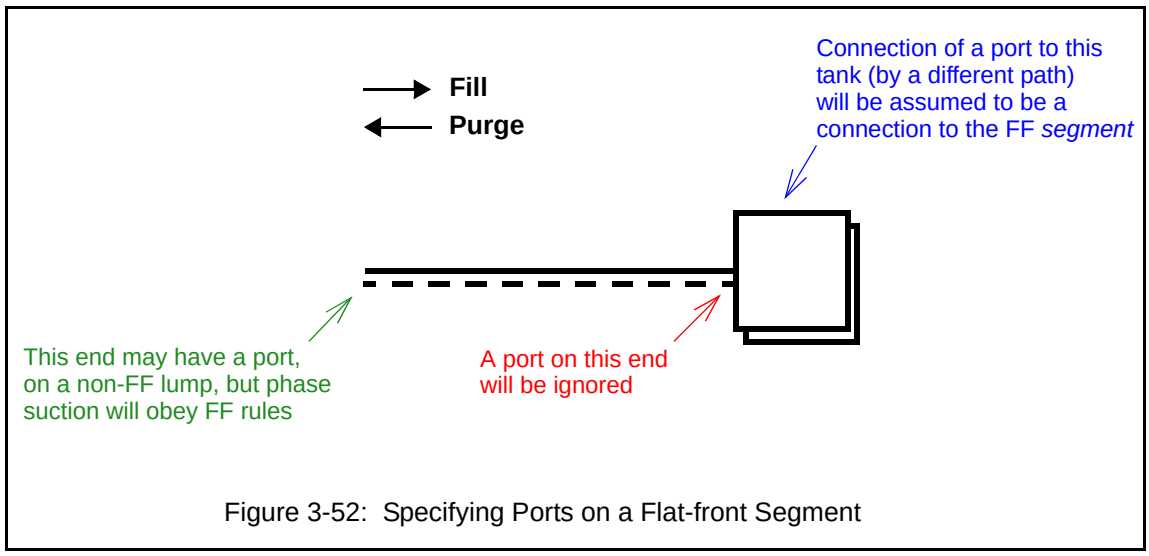


Figure 3-52: Specifying Ports on a Flat-front Segment

A path port cannot interrupt this relationship between a flat-front tube and its associated tank. Therefore, if a port is added *within* such a segment, it will be ignored (red text in the figure).

The other "open" end of the segment (defined upstream end of a FILL segment, or defined downstream end of a PURGE segment), however, can make use of a port (green text in the figure). For example, if a line extracts liquid from the bottom of a vessel, and if this line is best represented by a flat front approximation, then the beginning of an FF=FILL segment or duct macro can have a port defined at its inlet. Similarly, a PURGE segment can terminate at the top of a partially filled vessel. (Refer to the sight glass example presented later in this section.)

These port connections (to the open end of a flat-front segment) are allowed to violate the rule that ports cannot be used to join twinned paths to twinned tanks. This exception is made because in a flat-front model only the primary path is active at any one time.*

Another complication of using twinned tanks at the inlet of an flat-front segment occurs if the "wrong" phase is present: if vapor or gas is present within the liquid tank, or liquid is present within the vapor/gas tank. In this case, the normal operation of the flat-front segment means that vapor is preferentially ingested even if the port is below the liquid level, or liquid is ingested even if the port is above the liquid level. Consider adding a separate path between the vessel and the flat-front segment if this is an issue, such as LOSS connector representing the inlet or exhaust (see also the sight glass example below).

* Do not use DUCT_MODE in other than the FF_ON1_H mode for this case. Otherwise, if the inlet path is allowed to resume slip flow, it will be assumed to connect to both phases of the twinned tank at the port end, and the port will become inactive.

Finally, consider that another path is added to the tank (or twinned pair) that represents the flat front segment, and that this path uses a port to connect to that tank. This “side port” (in blue in Figure 3-52) is assumed not to be attached to the *tank* specifically, but rather to the whole flat-front *segment*. (In PORTTAB output, this situation is designated by “(FF)” placed after the liquid/vapor front location.) In this situation, the coordinate location (CX,CY,CZ) of the tank stays fixed, and the program calculates the liquid/vapor position as being somewhere along the tube. (If the segment is completely full, the location of the tank is used as the liquid/vapor position. If it is completely empty, the location of the lump at the other end of the path is used as the position.)

To explain these options, consider the case of a sight glass added in parallel to a partially filled vessel that is modeled as shown in Figure 3-53.* The same sight glass is shown modeled two ways, as either a FILL line (at the left in that figure) or a PURGE line (at the right).

If a FILL line is used to model the sight glass (shown at the left in Figure 3-53), the bottom-most tube can be connected to the bottom of the vessel using a port (i.e., by defining the coordinates of the i^{th} end to be a point at the bottom of the vessel). This port can either exist at the “open” end of the first FF segment’s path, or a new path can be inserted that flows from a port at the bottom of the vessel to a new junction at the base of the FILL line.

Since the FILL line terminates with a tank at the downstream end (representing the top of the sight glass in this case), this tank can be connected to the top of the vessel again using a port, but a new path is not optional at the top: it *must* be added or else the sight glass would represent a dead end. Suitable paths include a LOSS, LOSS2, STUBE, REDUCER, EXPANDER, or ORIFICE connector, depending on modeling decisions unrelated to this example.

If the FILL line is twinned, this new path would connect to the secondary tank in the top of the sight glass model. If the line is not twinned, STAT=VS should be used. If this connection sounds complicated, there is an easier method: just use a second port at the *inlet* of this new path and let it make all the right connections. In other words, the CXI/CYI/CZI of this new path represents the top of the sight glass,† and its CXJ/CYJ/CZJ is a point at the top of the large vessel. A port can exist on each end of this path.

The PURGE line shown at the right of is nearly identical to an inverted FILL line. (Indeed, a fill line with a receding front is essentially identical to a PURGE line with a negative flow rate.) So there is no need to completely repeat this discussion.

* Normally twinned tanks would be used for the model of the vessel itself, but a single homogeneous tank is acceptable too.

† In this case, it is likely that the BFI of the new path will be exactly zero, and the default STAT (usually ‘NORM’) will remain active due to the hysteresis in the STAT operation of ports (as explained in Section 3.13.4). So specifying an initial condition of STAT=VS is still likely to be a good idea, even in combination with a port.

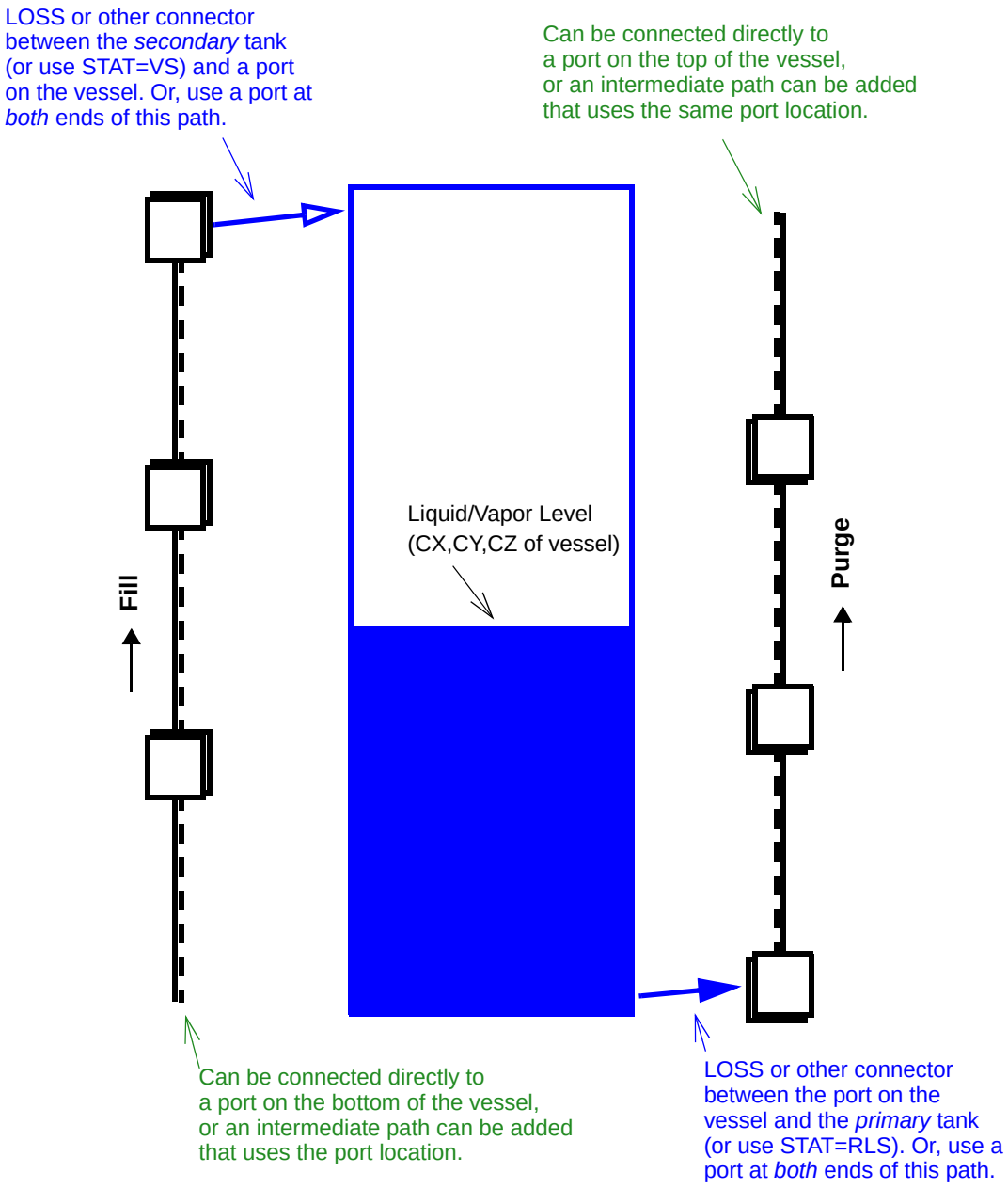


Figure 3-53: Two Ways to Model a Sight Glass: Flat-front and Port Example

In summary, a new path must be added at the bottom of the PURGE line. Either both ends of that path should use ports, or that new path should use STAT=RLS* and/or connect to the primary side of the PURGE line twinned tank. At the top, the "open" ended tube can be connected to the top of the vessel using a port, or a new junction and path can be added, analogous to the choices at the bottom of the FILL line.

* The same caution applies here since the BFJ of this new path might be exactly zero. Setting STAT=RLS as an initial condition is still a good idea.

4 EXECUTION, LOGIC, AND CONTROL

SINDA/FLUINT creates a unique Fortran executable every time it is run. There is therefore no set sequence of analysis operations. While this characteristic purposely provides the user with almost complete control and is in fact largely responsible for the success and longevity of SINDA, it makes documentation difficult and requires an extra investment in learning on the part of the user. The interdependency of the topics in this section will likely require that the new user read it twice.

This section starts by describing the logic blocks available in SINDA/FLUINT: which ones are available, what their intended uses are, and how to generate them. Next, program control constants are introduced, along with the network solution routines they control. Finally, common operations such as restart and parametric operations and submodel control are summarized, along with other miscellaneous topics.

This chapter does not cover additional logic blocks and control parameters that are associated with the Solver, using which the user can “program” SINDA/FLUINT to perform a certain task or seek a certain solution. Nor does it cover the Reliability Engineering module, in which the user can sample a SINDA/FLUINT “run” to estimate the probability of meeting or exceeding design limits. Running the Solver and the Reliability Engineering modules are in many ways similar to setting up a series of SINDA/FLUINT runs such as those described in this chapter. Therefore, in order to avoid confusion, those topics will be addressed separately in Section 5. However, to be consistent with the use of this chapter as a reference, some mention of those advanced topics will be made in this chapter as required.

Introduction—As an introduction to the execution and control of SINDA/FLUINT, and as a means of demonstrating the interdependence of these topics, the brief example introduced in Section 1.5.1 will be reexamined. The table from that section containing the complete input file for that sample problem is reproduced here as Table 4-1 for convenience.

The thermal network and its initial conditions are input in the NODE, CONDUCTOR, and SOURCE data blocks, which have been previously described in Section 2.

The analysis operations to be performed on that network are contained within the OPERATIONS block. This is a logic block that is transformed into the main Fortran routine in the SINDA/FLUINT processor. It is executed once: the program terminates when all the operations specified in OPERATIONS have been completed. In this simple example, only two operations are performed. First, the configuration to be analyzed is stated in the BUILD instruction, which in this case consists of a single thermal submodel. Second, a transient integration of that thermal network is ordered by calling TRANSIENT (a nickname for FWDBCK, a prepackaged network solution routine). The integration starts by default with the initial and boundary conditions stated in the data blocks.

What constitutes the end of the transient integration? What output is desired, and when? In the problem statement, the user desires to end the problem when the temperature at the center of the bar exceeds 200 degrees, and the temperature profile at this time is desired. How can these be expressed to the program?

Table 4-1 Complete Input File for Sample Problem

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM
      OUTPUT  = BAR.OUT
      MODEL = TEST
HEADER NODE DATA, SUB1
     10, 70.0, 0.006               $    record 1
     15, 70.0, 0.006               $    record 2
     20, 70.0, 0.006               $    record 3
     -99, -460., 0.0               $    record 4
HEADER CONDUCTOR DATA, SUB1
     1015, 10, 15, 0.00417         $    record 5
     1520, 15, 20, 0.00417         $    record 6
    -2099, 20, 99, 1.98E-15        $    record 7
HEADER CONTROL DATA, GLOBAL
     TIMEND = 1000.0, OUTPUT = 1.0 $    record 8
HEADER SOURCE DATA, SUB1
     10, 10.0*3.413/60.0           $    record 9
HEADER OPERATIONS
BUILD TEST, SUB1
      CALL TRANSIENT               $    record 10
HEADER OUTPUT CALLS, SUB1
      IF(T15 .GE. 200.0) THEN      $    record 11
        CALL TPRINT('SUB1')        $    record 12
        TIMEND = TIMEN             $    record 13
      ENDIF                        $    record 14
END OF DATA
```

Part of the answer to the above question is supplied by the data provided in the CONTROL DATA block. As that name implies, control “constants” supplied in that block help govern the execution of solution routines such as TRANSIENT. In this particular example, the user supplies the end time of 1000 minutes (purposely oversized, since the exact duration is unknown and in fact is one of the key results expected from this analysis). The user also specifies the interval at which output is to be produced: 1 minute. (The output will be directed to the file specified in OPTIONS DATA.)

Exactly which output operations are to be performed at that interval are supplied in OUTPUT CALLS. This block is another logic block that is transformed into a Fortran routine that will be called at the intervals specified by the control constant OUTPUT. In this particular example, more than just output operations are performed in that block. Every minute, the temperature of the middle node is checked against the limit. No operations are performed (and therefore no output is produced) until that limit is reached. Upon reaching the temperature limit, all nodal temperatures are printed, using the standard output routine TPRINT, and the end of the integration is signaled by setting the problem end time (control constant TIMEND) equal to the current time (TIMEN). TRANSIENT will then cease analysis, returning control to OPERATIONS for further operations, which in this case are absent. The program will therefore stop after printing out the end results.

In summary, the user determines the analysis sequence and other operations (such as output generation) via logic blocks. Analyses are invoked by calling solution routines, which in turn call other logic blocks at either predetermined or user-determined intervals. Control constants regulate both the numerical integration and the frequency of logic block calls, and can themselves be inspected or altered in logic blocks along with network parameters such as nodal temperatures.

More than a program with fixed inputs and outputs, SINDA/FLUINT is a language with which arbitrary thermal/fluid problems can be creatively posed and solved. Although the user must master both the lexicon and the grammar to be completely fluent, learning a few common words and phrases will quickly enable him or her to communicate.

4.1 LOGIC BLOCKS

A logic block is a group of input records (delineated by a header record) that specifies a sequence of operations to be performed on the data input in the data blocks described in Section 2 and Section 3. The general form of a logic block is as follows:

```
HEADER name-of-block, smn
         Statement-1
         Statement-2
              .
              .
              .
         Statement-n
```

where `name-of-block` can be any of the following:

```
OPERATIONS*
VARIABLES 0
VARIABLES 1
VARIABLES 2
FLOGIC 0
FLOGIC 1
FLOGIC 2
OUTPUT CALLS
SUBROUTINES*
```

(Additional logic blocks are defined in Section 5. In particular, blocks called PROCEDURE and RELPROCEDURE are available that obey the same rules as does OPERATIONS.)

Unlike data blocks, whose input formats are dictated by the type of block, all logic blocks are written in a Fortran-like language that is converted (or *translated*) into real Fortran. Almost all aforementioned element parameters (T, G, TL, FR, UA, etc.) can be translated, accessed, and perhaps modified within logic blocks. (See subsections 4.1.3.9 and 4.1.3.10 for a complete list.) Each logic block becomes a separate Fortran subroutine, with the exception of SUBROUTINES, which is intended to contain several additional routines.

The subroutines resulting from the logic blocks fall into separate categories. The OPERATIONS block is translated into subroutine OPER which is the user's main or driver routine. All of the calls to the SINDA solution routines (e.g., TRANSIENT) should be made from this routine (or from PROCEDURE or RELPROCEDURE, to be defined in Section 5). It is also the appropriate place to initialize the values of any program variables that were not assigned values within the data blocks (e.g., NODE DATA).

* Note "smn" is not allowed with these global header names.

The VARIABLES 0 blocks (one per thermal submodel, plus a special global block) translate into a set subroutines called from the subroutine VARBL0. The VARIABLES 1 blocks translate into a set of subroutines called from the subroutine VARBL1, and the same is true for the VARIABLES 2 blocks in the subroutine VARBL2. FLOGIC n blocks translate into routines called from FLOGIn, where n = 0,1 or 2. The set of OUTPUT CALLS blocks translate into subroutines called by subroutine OUTCAL. The VARBLn, FLOGIn, and OUTCAL subroutines are called from the solution routines at predefined points to update time-dependent thermal variables (VARBLO), temperature-dependent thermal variables (VARBL1), fluid variables (FLOGIn, as described below), and to perform program output operations (OUTCAL).

The SUBROUTINES block is provided so that the user can add any number of arbitrarily named subroutines to the program that will be built from the logic blocks and the portions of the SINDA library that they invoke. These subroutines are useful where the same series of operations are required at a number of points in the other logic blocks. For example, the same specialized printout operations may be required in all or part of the individual OUTPUT CALLS blocks. SUBROUTINES can also be used to replace standard or default subroutines with equivalent arguments but altered logical operations.

The logic blocks contain two types of statements: *F-type* (untranslated or verbatim) Fortran statements and *translatable* Fortran statements.* An F-type statement is any valid Fortran statement *that requires no translation operations*. A translatable statement is similar to an F-type statement except that they may contain SINDA/FLUINT variables that require translation into real Fortran before compilation can be performed. These variables can include submodel name prefixes and actual element number suffixes such as: FRED.T10, meaning the temperature of node 10 in thermal submodel FRED. Such SINDA-specific subscripts are converted to the appropriate relative numbers that are used to find the variables in the program data storage. The two types of statements are illustrated below:

Translatable statement (contains SINDA/FLUINT variables requiring translation):

```
        CALL D11MDA(TIMEM,A200,A217,Q200,POD2.XK20)
```

F-type statement (does not require translation):

```
F       CALL D11MDA(TIMEM,A(21),A(38),Q(108),XK(5))
```

The distinction is *not* the presence or absence of parentheses. While F-type statements must use such parentheses while referring to an array, in translatable statements they are optional: A(200) is equivalent to A200. The real difference is that “A200” is a reference to a user array number 200, which is then translated into a cell in a single Fortran array (perhaps “A(21)”) that contains all arrays input in all ARRAY DATA blocks. Normally, the user need not know into which array or cell location a particular variable is translated.

The following discussion of the SINDA logic blocks requires the introduction of several concepts if the prospective user is not familiar with Fortran programming techniques. This is true because the logic blocks are user’s means of specifying the sequence of operations (mathematical and otherwise)

* Users of old SINDA might note that translatable statements are equivalent to “M-type” statements, in which an optional “M” may be placed in column 1 to invoke translation.

to be performed by the computer in order to solve the user's problem, and, as such, they constitute a "computer program." Just as the network data blocks are used to tell the plug-and-grind computer *what* to plug, (the *data*), the logic blocks are used to tell it *how* to grind (the *logic*).

A general understanding of three key concepts is necessary in order to take full advantage of the versatility of the SINDA/FLUINT system. These concepts are (1) flow charts, (2) subroutine usage, and (3) program flow control. These concepts will not be discussed explicitly but they will serve to unify the discussion in the following sections. The discussion will show, by examples, how these concepts relate to the functional and operational aspects of the logic blocks.

After describing the macrocommands available in logic blocks in Section 4.1.1, a functional description of the logic blocks will be given in Section 4.1.2, and application guidelines will be presented in Section 4.1.3. The operational details of actually preparing SINDA/FLUINT logic records will be deferred until Section 4.1.4.

4.1.1 Macroinstructions in Logic Blocks

Macroinstruction records all begin with a key character in Column 1. These records are few in number and powerful in function and should be well understood before attempting to generate SINDA/FLUINT input.

The following is a list of the allowed macroinstructions records in logic blocks. (General macroinstructions are introduced in Section 2.6.)

(B)UILD	Records
(B)UILDF	Records
(S)PELLON	Records
(S)PELLOFF	Records
(D)EFMOD	Records
(F)START	Records
(F)STOP	Records

The use of the BUILD and BUILDF commands are restricted to OPERATIONS (and PROCEDURE and RELPROCEDURE, to be defined in Section 5). The DEFMOD command can only be used in global blocks such as OPERATIONS, PROCEDURE, RELPROCEDURE, and SUBROUTINES. The remainder can be used in any logic block.

4.1.1.1 BUILD and BUILDF Instructions

These records define the currently active SINDA/FLUINT model in terms of a list of submodel names. They appear *only* in OPERATIONS (and PROCEDURE and RELPROCEDURE, to be defined in Section 5). *No submodel will be analyzed or even preprocessed unless it has been built; at least one BUILD or BUILDF instruction must appear in OPERATIONS (or PROCEDURE or RELPROCEDURE) before calls to any solution or output routine.* The format is:

$$\texttt{BUILD mn, } sn_1, sn_2, \ldots sn_i, \ldots sn_n$$

where:

`mn`..... arbitrary configuration name
sn_i..... *thermal* submodel names, 1 to 32 characters.

A short-cut is available *to activate all input thermal submodels*:

```
BUILD mn, ALL
```

and this can be further abbreviated by letting the configuration name be “DEFAULT:”

```
BUILD ALL
```

The BUILD commands are translated into Fortran code. If more than one 132 column record is required for the sequence, continuation characters are placed in column 6. The configuration name must be to the right of column 6. Either a HEADER NODE DATA or (in the case of conductor-only submodels) a HEADER CONDUCTOR DATA record must be present for every submodel name on the BUILD list. The NODE DATA block may be empty in the case of submodels consisting of conductors only.

Any subsequent BUILD operations erase rather than append to the previous BUILD operation. However, any submodel not currently active will retain the status it had when last active. Thus, results are never lost (barring the use of parametric or restart options).

BUILDF instructions operate analogously for *fluid* submodel names. To build both types of submodels into one configuration, use the same configuration name on adjacent BUILD and BUILDF instructions, as depicted in the following example:

```
BUILD CONFIG, THERM1, THERM2
BUILDF CONFIG, FIRST, SECOND,
      + THIRD
```

A short-cut is available *to activate all input fluid submodels*:

```
BUILDF mn, ALL
```

and this can be further abbreviated by letting the configuration name be “DEFAULT:”

```
BUILDF ALL
```

Models are not normally explicitly unbuilt. Once built, they can be unbuilt by another BUILD or BUILDF instruction that does not include that submodel. To unbuild all thermal models (presumably leaving fluid submodels active), use:

```
BUILD NONE
```

Analogously, to unbuild all fluid submodels (presumably leaving thermal submodels active), use:

```
BUILDF NONE
```

4.1.1.2 FSTART and FSTOP Instructions

These instructions direct the processor to copy *verbatim* all logic records included between FSTART and FSTOP without attempting any conversions or translations as described later. This is equivalent to placing an F in column 1 for all statements between the FSTART and the FSTOP commands. This option is most useful in the SUBROUTINES block to prevent any translation from taking place when a Fortran routine is inserted.

Care should be exercised in the use of these instructions in logic blocks, however, since they may prevent desired translations. Significant improvements in SINDA/FLUINT translations over those of older SINDA versions, on the other hand, make it unlikely that the preprocessor will attempt to erroneously translate any valid Fortran expressions.

4.1.1.3 The Spelling Checking Option: SPELLON and SPELLOFF Instructions*

One of the most troublesome errors a SINDA/FLUINT user can make is to misspell a variable name within the logic blocks. Because this misspelling results in a Fortran local variable being inadvertently defined, the error may never be really visible or may turn up much later in the run, usually in an unpredictable and catastrophic fashion.

SINDA/FLUINT therefore offers a spell checking feature, which is active by default. The SINDA/FLUINT preprocessor declares a undefined variable to be one whose name does not match any program control constant, register name, named user constant, numbered user constant, array element or reference, or any of the key program variable names (e.g., T, C, Q, G, etc.). In other words, an undefined variable is one which SINDA/FLUINT does not recognize. *When spell checking is active, all variables within translatable statements are checked* against the list of variable names and key word combinations that are defined in the data blocks. When an undefined variable is identified, an error message will be printed just after the record containing the local variable. For instance, the statement:

```
TEMP = T50
```

will generate a preprocessor error message if TEMP is not declared as a global user constant or as a register. If node 50 did not appear in the default submodel's NODE DATA block, an additional error would result, but such checks are performed even if spell checking is disabled. Note, however, that if spell checking had been inactivated and the user had inadvertently typed "TT50" instead of "T50", *the error would not have been trapped,* and TEMP would have been set to zero. ***For this reason, use of the spell checker is strongly recommended.*** The extra effort spent "declaring" variables as registers or global user constants is well worth the avoidance of such insidious errors.

The difficulty with this feature is that the user can frequently generate valid local variables that are not misspelled. The user may then simply inactivate the spell checker in order to eliminate annoying messages, which is dangerous. The user should instead get in the habit of declaring all of the variable names planned for use in translatable statements in the REGISTER DATA or global USER DATA blocks.† Users familiar with programming languages other than Fortran will recognize this as a firm requirement. Fortran, however, does not enforce this discipline. If these variables are

* Formerly the DEBON and DEBOFF instructions, which are still accepted equivalently.

all defined in the USER DATA, GLOBAL or REGISTER DATA blocks, the spell checking option will only point to misspelled local variables and keyword variables that have not been declared properly or were misspelled.

If the user takes no action, the spell checking feature will be on by default. The SPELLON and SPELLOFF macroinstructions provide local control of spell checking. A SPELLOFF appearing in a logic block will turn off spell checking until the next logic header record or a SPELLON instruction is encountered. A SPELLON instruction will turn on the spell checking feature until a header card or a SPELLOFF record is encountered. These instructions have no arguments.

SPELLOFF in the OPTIONS DATA block turns off spell checking for all logic blocks, although it can be turned back on locally with a SPELLON instruction.

It should be emphasized that the spell checking option does not apply to F-type statements. This is because F-type statements are not examined character by character, but rather they are passed on to the compiler unchanged. In the majority of cases, a translatable statement can be written that is equivalent to an F-type statement. Therefore, it is recommended that translatable statements be used as much as possible in large, complex models that are likely to benefit from this spell checking feature.

4.1.1.4 DEFMOD Instruction

This instruction sets or resets the current default submodel name. It can be used only in OPERATIONS, PROCEDURE, RELPROCEDURE, and SUBROUTINES where the default submodel name is 'GLOBAL'. The format is:

```
DEFMOD smn
```

The use of this option eliminates the need for the repetitive input of a model name. For example, "MODL.T100" could be input as "T100" after a DEFMOD MODL record.

The default submodel name may be either a thermal or fluid submodel, but only one submodel name may be defaulted at any time.

† Although both REGISTER DATA and USER DATA, GLOBAL may seem at first to serve the same purpose with respect to declaring valid variables to the spell checker, it is recommended that only USER DATA, GLOBAL be used for this purpose. REGISTER DATA has other, more comprehensive, purposes.

4.1.2 Functional Descriptions of Logic Blocks

After the preprocessor has processed the network data blocks and translated the logic blocks, the data is placed in the minimum required disk storage (thus releasing core memory for other uses) and the resulting subroutines are passed to the system FORTRAN compiler. Following compilation, the resulting program is loaded into core and executed as shown in the flow chart in Figure 4-1.

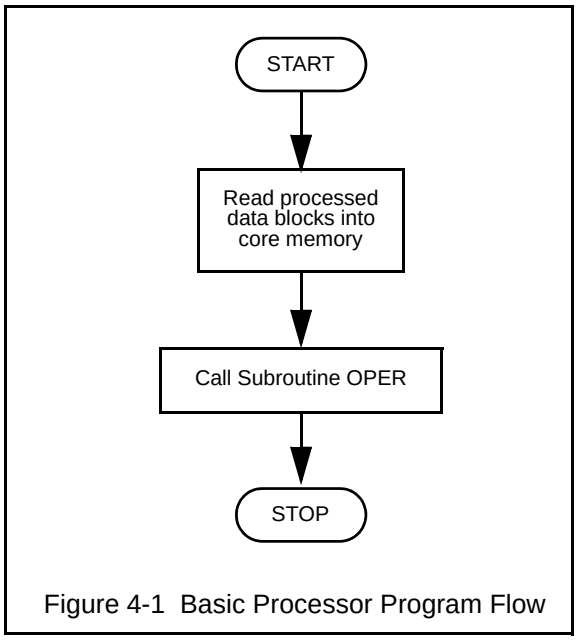


Figure 4-1 Basic Processor Program Flow

This simple flow chart reveals and implies several things:

1) The actions depicted in the flow chart take the place within the framework of a “main program” (the processor). Since this “main program” is fabricated entirely by the preprocessor, the actions therein occur *automatically* from the standpoint of the user (i.e., the user has no control over these actions).

2) The first action is the reading of the preprocessed data into core memory.

3) The final action is a transfer of control to subroutine OPER, which was formed from the user’s OPERATIONS block.

4) Clearly, the first opportunity for the user to specify operations which will lead to the solution of his or her engineering problem occurs in the OPERATIONS block. That is, if the user includes no operations in this block, then subroutine OPER will be empty (i.e., it will indicate that nothing is to be performed) and, for all intents and purposes, the resulting program will do nothing. Conversely, exactly and only those operations included by the user in the OPERATIONS block will appear and will be performed in subroutine OPER when it is called.

5) The basic flow chart includes no explicit reference to subroutines VARBLO, VARBL1, VARBL2, FLOGI0, FLOGI1, FLOGI2, or OUTCAL. These are called from solution routines, which are in turn normally called from OPER.

No general flow chart of the OPERATIONS block can be presented here because everything in the block must be placed there by the user and the sequence of operations will always be specific to a particular problem. However, the flow chart for a typical OPERATIONS block is shown, for illustrative purposes, in Figure 4-2. The last line in each block of this flow chart shows the SINDA operation which will accomplish the action described there in words. The various subroutines referenced by name (i.e., D1DEG1, TRANSIENT and TPRINT) exist in prewritten, “canned” form in the SINDA/FLUINT library, and are described in Section 7.

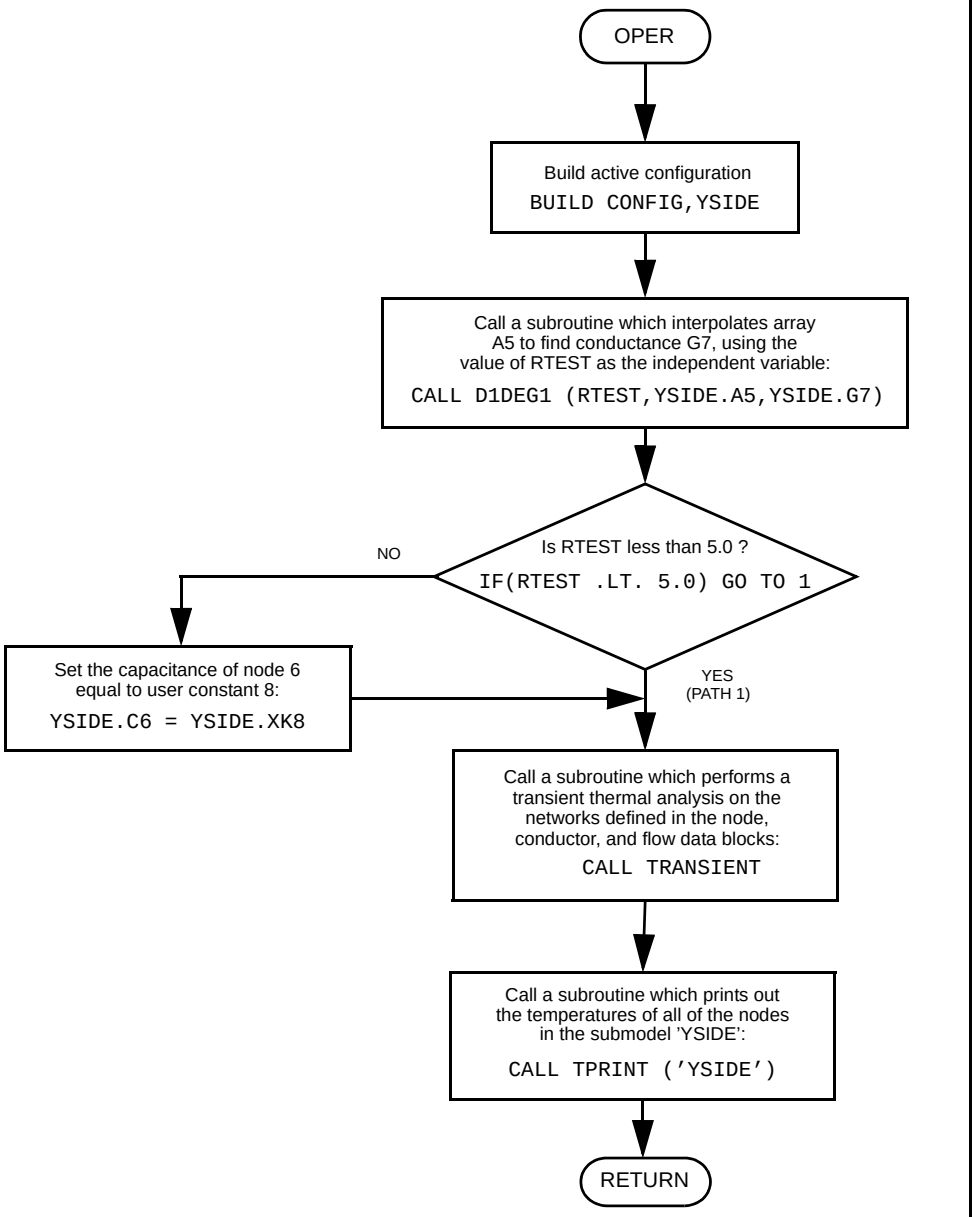


Figure 4-2 Sample Flow Chart for an OPERATIONS Block

It will be noticed that this flow chart still makes no explicit mention of subroutines VARBL0, VARBL1, VARBL2, FLOGI0, FLOGI1, FLOGI2, or OUTCAL. This will normally be true for any OPERATIONS block flow chart because these subroutines are called automatically from within the prewritten network solution subroutines (e.g., TRANSIENT); they are not, in general, called directly by the user. For example, OPER might call TRANSIENT; TRANSIENT, at certain points in the sequence of heat transfer and fluid flow calculations, calls VARBL0, VARBL1, VARBL2, FLOGI0, FLOGI1, FLOGI2, and OUTCAL for each active submodel; each of these routines in turn, again as directed by the user, calls upon various library subroutines to perform operations which are unique to the problem at hand. Thus, the operations included in the VARIABLES n, FLOGIC n, and OUT-

PUT CALLS blocks are used to customize the canned network solution routines so that they can accommodate the peculiarities that are specific to a given user's current problem. Figure 4-3 will aid the user in visualizing what has just been described—namely, that the operations entered by the user in the VARIABLES n, FLOGIC n, and OUTPUT CALLS blocks serves as "customized additives" to the prewritten operations contained in the thermal and fluid network solution subroutines.

In order to illustrate the way multiple logic blocks are incorporated into the program when the problem involves more than one submodel, a flow chart of subroutine VARBL0 for a multiple submodel problem is presented in Figure 4-4. For this illustration, several concepts involved in subroutine execution are introduced. In order to execute subroutine VARBL0, the following statement must appear in a "calling routine", e.g., TRANSIENT:

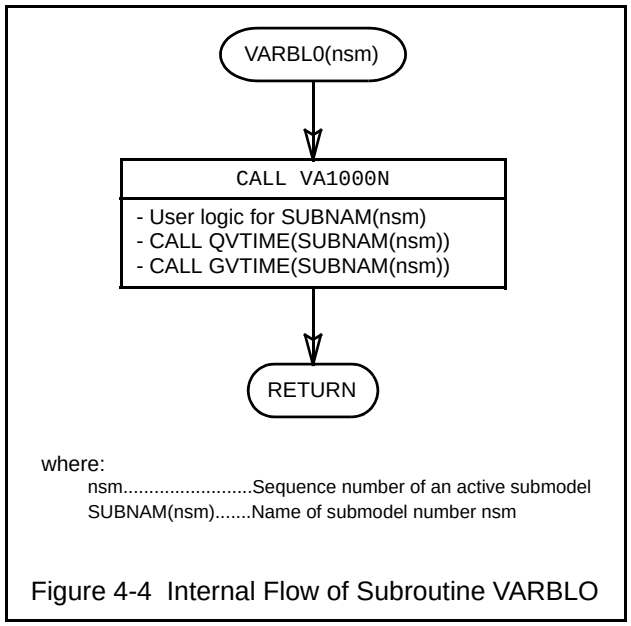


Figure 4-4 Internal Flow of Subroutine VARBLO

```
CALL VARBL0 (NSM)
```

where `NSM` is the internal number of the current submodel (as found by the function MODTRN—or for fluid submodels, MFLTRN). This statement is known as the "calling sequence" for VARBL0. NSM is an *argument* of VARBL0 and may be a literal integer number or a variable name that points to an integer number in core memory.

In reference to Figure 4-5, note that a number of calls to subroutine VARBL0 are made, one for each active submodel. Each call to VARBL0 executes a specific VARIABLES 0 block (specified by the submodel name appearing on its header record) followed by calls to subroutines QVTIME and GVTIME, again for a specific submodel named in the calling sequences (see Figure 4-4). This example also shows that submodels are identified within SINDA/FLUINT by sequence numbers from 1 to n for an n-submodel problem.

Subroutines QVTIME and GVTIME perform the interpolations necessary to update time-dependent Q-source values (QVTIME) and conductor values (GVTIME) defined in the conductor and source data blocks using the time-dependency options (e.g., TVS, CYC, PER). If no such options were used, these calls will exist, but do nothing. The calls are placed at the end of the user's logic by the preprocessor, assuming that it is acceptable to perform these updates after the user's logic is executed.

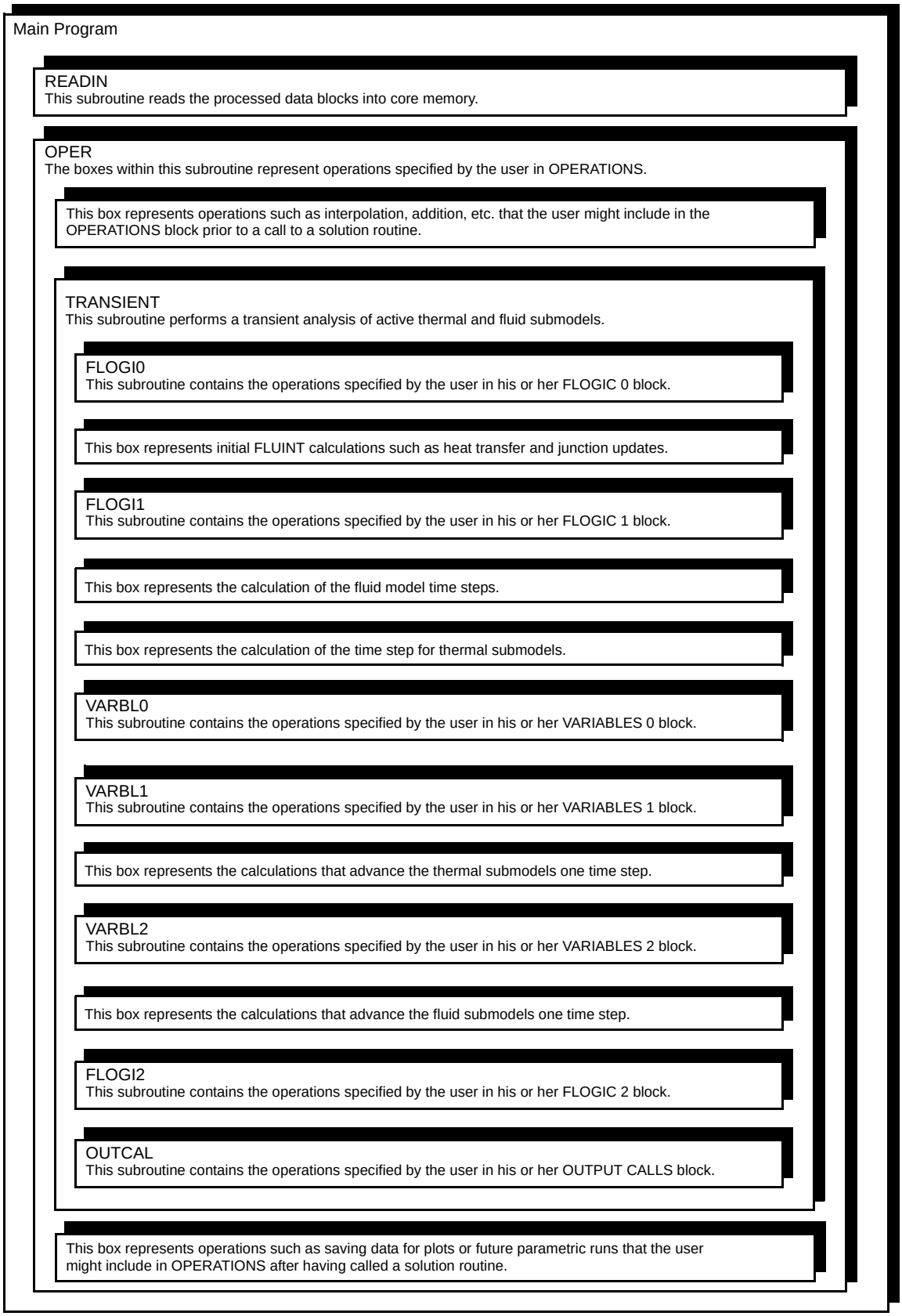


Figure 4-3 Nested Structure of the Logic Blocks

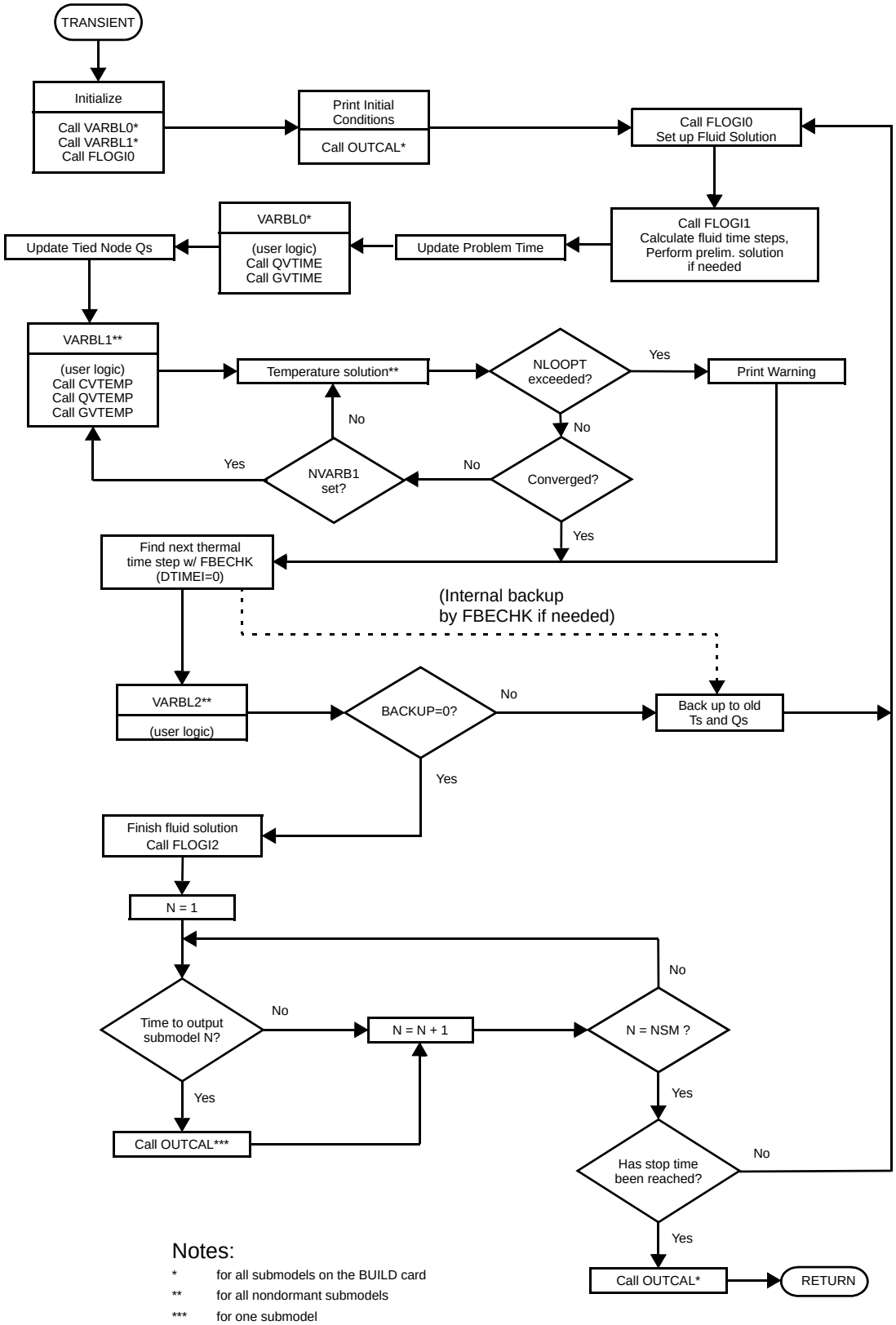


Figure 4-5 Flow Chart of Network Solution Routine TRANSIENT (“FWDBCK”)

If this sequencing is not acceptable, or if the user desires to modify the resulting values after the automated calculations have been performed, then the user may insert one or both of these calls at the beginning or anywhere within VARIABLES 0 logic. This is done with specialized translatable statements:

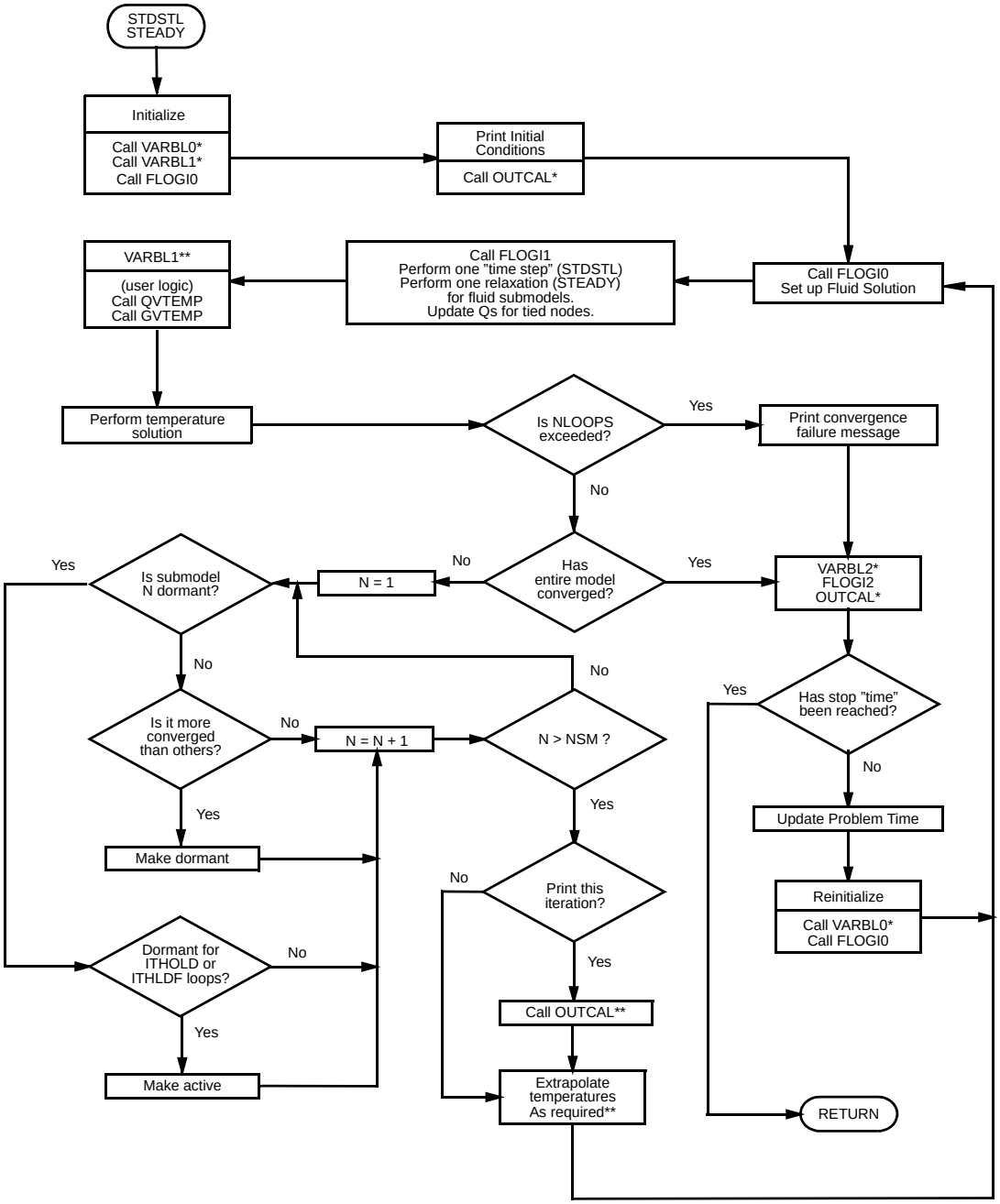
```
        CALL QVTIME('smn')
or
        CALL GVTIME('smn')
```

where `smn` must agree with the submodel name argument appearing on the preceding header record. When the preprocessor encounters such statements, it will refrain from inserting an identical call at the end of the user's logic.

With some specific differences, Figure 4-4 also represents the internal flow of subroutines VARBL1, VARBL2 and OUTCAL. The differences are that in VARBL1, QVTEMP, GVTEMP and CVTEMP replace the QVTIME and GVTIME calls. FLOGI0, FLOGI1, FLOGI2, VARBL2, and OUTCAL have no preprocessor-supplied calls.

The flow chart in Figure 4-5 details the sequence of operations contained in network solution routine, TRANSIENT. (Figure 4-6 and Figure 4-7 perform the same function for FORWRD and STDSTL/STEADY.) Careful examination of this flow chart will reveal the extensive versatility which may be built into the solution routines by the creative use of the VARIABLES n, FLOGIC n, and OUTPUT CALLS blocks. The VARIABLES 0 and FLOGIC 0 blocks allow the user to interject operations which will be performed after the problem time has been incremented but before the actual heat transfer equations are integrated. The VARIABLES 1 block is used similar to VARIABLES 0, except that thermal time steps have been predicted at this point in the solution. In steady-state routines, the differences are greater—the VARIABLES 0 block is executed once per analysis, while the VARIABLES 1 block is executed once per iteration. Both FLOGIC 0 and FLOGIC 1 blocks are executed each iteration in steady-states and each time step in transients. The FLOGIC 1 block, which should be used only by advanced users or as instructed by the documentation for various simulation routines, represents a chance to modify program calculations before a solution is performed. The VARIABLES 2 and FLOGIC 2 blocks allows the user to interject operations which will be performed after the equations are integrated for the current time step, and the OUTPUT CALLS blocks allow the user to interject operations which will be performed only when the problem time has progressed to a multiple of some specified interval (i.e., the OUTPUT control constant).

The flow chart also reveals how the control constants can be used to achieve program flow control within the network solution routines. For example, the control constant OUTPUT (actually an array of control constants, one per submodel) is used to specify the time interval at which subroutine OUTCAL will be performed. Other control constants provide checks on the time step used, temperature change calculated per iteration, etc., as discussed in Section 4.3. Suffice it to say that by examining and modifying these constants in the VARIABLES n, FLOGIC n, and OUTPUT CALLS blocks, users are able to effect complete control over the flow of the network solution routines, in addition to interjecting their own operations into the prewritten sequence of calculations contained therein.



Notes:

* for all submodels on the BUILD card

** for all nondormant submodels if ITERXT .GE. 3

Figure 4-7 Flow Chart of Network Solution Routines STDSTL and STEADY (“FASTIC”)

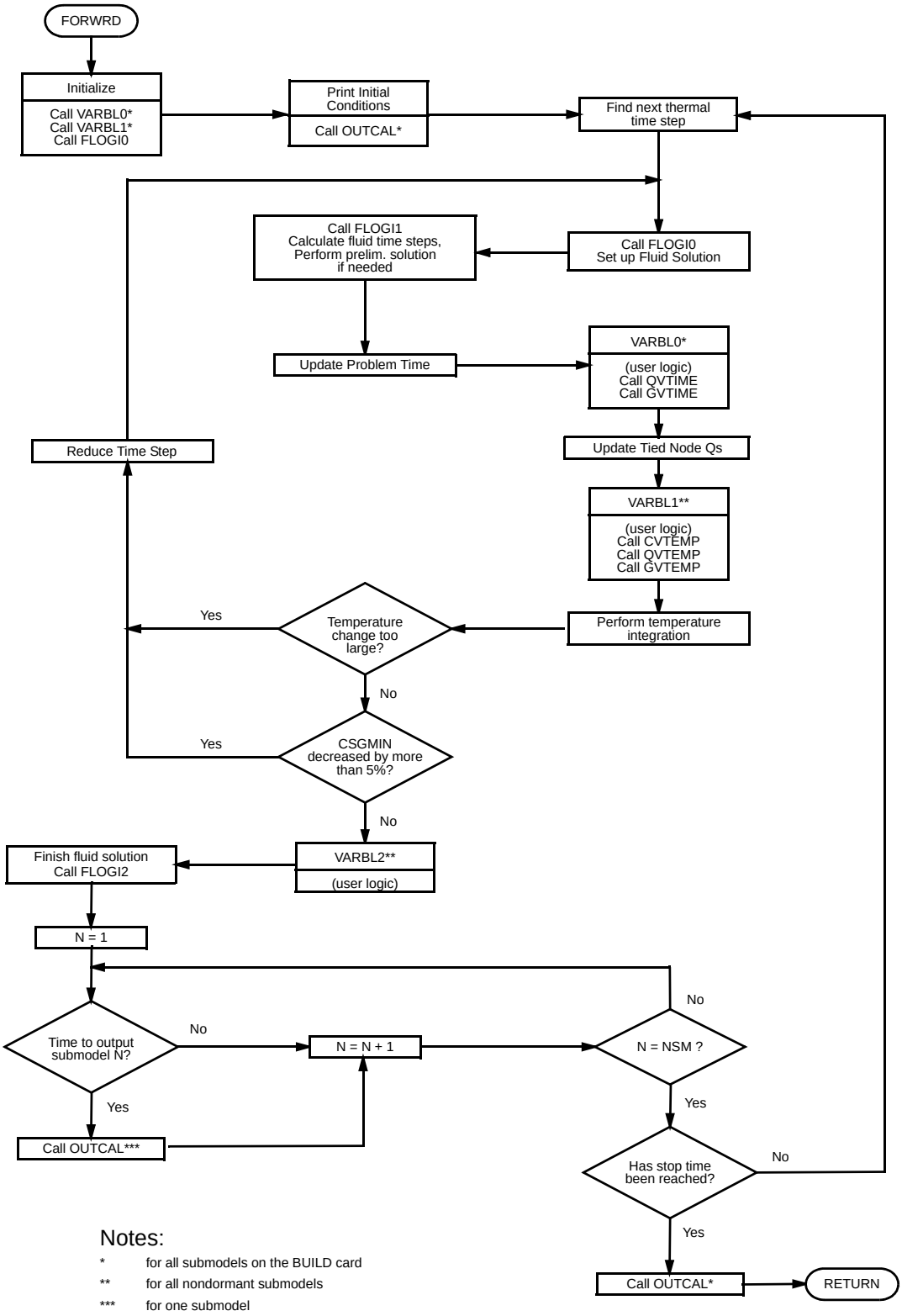


Figure 4-6 Flow Chart of Network Solution Routine FORWRD

The sample flow chart for the OPERATIONS and VARIABLES 0 blocks (Figure 4-2 and Figure 4-4) represent the result of applying three basic concepts (i.e., flow chart development, subroutine usage, and program flow control) to a specific problem. These concepts are equally applicable to the VARIABLES n, FLOGIC n, and OUTPUT CALLS blocks. The structure of the flow chart indicates the exact sequence of operations which the user has established as being necessary for the solution of his or her problem. The discrete operations detailed in each box are mechanized through the usage of prewritten subroutines from the library, and the Fortran "IF" statement (in the diamond) is used to accomplish program flow control (i.e., the selection of alternate sequences of operations). These three concepts will generally be applied by the user to each individual problem which he or she desires to solve. Various guidelines for structuring the flow of an OPERATIONS block will be presented in Section 4.1.3, but the user must exercise a certain degree of judgment in applying them to his or her problem. In addition, the user should recognize that the SINDA/FLUINT system provides the capability and freedom to innovate in those cases where general guidelines cannot be applied. Similarly, the actions and generalized calling sequences of the subroutines in the SINDA/FLUINT library will be given in Section 7, but it is up to the user to select the routines which are appropriate for the problem at hand, as well as to supply them with appropriate actual arguments. In the same fashion, Section 4.4 describes the available methods for achieving program flow control, but the responsibility for selecting and correctly applying these methods rests with the user.

4.1.3 Applications Guidelines

This section provides an understanding of the different types of logic blocks in terms of their typical usage. It also provides reference material on the SINDA/FLUINT variables that are available within logic blocks.

4.1.3.1 Real and Integer Variable Types

Unless otherwise stated, all real (floating point) and integer variables are stored in 8 byte (64 bit) words, as if REAL*8 (or DOUBLE PRECISION) and INTEGER*8 declarations had been used globally.*

This means there is no distinction between single and double precision, and that declarations such as "DOUBLE PRECISION" are unnecessary, as are intrinsic functions such as SNGL() or DBLE().† However, there is no harm in their continued usage. It also does not matter whether a variable has been set to 1.11E4 or 1.11D4, for example: both values are treated as identical.

4.1.3.2 OPERATIONS Block

When developing the OPERATIONS block for a problem, it is sometimes convenient to partition the sequence of operations into two segments. One segment will consist of those operations performed prior to calling a network solution routine, and the other segment will consist of those

* The following compiler flags are used: /integer_size:64 /real_size:64

† Since earlier versions distinguished between single and double precision variables, to be compatible with models created for those earlier versions, "DP:" registers are still collected together as a separate set, and some double precision versions of user-callable routines are still available even though they are now identical to the single precision versions.

operations performed after calling a solution routine. The first segment will contain everything necessary to set up and initialize the problem, and the latter segment will contain everything necessary to wrap up, print out, summarize, and/or save a problem which has been solved.

There is no reason, of course, why the OPERATIONS cannot consist of a series of these initialize-solve-wrap up sequences, and parametric options have been provided to aid in such usage (see Section 4.7). Unless a specific action has been taken to recall earlier results, all operations act sequentially on the active networks such that the final conditions of the first operation are the initial conditions for subsequent solutions. *Recall that the first segment of an OPERATIONS block must contain a BUILD and/or a BUILDF instruction (if those instructions are not supplied in PROCEDURE and RELPROCEDURE, blocks to be defined in Section 5),* and that such instructions may be repeated as needed throughout the OPERATIONS block. Any submodel not built will become inactive, and will return to its last status upon reactivation by another BUILD or BUILDF command.

The following operations are among those which would be appropriate for inclusion in the pre-solution segment:

1) Compute scale factors and other constant values which were inconvenient to compute by hand.

2) Scale data to consistent units.

3) Initialize required control constants if necessary.

4) Open any user files to be used during the run.

In performing such set-up and initialization operations, the user should recognize that, while all memory locations initialized by the data blocks (i.e., temperature locations, conductance locations, source locations, flow data locations, etc.) are accessible in the OPERATIONS blocks, not all of these locations contain meaningful values prior to calling a network solution routine. The following list describes the status of all accessible memory locations upon entering subroutine OPER:

1) All temperature locations contain the “initial temperature” of their corresponding node, as defined on the NODE DATA records.

2) Capacitance locations corresponding to nodes which were defined using the standard (3 blanks) or GEN options contain the capacitance values defined on the input records.

3) *Capacitance locations corresponding to nodes defined by the automated, variable capacitance options (SIV, DIV, etc.) contain no meaningful values.* These locations will be automatically loaded with the current capacitance of their respective nodes (as computed according to the options specified in the NODE DATA block) during each iteration of the heat transfer equations, and hence, they contain meaningful values only after a network solution routine has been started.

4) Source locations defined by the standard and GEN options will be initialized to the values on the SOURCE DATA records.

5) *Source locations defined by the temperature dependent and time dependent options (SIV, SIT, etc.) will contain meaningless data* for the reasons similar to those given in Item 3.

6) Conductance locations corresponding to conductors which were defined with the standard (3 blanks), or GEN options contain the conductance values specified in the input.

7) *Conductance locations corresponding to conductors defined by the automated variable conductance options (SIV, DIV, etc.), contain meaningless values* for reasons similar to those given in Item 3.

8) All registers contain the values resulting from their initialization in REGISTER DATA.

9) All user constants locations contain the values assigned in the USER DATA.

10) Control constants mentioned in the CONTROL DATA blocks contain the values assigned therein or defaulted by the preprocessor; all others contain zero.

11) All array locations contain the values assigned in the ARRAY DATA blocks (locations reserved with the SPACE option contain zeros).

12) All fluid variable locations are filled with the exceptions of QDOT for lumps; QTIE for all ties and UA more some ties; GK, HK, EI, EJ, and DK for all connectors; AC, FC, and FPOW for all tubes and STUBE connectors, and FD, FG, and AM for those that are twinned.

While it is impossible to directly scale the value of a parameter which falls in the category of Items 3, 5 or 7 above, these parameters may be altered indirectly by modifying the value of their associated multiplying factors. This approach may be effected only if the multiplying factors are input as numbered user constant references (instead of floating point data values or expressions).

An operational description of each of the network solution routines is given in Section 4.2 of this manual.

The following operations are among those which would be appropriate for inclusion in the post-solution segment of the OPERATIONS block:

1) Perform summary calculations.

2) Print final values of interest in special formats.

3) Call for plots of data accumulated during the solution of the network.

4) Save any values which might be needed for later runs.

When a network solution routine has finished its calculations, it returns to the next operation in sequence in the OPERATIONS block. At this point, all data locations (temperature, capacitances, control constants, etc.) contain the current values of their respective parameters, which may be used as initial conditions for subsequent solution routines.

4.1.3.3 VARIABLES 0 Block

The following operations are appropriate for inclusion in the VARIABLES 0 Block:

1) Update the current values of time-dependent quantities.

2) Examine the estimated time step about to be used and modify if desired.

3) Save current values of time-dependent boundary conditions and conductances.

4) Compare current time-dependent boundary conditions and parameters with those of the previous time step.

5) Modify the time step and other control constants as desired for the impending time increment.

Upon entering VARBLO, DTIMEU (the time step used by the last pass through the temperature update algorithm) is available. Problem times TIMEN and TIMEM have also been updated, so the necessary initialization has been done for updating time-dependent variables and boundary conditions. If the user entered no VARIABLES 0 blocks, these updates will be performed by preprocessor supplied calls to subroutines QVTIME and GVTIME (see Figure 4-4). If desired, the user can insert logic statements before and/or after the calls to QVTIME and GVTIME. Thus it is possible to save time-dependent variables in user arrays at the beginning of VARBL0, then compare them with the same variables after the QVTIME & GVTIME calls. The user may also examine parameter vs. time arrays that are known to contain fast transients for the interval TIMEM minus TIMEO in order to be certain that significant changes in boundary conditions are not being lost due to an over-long time step. If this is happening, the user can change DTIMEU, TIMEN and TIMEM, then loop back to the beginning of VARBLO.

Optionally, a VARIABLES 0, GLOBAL block may be specified. This special block is always called independent of the BUILD configuration, and it is called after all the VARIABLES 0 blocks for all currently built submodels have been called. The submodel prefix for variables such as T and A is not optional for this block, though the DEFMOD command is available as an input convenience.

4.1.3.4 VARIABLES 1 Block

The following operations are among those which would be appropriate for inclusion in the VARIABLES 1 block:

1) Update the values of temperature varying quantities (i.e., capacitances, conductances, boundary temperatures, heat rates, etc.)

2) Modify control constants relevant to the network solution routine in progress.

Upon entering VARBL1, for the first iterative pass through the temperature solution algorithm, all variables will be just as they left VARBL0. VARBL1 is called before each iterative pass through the temperature solution loop in STDSTL (or STEADY) and once before each time step in FORWRD or TRANSIENT but after the time step itself has been predicted. (If fluid submodels are active, it is impossible to precisely know a time step size before the actual solution—the available time step is therefore only an estimate.) In addition, if the control constant NVARB1 is set to 1 for a submodel, VARBL1 will be called each iteration in TRANSIENT for that submodel within a time step.

The VARIABLES 0 and 1 blocks are thus the place for the user to enter any logic statements that use the temperatures that are correct at the end of the previous time step, as opposed to temperatures at the end of an iteration, which are merely intermediate values encountered in the iterative solution. The temperatures are accessed with the usual Tn or T(n+i) syntax.

Unless instructed otherwise, the preprocessor supplies calls to CVTEMP, QVTEMP and GVTEMP at the end of VARBL1. These calls update the values of the capacitances, sources and conductances defined, in the preceding data blocks, using the automated variable capacitance source and conductance options. If users desire, they may insert their own calls to CVTEMP, QVTEMP or GVTEMP in one or more VARIABLES 1 data blocks. Using CVTEMP as an example, these calls are made as follows:

```
CALL CVTEMP ('smn')
```

where `smn` is the submodel name appearing on the HEADER VARIABLES 1 record. If the preprocessor encounters such calls in a VARIABLES 1 block, they will not be supplied at the end of the block.

Making CVTEMP, QVTEMP and GVTEMP calls available to the user adds considerable flexibility to VARIABLES 1 operation. For example, suppose the user has the following entry in the NODE DATA block:

```
SIM 25,3,5,70.0,A9,XK5
```

This record generates nodes 25, 30 & 35, all with the same temperature-dependent capacitance. If the user desired to increase the value of one of these capacitances in VARBL1, the following logic would do this:

```
CALL CVTEMP ('smn')
C30 = 1.1*C30
```

Without this option the user could change the capacitances only indirectly, by changing XK5, which would affect C25 and C35 as well as C30.

Since a location is provided in the Q source array for each diffusion and arithmetic node, there is nothing that forces the use of the SOURCE DATA block at all. Sources may be computed and entered into the Q-locations entirely within OPERATIONS, VARIABLES 1 and/or VARIABLES 0 blocks.

Similarly, the user may find that there is no way to define a capacitance, source or conductance using anything other than special logic not provided in any of the regular input options. In this case, the special logic appears in a VARIABLES 0 or VARIABLES 1 block depending upon whether time or temperature is the appropriate independent variable. The user must keep in mind that the calls to QVTIME, GVTIME, CVTEMP, QVTEMP and GVTEMP must precede this logic if any of the automated data entry options were used for the variable to be defined. *Otherwise the results of this user-supplied logic will be overwritten at the end of VARBL0 or VARBL1*. It should be emphasized that a user-supplied call to an xVTIME or xVTEMP routine affects only a single submodel rather than all thermal submodels.

It can be seen by examining Figure 4-5, Figure 4-6 and Figure 4-7 that the relationship between VARBL0 and VARBL1 is different depending on the temperature solution routine. For explicit forward differencing (FORWRD), VARBL0 and VARBL1 could be combined into a single routine. In forward-backward differencing (TRANSIENT or “FWDBCK”), the only difference between

VARBL0 and VARBL1 is that the time step prediction is current in VARBL1.* The difference is greater for steady state solutions because VARBL0 is called once only, prior to beginning temperature iteration. This allows the user to initialize time-dependent program variables at some artificial TIME0 before beginning the steady state solution. VARBL1 is called once per iteration in steady state routines. If fluid submodels are active in any solution routine, then nodal source terms for tied nodes are updated between the VARBL0 and VARBL1 calls.

Optionally, a VARIABLES 1, GLOBAL block may be specified. This special block is always called independent of the BUILD configuration, and it is called after all the VARIABLES 1 blocks for all currently built submodels have been called. The submodel prefix for variables such as T and A is not optional for this block, though the DEFMOD command is available as an input convenience.

The instructions contained within the HEADER VARIABLES 1, GLOBAL block are executed once per TRANSIENT (aka. “FWDBCK”) time step unless NVARB1=1 for *all* thermal submodels (i.e., unless NVARBL1=1 has been set in HEADER CONTROL DATA, GLOBAL).

One usage of the global block is to avoid any conflicts updating Q’s or T’s (e.g., HEATPIPE or TEC calls) by putting those updates in global VARIABLES 1 block.

4.1.3.5 VARIABLES 2 Block

The following operations are appropriate for the VARIABLES 2 data blocks:

1) Examine changes in parameters that occurred during the last time step and set control constant BACKUP accordingly.

2) Accumulate net heat rates

3) Revise control constants

4) Predictor/corrector logic

5. Change-of-state calculations

6. Set new initial conditions for steady-state calculations

The following list describes the status of the thermal network parameters when subroutine VARBL2 is entered:

1. TEMPERATURES — Each location in the T-array contains the temperature of its corresponding node at the end of the current time step or steady-state solution.

2. CAPACITANCES — Each capacitance location contains the value of the capacitance used during the most recent iteration.

3. CONDUCTANCES — Each conductance location contains the value of the conductance used during the most recent iteration.

4. SOURCES — Each source location contains the value of the Q-source values used during the most recent iteration.

* However, if NVARB1 is set to unity, VARIABLES 1 is called each iteration within a time step in TRANSIENT, in a manner more analogous to the steady state routines.

The flow charts for the transient network solution routines in Figure 4-5 and Figure 4-6 indicate that the current iteration will be accepted as correct if the user does not set control constant BACKUP = 1.0 in VARBL2. Thus, the VARIABLES 2 blocks give the user the opportunity to examine the current iteration and to decide whether or not to proceed. This means that if BACKUP is set = 1.0 in any of the VARIABLES 2 blocks that correspond to submodels *not currently in the boundary state*, time will be set back for all of the submodels that participated in the last temperature update pass. Other parameters might also be adjusted in VARIABLES 2 according to some predictor/ corrector scheme. Since VARBL2 is called only once for each time increment, the user can force each time step to be repeated, using the first pass for prediction and the second for correction. If fluid submodels are active, then they are not solved until all thermal back-ups have been completed, and the time step cannot be prescribed completely by DTIMEI.

If the cumulative heat flow in and out of certain nodes is pertinent to the problem, (e.g., for phase change calculations) VARBL2 is the appropriate place for this logic.

Program flow for steady state (Figure 4-7) is similar to the transient routines. VARBL2 is called only once after a steady-state solution has been found, rather than at each iteration approaching that solution. After calling VARBL2, TIMEN for each submodel is advanced DTIMES time units if that option is used. If the problem end time, TIMEND, has not been reached, a new steady-state solution is sought. Hence, the VARIABLES 2 data blocks may be used to compute the initial conditions applicable to the next solution. The user must remember, however, that the next solution will begin with a call to VARBL0, which may contain inappropriate initialization statements. These may be bypassed, on a submodel basis if desired, in VARBL0 by appropriate tests on the values of TIMEN, NSOL, or LOOPCT.

Optionally, a VARIABLES 2, GLOBAL block may be specified. This special block is always called independent of the BUILD configuration, and it is called after all the VARIABLES 2 blocks for all currently built submodels have been called. The submodel prefix for variables such as T and A is not optional for this block, though the DEFMOD command is available as an input convenience.

4.1.3.6 FLOGIC Blocks

FLOGIC 0, FLOGIC 1, and FLOGIC 2 blocks are used similarly to VARIABLES n blocks for the fluid submodel solution. The FLOGIC 0/1/2 sequence represents opportunities to respectively initialize/tailor/wrap-up the fluid solution. While both thermal and fluid variables are available in both FLOGIC and VARIABLES blocks, *as a general rule the user should only alter fluid variables in FLOGIC blocks and only thermal variables in VARIABLES blocks*. Furthermore, some FLUINT variables (PL, TL, XL, HL, DL, VOL, XGx, XFx, PPGx, PVFx, and FR) should not be changed directly in these blocks. (See utility routines CHGLMP and CHGVOL for controlled changes in lump states, Section 7.11.1.1.) Note that, unlike VARIABLES n blocks, there are no default operations in FLOGIC n blocks. Figure 3-3 illustrates the appropriate block in which to update, inspect, or modify tube and STUBE parameters (e.g., AC, TLEN, etc.).

Optionally, GLOBAL-level FLOGIC blocks may be specified (e.g., FLOGIC 0, GLOBAL). These three special blocks are always called independent of the BUILDF configuration, and they is called after all the corresponding submodel-level blocks for all currently built fluid submodels have been called. The submodel prefix for variables such as PL and FR is not optional for this block, though the DEFMOD command is available as an input convenience.

FLOGIC 0 Block — This logic block represents an opportunity to initialize a FLUINT solution before a subsequent solution step. FLOGI0 is called at the beginning of each iteration in STDSTL or STEADY (unlike its analog VARBL0), and at the beginning of each new time step solution in FORWRD and TRANSIENT. Appropriate operations in this block can include:

1) Preparing for the heat transfer, pressure drop, and component model calculations.

2) Performing any transient control logic, such as updating valve or loss elements.

3) Make any calls to CHGLMP, HLDLMP/RELLMP, CHGSUC, CHGLSTAT, etc.

Upon entering FLOGI0, lump states and path flow rates have either been initialized or exist at the state resulting from a previous run or time step solution. Time steps for the fluid solution have not been calculated. Values that are appropriate to change at this point include QL, VDOT, COMP, CX, CY, CZ, IPDC, WRF, UPF, AC, FC, FPOW, HC, AM, FG, FD, FK, AF, AFTH, AFI, AFJ, DH, TLEN, TPF, SMFR, SVFR, SPD, GPMP, DGDH, EFFP, TORQ, RC, CFC, XVH, XVL, GK, HK, EI, EJ, DK, DUPI, DUPJ, UA, DUPN, DUPL, FKH, FKL, PSET, RLAG, RCAP, RBP, DPAB, DPDV, VZRO, VSUB, VHI, VLO, DUPA, DUPB, GF, DUPC, DUPD, AORI, CDIS, MCH, HCH, MODA, MODW, MODO, ROTR, VAI, VAJ, RVR, VXI, VXJ, CXI, CYI, CZI, CXJ, CYJ, CZJ, RLAM, RTURB, and any control constants.

FLOGIC 1 Block — This logic block represents an opportunity for advanced users to tailor a FLUINT solution after most calculations have been made in preparation for a solution step, but before a solution step has actually been taken. FLOGI1 is called at least once each iteration for steady-state solutions and at least once each time step for transient solutions. Appropriate operations in this block can include preparing or tailoring any heat transfer, pressure drop, and device and component model calculations. Note that *this is **not** the correct place to make major changes* or to call CHGVOL, CHGLMP, CHGLMP_MIX, HTRLMP, HLDLMP, HLDTANKS, or RELLMP. Unlike VARIABLES 1, *FLOGIC 1 should not be used as the principal logic block.*

Upon entering FLOGI1, all operations prior to the time step prediction and the subsequent solution have been performed. Changes to heat transfer ties and connector device and component model parameters will be ignored until the next solution step. Changes to lump states or similar changes can cause errors. However, except for STEADY runs this is the appropriate place to change tank (but not junction) QL, COMP, and VDOT values, connector GK, HK, EI, EJ, and DK values (which have now been calculated for all connectors), iface VA, VB, PA, PB, and FPV values (which have now been calculated for all ifaces), and any tube parameters other than those relating to friction options. Time steps for the fluid solution have not yet been calculated, and cannot be completely known before a solution has occurred.

FLOGIC 2 Block — This block represents an opportunity to wrap-up the last solution step and/or prepare for the next. It can be used similarly to FLOGIC 0, and is called once after each steady-state solution and once after each time step has been taken.

4.1.3.7 OUTPUT CALLS Block

All submodels for which unique data is output to a file require an OUTPUT CALLS data block. It is therefore appropriate but not required that there be an OUTPUT CALLS data block for each thermal or fluid submodel. Otherwise, global output options are also available if data common to all submodels is needed.

The preprocessor converts the OUTPUT CALLS logic statements as required and consolidates them into a single subroutine OUTCAL in the manner similar to that shown in Figure 4-4. One notable difference between OUTCAL and the VARBLn blocks is that the preprocessor does not insert subroutine calls into the Fortran logic, thus, *there is no default output.*

The portion of OUTCAL that applies to a given submodel is executed whenever the value of the current problem time or iteration count advances to a multiple of the output interval for that submodel. The output intervals are defined by an array of values of control constant OUTPUT and ITEROT, one value per thermal submodel, and OUTPTF and ITROTF, one value per fluid submodel. OUTCAL may be called after every solution step. This is done for individual thermal submodels in transients by setting elements in the control constants array OPEITR to 1 (use OPITRF for fluid submodels); setting ITEROT and/or ITROTF to 1 forces OUTCAL each steady-state iteration.

OUTCAL is called at the beginning of each solution routine to record initial conditions. It will also be called in the event of an abort due to a condition found by the program.

The following are appropriate operations to include in the OUTPUT CALLS blocks:

1) Print values of interest
2) Save current values for postprocessors and graphics programs.
3) Save current values for printing or plotting in the post solution segment of the OPERATIONS block.
4) Write to a restart output file or to a crash file.
5) Write current values to a program file for initialization of solutions later in the run.

These operations are all quite straight forward and are implemented using several library subroutines that offer a variety of printout formats and file writing capabilities. Some of these routines allow the user to supply multi-word character strings as column header titles and parameter labels. The CARRAY DATA blocks provide a convenient means of entering this data. The PAGHED routine may be used to change the presentation of such output routines.

All the program variables and parameters are available for use in OUTPUT CALLS logic, as is also true in the VARIABLES n and FLOGIC n blocks. This allows the user to control output operations based on the values of any number of program variables.

OUTPUT CALLS, GLOBAL: Optionally, a single OUTPUT CALLS, GLOBAL block may be specified. This special block is always called independent of the BUILD or BUILDF configuration, and it is called after all the OUTPUT CALLS blocks for all currently built thermal and fluid submodels have been called *if they are called* at some point. The submodel prefix for variables such as T and PL is not optional for this block, though the DEFMOD command is available as an input convenience.

The presumed usage of this block is to have a single, central place for output operations that is always present in every model.

The instructions contained within HEADER OUTPUT CALLS, GLOBAL are invoked if *any* submodel's block is invoked during an output interval, time step, or iteration (per OUTPUT, OUTPTF, ITEROT, ITROTF, OPITER, or OPITRF).

As an example, if OUTPUT were set to 2 time units in HEADER CONTROL DATA, GLOBAL and OUTPTF were set to 3 time units, the global OUTPUT CALLS block would be called at TIMEN = 0.0, 2.0, 3.0, 4.0, 6.0, 8.0, 9.0, 10.0, 12.0 ... units. If OPITRF=1 were set for any fluid submodel during a transient solution (or if OPITER=1 were set for any thermal submodel), then the global output block would be called every time step after the thermal and fluid output blocks had been called.

Usually, output control constants are set globally, but if they are adjusted per submodel, the output frequency of the global block can only increase even if some submodels contain no output instructions. To avoid too-frequent output operations, use a register to define values such as OUTPUT and OUTPTF in the HEADER CONTROL DATA, GLOBAL block, and then change the register in logic to *indirectly* adjust the output frequency for all submodels at once.

4.1.3.8 SUBROUTINES Block and CALL COMMON

The SUBROUTINES block is intended to contain any subroutines the user wishes to call from the other logic blocks. Like the OPERATIONS block, there is just one SUBROUTINES block, though it may contain any number of user-supplied subroutines. The subroutines may utilize F-type or translatable statements which means that SINDA/FLUINT variables such as temperatures, conductors, pressures, arrays, etc. may be referenced using the smn.Xn or smn.Z(m+i) forms. Note that, as in the OPERATIONS block, the smn prefix must always be supplied when using references to SINDA/FLUINT variables within SUBROUTINES (see also the DEFMOD instruction).

CALL COMMON is a macroinstruction unique to SUBROUTINES. It includes the Fortran data modules that are automatically supplied at the beginning of the other logic blocks (see below), allowing access to all of the SINDA/FLUINT variables. This command is optional since it is not always necessary for a particular subroutine to reference SINDA/FLUINT variables. When it is used:

1. CALL COMMON must immediately follow the subroutine declaration, or at least precede other executable statements since it causes the insertion of various Fortran USE and other declaration statements.
2. CALL COMMON must not be preceded by an "F" (nor contained within an FSTART/FSTOP block) lest it be ignored by the preprocessor and result in a loader failure.

Thus, the user has a choice of whether any routine in SUBROUTINES should be (1) treated like other logic blocks, e.g., have access to central variables like C, G, DH, PL, and QTIE and control constants like TIMEND, OUTPTF, and NLOOPS, or (2) whether the routine should be more isolated, with all variables considered local, thereby alleviating concerns regarding collisions with reserved SINDA/FLUINT words. In the latter case, the FSTART/FSTOP and SPELLOFF/SPELLON instructions are very useful to turn off translation and spell checking, respectively. (No spell checks are performed if translation is off, but the reverse is not true: translations can still occur if the spell check feature is turned off.)

Section 4.1.3.9 documents the data in the modules that are inserted when CALL COMMON is detected.

The generic format of the SUBROUTINES block is as follows:

```
HEADER SUBROUTINES
       SUBROUTINE SUBR1 (arg1, arg2, ... argN)
       CALL COMMON
       .          .          .
       .          .          .
       .          .          .
       translatable and/or F-type statements
       .          .          .
       .          .          .
       .          .          .
       END
FSTART
       SUBROUTINE SUBR2 (arg1, arg2, arg3, ... argN)
       .          .          .
       .          .          .
       .          .          .
       F-type statements
       .          .          .
       .          .          .
       .          .          .
       END
FSTOP
```

As in any Fortran program, the argument list is optional. The END record at the conclusion of each subroutine is mandatory. The first subroutine SUBR1 is an example of a routine which will be treated like any other logic block. The second routine, SUBR2, lacks the CALL COMMON command and therefore any access to central variables and control constants. Translation is therefore turned off to avoid the need to worry about name collisions—any valid Fortran variable name may now be used in SUBR2 such as “T”, which will no longer mean a nodal temperature.

4.1.3.9 Program Data Modules

The CALL COMMON macroinstruction supplies Fortran data modules and a few named common blocks, which are always present in any FLOGIC, VARIABLES, OUTPUT CALLS, or OPERATIONS block. The following pages summarize the most important or frequently used variables.

Variable Name	Description
K	Numbered user constants
A	User arrays (other than T-arrays)
UCA	Character arrays
T	Temperature (nodal) array
C	Capacitance array
Q	Source (nodal) array
NDNAM	Submodel name array (character*32)
NDINT	Array of user node numbers
G	Array of Conductor values
HR	Conductor heat rates
PL	Pressure array
XL	Quality array
TL	Temperature (lump) array
DL	Density array
VOL	Tank (and junction) volume array
QDOT	Calculated source (lump) array
COMP	Tank compliance factor
CX	Lump X coordinate
CY	Lump Y coordinate
CZ	Lump Z coordinate
AL	Lump void fraction
QL	Input lump source
QTIE	Tie heat rate (before duplication factors applied)
UA	Tie conductance (before duplication factors applied)
GF	Ftie conductance
QF	Ftie heat rate
FR	Flow Rate array
DH	Tube (STUBE, REDUCER, EXPANDER) Diameter array*
TLEN	Tube (STUBE, REDUCER, EXPANDER) Length array*
AF	Tube (STUBE, REDUCER, EXPANDER) Flow area array*
WRF	Tube (STUBE, REDUCER, EXPANDER) Wall roughness array*
IPDC	Tube (STUBE) Pressure drop correlation array*
AFTH	Tube (and most connectors) family throat area*
VLO	iface lower volume limit, tank A
VHI	iface upper volume limit, tank B
EMA	iface mass
FI	Fluid properties array
UID	Unit system flag (INTEGER)
PATMOS	Global control constant (abs. zero in pressure units)
ACCELX,Y,Z	Acceleration vector components
Registers	Real and integer registers

Additional single-value program variables include:

MMODS . . . . . Total number of thermal submodels
NNOD . . . . . Total number of thermal nodes
NGTOTAL . . . . Total number of conductors
ABSZRO . . . . . Absolute zero.
SIGMA . . . . . Stefan - Boltzmann constant.
NMACT . . . . . Number of currently built thermal submodels
LINECT . . . . . Current line number on a printout page
PAGECT . . . . . Current page number
DTIMES . . . . . Pseudo time step for steady-state
TIMEN . . . . . Current problem time
TIMEO . . . . . Old problem time (Previous TIMEN)
TIMEM . . . . . Average of TIMEN and TIMEO for TRANSIENT (“FWDBCK”)
TIMEND . . . . . Stop time
NLOOPS . . . . . Number of steady state iterations allowed
LOOPCT . . . . . Current iteration count
MLINE . . . . . Number of lines allowed on a page

Most of these parameters are for information only, and should never be changed during a run. Exceptions include NLOOPS, TIMEO, TIMEND, and MLINE, and these should only be changed within OPERATIONS, not during a steady state or transient solution.

NSOL: This is a very useful integer variable. NSOL identifies the current solution routine, which can be used to alter the instructions in user logic blocks (VARIABLES and FLOGIC) and in user subroutines:

NSOL	= 0	STEADY (aka, “FASTIC”)
	= 1	STDSTL
	= 2	TRANSIENT (aka, “FWDBCK”)
	= 3	FORWRD
	= 4	(reserved)

Note that logic separated on the basis of steady-state vs. transient can use an NSOL comparison such as “IF(NSOL .LE. 1)”, or in an expression “(nsol <= 1)?”, to mean “during a steady-state routine ...”

Additional included data relates to the Solver and the Reliability Engineering module, which are introduced in Section 5.

UID: The UID (unit identifier) parameter is available as an integer in logic or expressions as follows:

UID = 0 (ENG)
UID = 1 (SI)

One purposes of this constants is to be able to write unit-independent models. For example, in FLOW DATA:

```
DH = (UID == 0)? 1/12 : fttom/12
```

or in FLOGIC 0:

```
      IF ( UID .EQ. 0) THEN
             CALL CHGLMP('CHILL', 102, 'PL',
     &                          PTEST*14.696/101325., 'XL')
      ELSE
             CALL CHGLMP('CHILL', 102, 'PL', PTEST, 'XL')
      ENDIF
```

4.1.3.10 Summary of Translatable Parameters

All above parameters constitute the reserved word list, meaning that none of these names can be reused by users in their logic without causing name collisions. While all are available, only a subset are recognized by the preprocessor as translatable (i.e., can accept submodel prefixes and/or user ID suffixes, which are converted into global array references). Single-value numbers like LOOPCT, ABSZRO, ATEST through ZTEST, ATEST_DP through ZTEST_DP, ATEST_SP through ZTEST_SP, and NSOL need no translation. The fluid identifier, FI, is translatable but follows unique rules (see Section 7.11.2). All control constants are translatable, given submodel prefixes. The remaining translatable variables, of the form [smn.]Zn, are:

Nodes	All	T, Q (Q ignored if boundary)
	Diffusion	C
Conductors	All	G, HR
Lumps	All	PL, TL, HL, DL, XL, CX, CY, CZ, AL, DF, DG, QDOT, QL (QL ignored if plenum), QTM, XGx, XFx, PPGx, MFx, where "x" is the letter identifier of the constituent.
	Tank/Junction	VOL, ASL, CASL, GLx, CGLx, FRDx, ZRx, ZJ, PH, FRHx, ULI,UVI, CULI, CUVI, HENx, or IDGC, where "x" is the letter identifier of the constituent
	Tank	COMP, VDOT, AST, CAST, ULT, UVT, CULT, CUVT, GTx, CGTx 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, TLTH, XLTH, RADI, RADJ, VAI, VAJ, ROTR, VXI, VXJ, EFFP, TORQ, QTMK, 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, REW, REX, RVR, MREG, XMA, 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
	REDUCER and EXPANDER	HC, AC, TLEN, WRF, DH, DHJ, DHJ, AFI, AFJ, FKI, FKJ, RAD_A, RAD_R, MODEC
	MFRSET	SMFR (stored internally* in the HC array)
	VFRSET	SVFR (stored internally* in the HC array

* "Internally stored as ..." parenthetical comments: Actually, there are no real arrays named after most connector device mnemonics such as "RC" or "SMFR." The exceptions are STUBE parameters, all of which are real arrays that also contain tube data. To reduce memory requirements, device data for all other connector devices is stored within the arrays set aside for tubes and STUBEs. As an example, the SMFR for an MFRSET connector is stored internally within the program in the appropriate cell of the HC array, which is otherwise meaningless for that connector. The preprocessor converts the occurrence of "SMFR(" in logic blocks to "HC(" as can be seen in the output Fortran files. This has little impact on the user, except that **none of these arrays should be used as scratch space.**

	LOSS/valves	FK, TPF (includes CHKVLV, CTLVLV, UPRVLV, and DPRVLV connectors; TPF is stored internally* in the WRF array)
	LOSS2	FKB (stored internally* in the AC array)
	ORIFICE	FK, AORI, CDIS, ELLD, MODO (integer) (internally* stored in the FK, AC, WRF, TLEN, and IPDC arrays, respectively) MODA (integer), AORI_C, AORI_T, AORI_L, AORI_H (internally* stored in the IREG, ASPU, ASPD, FC, and FPOW arrays, respectively)
	TABULAR	OFAC (internally* stored as FC divided by FPOW), AFI, AFJ, UPF
	(U/D)PRVLV	FKH, FKL, PSET, RLAG
	CAPIL	RC, CFC, XVH, XVL (same for 1st CAPPMP connector; stored internally* in the DH, FD, TLEN, and WRF arrays, respectively)
	PUMP	SPD (stored internally* in the FC array), also RSPD, HPMP, GPMP, DGDH, HCF, GCF, TCF, ANPSH, RNPSH, ANSS, RNSS, SSCF, CCH, CCE, VCOR, HVF, GVF, EFFVF, 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, MODR (integer), IHTC (integer, reserved), ICHF (integer), RLAM, RTURB, NTWOP (integer, output only)
	HTP	DCH, DEPTH, ACCM, SPR, CSF, XNB, CHFF, HFFL, MODR (integer), XNUL, XNLM, XNLA, HCON, THKL. (Many are internally stored in other arrays, such as UEC, UEP, etc.)
Fties	All	GF (meaningless for CONSTQ), QF, DUPC, DUPD
Interfaces	All	VA, VB, PA, PB, FPV, DUPA, DUPB
	All but NULL	EMA, VLO, VHI
	OFFSET	DPAB
	SPRING	DPDV, VZRO
	WICK	VZRO, VSUB, RCAP, RBP, XIL, XIH
	SPHERE	XIL, XIH
User data	Arrays	A or NA (equivalenced)
	Constants	XK or K (equivalenced)
	CARRAYs	UCA

4.1.4 Logic Translation

[The reader is advised that many of the details regarding logic translation can be avoided when making changes to SINDA/FLUINT variables by using the dynamic register and expression features described in Section 4.9. However, when referencing processor variables in such expressions, the rules governing these references are largely the same as those described in this section.]

The following discussion presents the basic information required to prepare the logic blocks. Examples of properly prepared inputs are presented and will serve to introduce the user, by illustration, to the techniques of executing the activity described in flow charts. Fluid submodel operations will not be described explicitly here because they follow the same rules as the thermal submodel operations. Similarly, the logic blocks used by the Solver and the Reliability Engineering module (Section 5) follow the same rules detailed below.

The user is reminded that all logic appears beneath one or more of the following header records which may appear in any order. The OPERATIONS block is the only one that *must* appear, although a run without OUTPUT CALLS would be a very special case.

```
HEADER OPERATIONS
C         Executive (Driver) Operations
HEADER VARIABLES 0, smn
C         VARIABLES 0 Operations
HEADER VARIABLES 1, smn
C         VARIABLES 1 Operations
HEADER VARIABLES 2, smn
C         VARIABLES 2 Operations
HEADER FLOGIC 0, smn
C         FLOGIC 0 Operations
HEADER FLOGIC 1, smn
C         FLOGIC 1 Operations
HEADER FLOGIC 2, smn
C         FLOGIC 2 Operations
HEADER OUTPUT CALLS, smn
C         OUTPUT CALLS Operations
HEADER SUBROUTINES
C         User Subroutines
```

The header records must appear in the format shown. *Blanks and commas serve to delineate key words and must appear.* More than one blank between key words is of no consequence, but consecutive commas must not appear.

As a further reminder, all logic consists of either F-type (untranslated or verbatim) Fortran statements and translatable Fortran statements. An F-type statement is any valid Fortran statement. A translatable statement is similar to an F-type statement, except that the *actual* numbering system (user-provided integer identifiers) may be used to reference data. That is, F-type statements are

FORTRAN statements which are not translated by the preprocessor, whereas translatable statements are Fortran statements that are modified by the preprocessor to reflect translation from the *actual* (user) to the *relative* (internal) numbering system. Another feature of translatable statements is the submodel name qualifier on certain variable names. This is another aspect of the translation from *actual* identification to the *relative* numbering system. The default statement is translatable: a blank in column 1 will be understood as a translatable statement. An F *must* appear to identify F-type statements. A block of F statements may also be delineated by a FSTART and FSTOP (Section 4.1.1.2).

4.1.4.1 Translatable Statements

Translatable statements may be used for any purpose that a regular Fortran statement can accomplish in a Fortran program. Thus, translatable statements may be thought of as a SINDA-specialized extension of the Fortran language. Nonstandard statements run the risk of generating compiler errors.

The following paragraphs illustrate, by example, the various formats and uses of translatable statements. Also note that although the examples used in the next two sections are for thermal variables, almost all FLUINT variables are likewise accessible, as was summarized in the last subsection.

Translatable statements may be used to call subroutines. Such subroutines may reside in the SINDA/FLUINT library, the computer system library or as user-supplied subroutines in the SUBROUTINES block. The subroutines in the libraries are pre-compiled Fortran subroutines. The subroutines in the SUBROUTINES block are interpreted and compiled along with the other logic and thus may consist of translatable statements.

The general format for a subroutine call that requires no arguments is as follows:

```
120     CALL STEADY             $ EXAMPLE 1
        CALL TRANSIENT          $ EXAMPLE 2
```

The number 120 in Example 1 is a statement number. Its only purpose is to provide a target point for program flow control using Fortran IF, GO TO, or DO statements (see Section 1.7.4). When no such target point is needed, the statement number field may be left blank. When a statement number is used, it should be unique to its data block, but it need not be chosen in any particular sequential order.

Example 1 shows a translatable statement which calls subroutine STEADY. A statement number, 120, is used so that some such Fortran statement as GO TO 120 will cause program flow to proceed directly to this statement. Example 2 shows a statement that calls subroutine TRANSIENT.* Note that both are understood as translatable statements, even though it wasn't explicit in example 1. These examples also show the use of the optional comment data field to the right of a "$" delimiter.

* This routine will be introduced along with other solution routines in this section. Detailed descriptions of other routines may be found in Section 7.

Translatable statements that call subroutines that require arguments are formatted similarly, as follows:

```
CALL D1DEG1 (RTEST, A5, G7)          $ EXAMPLE 3
CALL QVTIME ('SUBM2')                $ EXAMPLE 4
```

In Example 3, RTEST, A5 & G7 are arguments. A5 is a reference to User Array 5 and G7 is a reference to Conductor 7. RTEST is a user constant that should appear in USER DATA, GLOBAL. Its appearance in USER DATA is not mandatory, however. The ramifications of this are explained below in Section 4.1.1.3.

Example 3 would be correct only if it appeared under a header record with the same submodel name that A5 and G7 were entered under. If Example 3 were under another submodel name, or if it were in the OPERATIONS block or the SUBROUTINES block, it would have to appear as follows:

```
CALL D1DEG1 (RTEST, SUBM1.A5, SUBM1.G7)$ EX. 3A
```

where SUBM1 is the submodel name associated with A5 and G7.

Both translatable and F-type statements must end prior to column 132. If this does not provide sufficient space, they may both be continued, according to the Fortran rule, if a continue character is placed in column 6 of the statements that follow, as shown below:

```
  Column 6
      | |
101     CALL ADD (SUB1.T4, SUB2.T7, 9, SUB1.T21,
       +   T36, SUB1.T3, SUB2.T8, T27, T19,  $ COMMENTS
       +   T14, SUB2.T106, T9241, 308.4, XK5,
       +   XK5, TTEST)                       $ EX. 5
```

It should be noted that:

1) Any statement numbers must appear only on the first line of multi-line statements.

2) Permissible arguments allowed on translatable subroutine calls include literal data values, user constants names, register names, and references to program variables by *actual* (user) identification numbers, with or without submodel qualifiers.

3) Arguments must be separated by commas. Blanks are ignored.

4) Comment material will not affect the translation to Fortran as long as it is to the right of the $ terminator on any line.

Consider example 3, above. To properly use the interpolation subroutine D1DEG1, the user must supply it with three arguments, as follows:

1) The location of the value of the independent variable, x.

2) The location of the integer count for an array of x, y pairs.

3) The location where the result of interpolation, y, is to be placed.

The identifier RTEST refers to a specific location in the block of memory reserved for global user constants. The identifier (or symbolic reference form), A5, refers to the location of array number 5. The identifier, G7, refers to the location reserved for the conductance of conductor number 7. Thus, example 3 illustrates a translatable statement which accomplished the operation: “Call a subroutine which interpolates an array number 5 to find the conductance of conductor number 7, using the value of user constant RTEST as the independent variable.” Note that, in using subroutine D1DEG1 to operate on RTEST, A5, and G7, it was not necessary for the user to know, for example, where conductor 7 was located within the block of memory locations reserved for conductances. It was merely necessary for the user to indicate, symbolically, that it was specifically conductor number 7 that needed to be evaluated. This *actual* (user) numbering system is unacceptable to the Fortran compiler because the preprocessor reorganizes and packs the user’s data, as described in the discussion of the data blocks. However, the preprocessor rectifies this inconsistency by performing a translation on each translatable statement, replacing each *actual* reference with its corresponding, Fortran compatible, *relative* (internal) reference.

Example 3A shows the extension of translatable statements to include submodel names in the identifier forms. In this case, the submodel name SUBM1 did not appear in the preceding header record, so the name had to appear as a qualifier with A5 and G7. RTEST is a “global” user constant that is not and cannot be identified exclusively with a particular submodel. If the independent variable was part of a particular submodel, the user constant would have to be numbered, and Example 3 might appear as follows:

```
CALL D1DEG1 (SUBM3.XK12, A5, G7)                $ EX. 3B
```

To generalize on examples 3, 3A & 3B, the following list describes the form of all arguments permitted in translatable statements:

1) Data values which are acceptable to the FORTRAN compiler:
   a. Integer (e.g., 12345678)
   b. Floating point or real (e.g., 12.4, 4.0E-8, 15.D3)
   c. Character (e.g., ’ABCD’)

2) Temperature references of the form Tn or smn.Tn where n = actual node number and smn is a submodel name (e.g., T4, POD2.T4)

3) Capacitance references of the form Cn or smn.Cn (e.g., C6, POD2.C6)

4) Source references of the form Qn or smn.Qn (e.g., Q7, POD2.Q7)

5) Conductance references of the form Gn or smn.Gn where n is an actual conductor number (e.g., G12, POD1.G12)

6) User constant references of the form Kn, XKn, smn.Kn or smn.XKn were n is an actual user constant number (e.g., K11, POD2.XK12)

7) Named global user constant references (e.g., TTEST, KTEST, PTEST_DP, JOE, MATT)

8) Register names (defined in HEADER REGISTER DATA)

9) Control constant references (e.g., TIMEND, POD6.DRLXCA)

10) Numeric user array references of the forms An, smn.An, A(n+i), smn.A(n+i), NAn, smn.NAn, NA(n+i) or smn.NA(n+i) where n is an actual user array number and i is the element position in the array (e.g., A6, POD6.A(6+12), NA2)

11) References to fluid variables (PL, TL, FR, etc.) as defined in other sections (see Section 6.18.4 for a summary).

Using Sinaps or Thermal Desktop, the input order of nodes, conductors, etc. is unknown and therefore should not be used as the basis for translations. *Dynamic translation routines like NODTRN (or the INTNOD function), CONTRN (or the INTCON function), INTLMP, INTPAT, and INTTIE are the preferred methods for such uncertain references.* Use of these routines eliminates concerns regarding storage rules and eliminates errors caused by editing a model.

As an example of translation, assume that the relative (internal) number for actual (user) number 90 is 9. Consider, then, the Fortran expressions generated by the preprocessor for the following expressions that appear in translatable statements:

	Translatable Expression	Fortran Expression	
(1)	T90	T(9)	
(2)	T(90)	T(9)	
(3)	NA90+1	NA(9)+1	
(4)	NA(90)	NA(9)	
(5)	A(90+1)	A(9+1)	
(6)	A(90+L)	A(9+L)	
(7)	G(90+1)+L	G(9+1)+L	
(8)	G(90+1+L)	G(9+1+L)	*Info only, use INTCON instead!*
(9)	GLOBAL.T(90)	T(90)	
(10)	GLOBAL.T(90+K90)	T(90+K(9))	*Info only, use INTNOD instead!*

As another example, the following translatable statement will cause all of the values in array number 90 to be written out on logical unit number *nuser1*, regardless of the number of values or the particular *relative* location of the array:

```
WRITE (NUSER1,*)(A(90+I),I=1,NA(90))          $ EX. 7
```

It can be seen that these reference forms can become extremely complex. The only inflexible rule is that in the sequence: “key variable name-left paren-n” [e.g., “T(n”], n must be a literal integer corresponding to some actual number of an input data value. This integer is always the first following a translatable keyword, and *only this integer will be translated.* Note also that the non-subscripted form (e.g., K12) disallows expressions.

In examples 6 and 7 submodel references were not included to enhance clarity. Any of the key variable names may be preceded by a submodel name qualifier in any translatable statement.

For those cases where access to relative data is required, global access is provided. The input “GLOBAL.key-variable-name (GLOBAL.Tn) turn off the translation of N. This allows translation to be turned off term by term and not just by line as before. Global references are dangerous and should be avoided.

The following statements illustrate the conversion process when the preprocessor generates Fortran expressions from translatable statements. In considering these examples, assume that there is the following *actual/relative* (user/internal) correspondence. Assume for clarity that the same correspondence applies by coincidence to each data block:

ACTUAL (n)	RELATIVE (m)
10	1
20	2
30	3
40	4
50	5

```
        XK20 = (T10+T40)/2.0                    $ EX. 9
42      G10   = SQRT(XK20)*XK30                 $ EX. 10
        XK40  = A50                             $ EX. 11
        ATEST = XK20 + GLOBAL.XK20              $ EX. 11A

C       FORTRAN EQUIVALENTS FOR EXAMPLES 9, 10, AND 11:

        XK(2) = (T(1)+T(4))/2.0                 $ EQ. EX. 9
42      G(1)  = SQRT(XK(2))*XK(3)               $ EQ. EX. 10
        XK(4) = A(5)                            $ EQ. EX. 11
        ATEST = XK(2) + XK(20)                  $ EQ. EX. 11A
```

Example 9 will cause the average temperature of nodes 10 and 40 to be placed in numbered user constant 20 as a floating point number. The statement in example 10, which has been assigned statement number 42, will compute the conductance of conductor 10 as the square root of the value in user constant 20, times the value in user constant 30. The conductance will be a floating point number, as required, and it is assumed that the user constant 30 contains a floating point number (because the floating point identifier, ‘XK’, was used in place of the integer identifier ‘K’). The array reference, A50, in example 11, is of the integer count form An, and thus references the integer count for array number 50. The statement will result in user constant 40 receiving this integer value. No conversion of arithmetic type will occur because the identifiers, ‘A’ and ‘XK’, are both of the same type (i.e., floating point), and the computer’s internal logic will not be confused because replacement (i.e., equals sign) is not an arithmetic or relational operation. Quite simply, the content of location A50 (which just happens to be an integer) will be stored in the location of user constant 40. As a result of this statement, the *integer* identifier, K40, may be used in subsequent statements as a reference to the value of the integer-count for array 50.

When using functions and subroutines, the user must take particular care to see that the number and type of each argument supplied matches the number and type of each argument expected. A compiler-level error will result otherwise (at least for functions and subroutines known to SINDA/FLUINT such as those contained in the library, Section 7).

That is, if the description of a particular subroutine specifies, for example, that each argument must be a floating point (real) number, the user must be certain that the identifiers that are supplied as arguments refer to locations which contain floating point numbers, and that any data values supplied directly as arguments are, in fact, floating point data values. The following examples illustrate this point. Consider subroutine ADD, which places the floating point sum of the first n-1 floating point arguments into the n^{th} argument. Assume that the following record was entered in a USER DATA block:

```
10=1.2, 11=2.4, 12=0, 13=1.0, 14=1, 15=0, 16=4
```

Example 12, below, shows a correct usage of subroutine ADD. This statement would cause the value 4.6 to be stored in the memory location for numbered user constant 12. Example 13, however, is incorrect in part because the second argument does not represent a floating point number.

```
CALL ADD  (XK10,XK13,XK11,XK12)      $ EX. 12 (correct)
CALL ADD  (K10,XK14,K11,K15)         $ EX. 13 (incorrect)
```

Even though the first argument in the above example (#13) *is* defined as a floating point, it is referred to as an integer value and so will cause a compiler error during argument type checking.

Subroutine ADDFIX, on the other hand, operates on integer (fixed point) arguments, thus making example 14 correct, with constant K15 receiving the value 5, and example 15 incorrect because K13 is not an integer:

```
CALL ADDFIX (K14,K16,K15)            $ EX. 14 (correct)
CALL ADDFIX (K13,K16,K15)            $ EX. 15 (incorrect)
```

As explained in the discussion of the USER DATA blocks, an identifier is equated to the memory location where the current value represented by the identifier is stored. When identifiers appear in argument lists, they serve to communicate either (1) the location of a particular value, or (2) the starting location of a sequence of values, depending on what the given subroutine expects. Thus, for example, the identifier T5 may be used as an argument that is expected to be a single floating point value. The value used by the subroutine will be the current value of the temperature of node number 5. In addition, however, this same identifier could be used as an argument that is expected to be the starting location of a sequence of floating point values (i.e., the starting location of an array). The values used by the subroutine will be the sequence of data values located in the node temperature table beginning with the temperature of node number 5.

To illustrate these two uses for the same identifier, consider the following examples. Assume that the following record was included in a NODE DATA block:

```
GEN 5,10,1,70.0,4.5              $ 10 NODES, 5 THRU 14
```

and the following cards appeared in a Variables 0 block:

```
CALL ADD (T5,T6,T7,T8,T9,T10,STEST)   $ EX. 16
CALL SUMARY (6,T5,STEST)              $ EX. 17
```

Example 16 shows a proper usage of the identifier, T5, as a reference to a single value. On the other hand, subroutine SUMARY requires three arguments. (1) the number of values to be added (integer), (2) the starting location of the sequence of values (array) to be added, and (3) the location to receive the result of the addition. Hence, example 17 shows a proper usage of the same identifier, T5, as reference to the starting location of an array. Clearly, the operation in example 16 performs the same action as the operation in example 17.

Supplying an identifier that represents the starting location of a sequence of data values, however, should not be confused with supplying an identifier that represents the location of a user array. In the former case, the number of data values in the sequence must be supplied as a separate argument, but in the latter case, the integer count of the array provides this number and the sequence of data values is assumed to begin at the memory location immediately following the integer count. Thus, while the user is restricted to using identifiers of the form An (never A(n+i)), when a subroutine expects an argument representing an array integer count, the user may supply almost any identifier as an argument to a subroutine which expects an array starting location (illustrated in example 17).

Consider the following illustration: Assume that array 10 contains floating point values. Comparing example 17 with example 18 below, it will be seen that the latter will cause the sum of the values in array 10 to be placed in global user constant UTEST. The array location reference, A10, supplies the number of values in array 10, and the data value reference, A10+1, supplies the starting location of those values. On the other hand, the use of the array location reference in example 19 supplies the number of values in array 10, along with the implicit knowledge (implicit, that is, in the sense that subroutine D1DEG1 expects it to be so) that the array of values begins at the next location in memory immediately following the location of this integer count.

```
      CALL SUMARY (A10,A(10+1),UTEST)       $ EX. 18
      CALL D1DEG1 (TTEST,A10,RTEST)         $ EX. 19
```

When using explicit data values (instead of identifiers) as arguments, the user must take great care to insure that the given subroutine does not alter such arguments in any way. Stated another way, any argument which is (or might be) altered by a subroutine must be supplied as an identifier, not a data value. While this rule is seldom violated intentionally, experience has shown that it is often violated through oversight, with generally undesirable consequences. In the usual case, the error is caused by transposing the order of the arguments when calling a particular subroutine. Consider the following statements (see next section for F-type statements):

```
      KTEST=1
      CALL ADDFIX (KTEST,3,2)
      LTEST=2+2
F     WRITE (6,100) LTEST
F 100 FORMAT (I8)
```

The call to subroutine ADDFIX is clearly in error because the result of the addition will be returned to the last argument which is not an identifier. Evidently the user believed that the result would be returned to the first argument. The chaotic effect of this bad call will be obvious when it is understood that the value of LTEST that is printed by the WRITE statement will be 8, not 4! Two

plus two equals eight? Yes, because ’2’ no longer equals 2 after the bad call to ADDFIX; it now equals 4, the sum of KTEST and 3. Hence, the warning is clear: *do not supply data values as arguments that receive the results of a subroutine.*

4.1.4.2 Function and Subroutine Argument Checking

Fortran is a “weakly typed language,” meaning (as an example) that it doesn’t stop a programmer (or SINDA/FLUINT user) from passing an integer data type to a routine expecting a real (floating point) type. Since only a pointer to a sequence of bits is passed, the receiving routine can completely misinterpret those bits, resulting in insidious errors.

SINDA/FLUINT include Fortran “interfaces” ... a definition of user-callable subroutines and functions that the compiler can use to check for appropriate arguments: number, type, and so forth. This allows the compiler to stop a dangerous run before SINDA/FLUINT even starts. For example, the function PRPMAP expects 3 arguments: a floating point pressure and temperature, and an integer pointer to a fluid identifier (this is what the “FI6123” or “model.FI” is converted to by SINDA/FLUINT):

```
call prpmap(400.0, 500.0, mymod.fi)
```

If the wrong type of argument had been supplied, a compiler error would have prevented the run from being initiated. For example, *the following are all illegal*:

```
call prpmap( 400, 500.0, mymod.fi) $ pressure should be real
call prpmap( 400.0d0, 5.0e0, wrong) $ fluid ID should be an integer
call prpmap( 400.0e0, 500.0)  $ not enough arguments and no default
call prpmap( ptest, itest, nother.fi) $ temperature must be real
call prpmap( i_reg, ttest, nother.fi) $ if “i_reg” is real, then OK
```

Because it is the Fortran compiler that catches these problems, the error messages are not produced by SINDA/FLUINT. Instead, the location in the astap.for file is noted and a message is produced that is more programmer-oriented than engineer-oriented. Some examples are given below that were produced by the CVF compiler (exact messages and formats are compiler-specific).

In the most common type of error, the type of data provided does not match the type expected. The first argument for NCPROP is a floating point pressure, but in the following case an integer data value (“40000”) has been supplied instead (on line 1305 of the astap.for file):

```
astap.for(1305) : Error: The type of the actual argument differs from the type of the dummy
argument.   [400000]
              CALL NCPROP( 400000 , TTEST , 0.0000000 , ATEST , PTEST , CTEST , FI(    64) )
------------------------------^
```

(In prior versions, care had to be taken to use double precision variables where expected. This is no longer necessary, since no distinction is made between single and double precision: all variables are now double precision. See Section 4.1.3.1.)

In some routines, the number and type of argument is variable (as defined in the documentation). SINDA/FLUINT has the capability to choose the correct routine based on the arguments provided for a small subset of user-callable routines. For example, PSWEEP can been called with either an integer or real register, so internally SINDA/FLUINT lists various options for the compiler, so that it can choose the right specific instance based on the arguments used. However, mistakes in input can mean the compiler will be unable to choose the right "generic" subroutine from the provided internal list. For example, an extra argument in the following case generates such a message:

```
astap.for(1317) : Error: There is no matching specific subroutine for this generic subroutine
call.   [PSWEEP]
        CALL PSWEEP( 'INT_REG', 1, 10, 1, 2, 'steady')
-------------------------------------------------------------------------------------^
```

In the next case, a user has written a function for SUBROUTINES that receives the fluid pointer (e.g., "mymod.fi") and passes it on to a SINDA/FLUINT function (namely: VVISCV, the viscosity of a vapor or gas given pressure and temperature and fluid pointer).

```
      integer d2(1)
```

This causes the following error message regarding a mismatch in anticipated versus provided "shape" (number of dimensions of an array used as an argument):

```
astap.for(2844) : Error: The shape matching rules of actual arguments and dummy arguments
have been violated.   [D2]
       re = FR(path)*DH(path)/AF(path)/vviscv(pl(lump),TL(lump),d2)
-----------------------------------------------------------------------^
```

Although it is difficult to provide guidance for such a rare problem in the general case, for this specific case, the above mismatch could be eliminated by changing the dummy variable d2 to be declared as follows:

```
      integer d2
```

Users of SINDA/FLUINT Versions 5.2 and older should consult Section 4.1.4.6 and Section 4.1.4.7 for additional guidance.

4.1.4.3 F-Type Statements

An F-type statement is any valid Fortran statement requiring no SINDA/FLUINT translations. Such statements may be used in any of the logic blocks, and their basic format is as follows:

```
                                                  7
1      67                                  Column  3
|       ||                                         |

Fsn     Fortran-statement      comments

F       GO TO 120                                  $ EX. 20
F140    IF (K(3).LE.75) ITEST = 4                  $ EX. 21
F130    IF (RTEST.LE.STEST.AND.ITEST               $ EX. 22
F     *  .GT.KTEST) GO TO 27
```

The 'F' in column 1 must be present in order to inform the preprocessor that the card contains an F-type Fortran Statement. A group of F statements may be blocked using FSTART and FSTOP. A statement number, sn, of one to four digits may be entered in columns 2-5, if required by the user's application. If a complete statement will not fit in columns 7-132, it may be continued in columns 7-132 of succeeding records by placing a "continuation character" (any character except blank or zero) in column 6 of each additional record.

Example 20 shows a correct F-type statement. Example 21 shows a correct statement including a statement number, and example 22 shows how a statement may be extended to more than one record.

Since the dollar sign, $, has a special meaning in Fortran, it may not be used to indicate the end of the statement field on an F-type record. Although SINDA/FLUINT provides a comments-only card in the form of the REM card, it will also accept the following, Fortran-compatible comment card:

```
C THIS FORTRAN-COMPATIBLE COMMENT CARD IS EXAMPLE 23
```

Note that a different type of user constant reference form, K(3), is included in example 21. This is a Fortran-compatible *relative* reference. It directly addresses the third location in the table of user constants. The constant so referenced will be the third user constant entered in the USER DATA block. *That example assumes the user knows absolutely what to expect in the third location of that array and is willing to take some risk that no changes will take place in the relative storage sequence.*

Since the user constants are referenced by the letter 'K', and 'K' is assumed by the Fortran compiler to represent integer values, a conflict of arithmetic type may result when the location referenced contains a floating point value. For this reason, references to floating point user constants should use the key letters 'XK' in place of 'K'. (Note that the identifiers 'K' and 'XK' are automatically EQUIVALENCED in each logic block.) The 'XK' form need be used only in arithmetic and relational expressions in Fortran.

For example, consider the following statements:

```
F       IF (RTEST.LE.K(3))STEST=K(4)          $ EX. 24 (incorrect)
F       IF (RTEST.LE.XK(3))STEST=XK(4)        $ EX. 25 (correct)
```

If user constants K(3) and K(4) contain floating point numbers, the statement in example 24 will produce erroneous results. The difficulty arises because the Fortran compiler will assume that K(3) and K(4) contain integers, which must be converted to floating point values before being compared with or transferred to the floating point values RTEST and STEST, respectively. Still assuming that K(3) and K(4) contain floating point numbers, then example 25 shows the correct form for referencing them.

Referencing data by its relative (internal) input order is clearly much more tedious and prone to error than referencing data by its user-assigned actual number. For this reason, the translatable statement was developed to permit the user to reference data by its actual number within ordinary Fortran-type statements.

If Fortran statements must be used (or if the user is uncertain of the storage sequence), pointers into the arrays are best obtained using one of the utility subroutines NODTRN (or the INTNOD function), CONTRN (or the INTCON function), ARYTRN, MODTRN, NUMTRN, MFLTRN, INTLMP, INTTIE, and INTPAT. (See Section 7.6 and Section 7.11.1.6.) For instance, the location of the temperature, capacitance or Q-source for actual node number 120 of submodel POD2 can be obtained with the following statement:

```
CALL NODTRN ('POD2', 120, ITEST)
```

This call will return the relative location of the three attributes of node 120 as an integer number with the variable name ITEST. (Note: the above statement could be either F-type or translatable, since no translation by the preprocessor is required.)

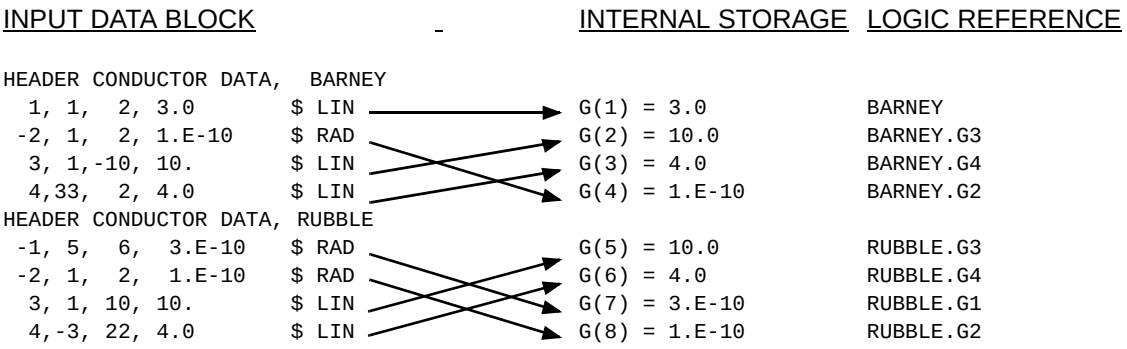
4.1.4.4 Internal Storage vs. Logical Reference

While a user need never know how a parameter is stored internally, advanced users occasionally demand such a level of understanding to enable more complex manipulations.* This understanding requires a knowledge of translation rules, internal storage sequences, and other such peculiarities. As an aid in putting it all together, this section provides specific multiple-submodel examples of the three types of variables: element-dependent, submodel-dependent, and array-dependent.

Caution: The user is **STRONGLY** advised to not rely on a knowledge of internal storage rules to perform his or her manipulations. Relying on an assumed internal storage sequence makes a model difficult to maintain, and can be the source of insidious errors. In many automatically generated models, including those generated by Sinaps or Thermal Desktop, *there is no input order at all* upon which to base a knowledge of internal storage. Dynamic registers (Section 4.9) or dynamic translation routines (e.g., NODTRN, INTLMP, as described in Section 7.6 and Section 7.11.1.6) should be used if at all possible.

* Dynamic translation routines such as NODTRN, CONTRN, INTLMP etc. are recommended even if the user is an expert on the internal storage schemes, since users who inherit those models might not be.

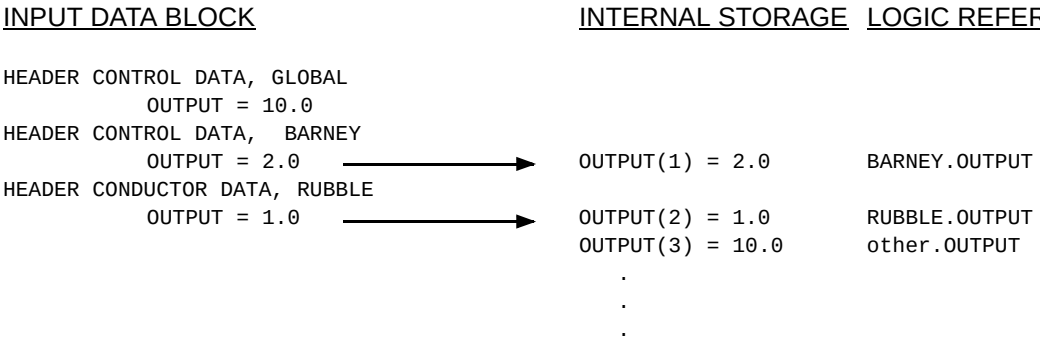
To illustrate element-dependent parameter storage and access, conductors will be used. Recall that conductor data are stored by input submodel order, subdivided by linear and radiation types. Node data are stored analogously, subdivided by diffusion, arithmetic, and boundary. Submodels are stored by input order (if ever built) according to the sequence of NODE DATA and FLOW DATA headers within the input file. The following example illustrates how input order, storage order, and logic references are related for conductors in two submodels:



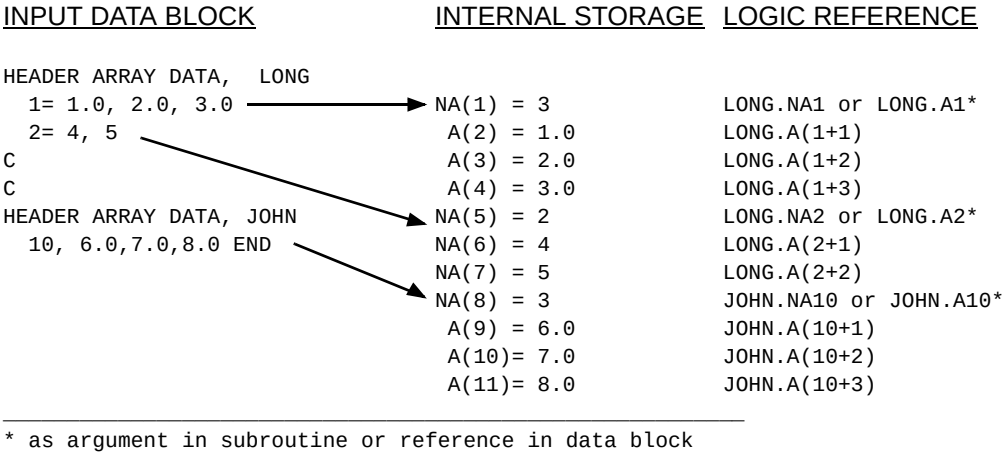
In the above example, HEADER NODE DATA, BARNEY is assumed to precede HEADER NODE DATA, RUBBLE.

Fluid submodel element-dependent data follow similar rules to the above example, except they are grouped first by type (tank, junction, and plenum, or tube and connector). Within the one category (e.g., tanks), they are subdivided by submodel.

Submodel-specific control constants, whether thermal or fluid, are stored internally as one dimensional arrays whose subscript refers to the submodel input order. The following example illustrates how input order, storage order, and logic references are related for control constants in the same two thermal submodels from the last example, assuming they are the first models input:



Array data (excluding CARRAY data) is all stored in one array, again according to submodel order. Unlike other data, a single cell is inserted at the beginning of each array which is equivalenced to its integer length:



4.1.4.5 Cautions Regarding Reference to Registers in Logic Blocks

Like named user constants (input in USER DATA, GLOBAL), registers (input in HEADER REGISTER DATA) can be accessed in logic. However, an important caution applies.

Unlike named user constants, the type of Fortran variable used for a register is *not* determined by its initial letter, but rather by the presence or absence of the prefix "INT:" in HEADER REGISTER DATA, as explained in Section 2.8.

4.1.4.6 Logic used in Versions Older than 5.3

Hopefully the dangerous nature of the problems listed at the end of Section 4.1.4.1 is apparent, such that the extra care that must be taken to satisfy the extra compiler rules will be tolerable.

For example, in SINDA/FLUINT versions older than 5.3 (which also distinguished between 32-bit and 64-bit variables), the difference between 0, 0.0, and 0.0D0 passed as an argument was often irrelevant since all of those point to a string of zeroed bits. This is not true for other numbers: -1 and -1.0 are represented very differently internally. For the compiler to check types, it cannot forgive rare cases such as 0=0.0 being equivalent, and must instead insist on the correct type of argument being provided within user logic.

For example, the pressure argument P for NCPROP and other natural convection routines, for EXTOBJ and other external flow convection correlations, for JETARN and other jet impingement convection correlations, etc. all use the mechanism of defaulting to atmospheric pressure if a zero or negative pressure has been supplied. In the past, a value of "0" was accepted and often worked fine. In the current SINDA/FLUINT, "0.0," "0.0e0," or "0.0d0" must be used instead to avoid a compiler error (also acceptable are registers, pressures (PL), etc.).

As an example, the following is illegal:

```
CALL JETSRN(Uout, 0, Temp, Flow, Diam, Height, Rad, FI8267)
```

and must be replaced with (for example):

```
CALL JETSRN(Uout, 0.0, Temp, Flow, Diam, Height, Rad, FI8267)
```

The numbered user constants (K and XK) are stored in the same array ("equivalenced," in Fortran terms). Similarly, user arrays (A for NA) are stored in the same master array. This meant that in the past, the use of K10 was equivalent to XK10 when used as an argument in a routine or function. More commonly, A was supplied as argument when NA should have been used for clarity when passing an integer string (e.g., the ID array for COMPARE). In Version 5.3 and beyond, such casual usage must be replaced by more careful attention to the type of array (and therefore the type of data it contains) to avoid compiler errors.*

As an example, the following is illegal:

```
CALL COMPARE(A100,A200,A300,0,'rmserr',object)
```

and must be replaced with:

```
CALL COMPARE(A100,A200,NA300,0,'rmserr',object)
```

Finally, for logic added to SUBROUTINES, care must be taken to invoke CALL COMMON before *any* user declarations. For example, the following is illegal:

```
INTEGER D2
CALL COMMON
```

and must be replaced with:

```
CALL COMMON
INTEGER D2
```

The reason for the above requirement is that CALL COMMON now inserts a "use sf_interfaces" command to invoke a Fortran module containing the routine and function INTERFACE declarations, and such a module must be placed before any other type declarations.

4.1.4.7 New Functionality or Calling Variations as of Version 5.3

Fortunately, additional obligations to be careful are offset by additional functionality. The following explanations are redundant to those contained within Section 7, and are intended solely as an introduction to new capabilities and methods for users of older SINDA/FLUINT versions.

---

* This extra care is required in logic blocks but not in data blocks (e.g., NODE DATA and CONDUCTOR DATA and SOURCE DATA options): the compiler only "sees" logic blocks.

For example, the arguments for SAVE (and TSAVE, SAVEDB, RESAVE etc.) are optional, which was not true in old versions. To demonstrate this point, note that the following are all equivalent:

```
CALL SAVE('all',0)
CALL SAVE('all')
CALL SAVE(0)
CALL SAVE
```

The default use of 'all' extends as well to tabulation, mapping, and printing routines such as:

```
CALL LMPTAB('ALL')
CALL LMPTAB
```

For a few routines with multiple character (string) arguments (e.g., LMPRNT and QMAP), if one argument is missing they both must be:

```
CALL QMAP('all','all',0)
CALL QMAP(0)
CALL QMAP
```

but because the following is ambiguous, *it is not legal*:

```
CALL QMAP('all')   $ illegal!
```

The assumption of 'all' extends as well to RELMOD, HLDTANKS, and RELTANKS but not to HLDLMP or other related routines, where such an assumption is considered rare or dangerous.

Other convenient changes to note include:

- For PSWEEP, STEADY is the assumed process if *procnam* is missing
- DSCANFF auto-calculates resolution if no argument is provided
- USRFIL and USRFIL2 assume STAT='UNKNOWN' if *status* is missing

4.2 Network Solution Routines

This section describes the four network solution routines available. These routines, called exclusively from OPERATIONS (or PROCEDURE or RELPROCEDURE, defined in Section 5), are the central drivers for most of SINDA/FLUINT. They are divided into two types: steady state and transient. Steady-state routines are normally used to find a single time-independent state for the currently built configuration. This state may be the end-product of the analysis, or it may be used for specifying initial conditions for subsequent transient analyses. Transient routines perform integrations of the network governing equations over time, resulting in a series of network states.

The four routines are:

FORWRD	Transient Integration by Explicit Forward Differencing	4.2.1
TRANSIENT*	Transient Integration by Forward-Backward Differencing	4.2.2
STDSTL	Steady State	4.2.3
STEADY†	Fast Initial Conditions (Steady State)	4.2.4

The network solution routines offer four basic combinations of algorithms for determining the transient or steady state solution of a thermal and/or fluid problem. STEADY (“FASTIC”) and STDSTL find the steady-state of the thermal and fluid networks (i.e., the conditions that would exist if the system were unperturbed for a long period of time). TRANSIENT (“FWDBCK”) and FORWRD track the transient changes in the system as a function of time. Each routine is described later in this section. The beginning user should use STEADY and TRANSIENT (which can be referred to as STEADY or TRANSIENT, respectively), reserving STDSTL and FORWRD for specialized needs.

Note that STEADY and STDSTL are *identical* with respect to thermal network solutions, and that TRANSIENT and FORWRD are *identical* with respect to fluid network solutions. This section describes only the solutions of the thermal networks; the methods used to find the simultaneous solutions for fluid networks are described in Appendix E. Matrix inversion details for fluid or thermal networks are described in Appendix F.

The steady state routines STDSTL and STEADY seeks the roots of the equation:

$$\mathbf{Q} = f(\mathbf{T})$$

where:

Q . . . . . Heat rates
T . . . . . Temperature of nodes

The basic equation used for all thermal transient analyses is as follows (neglecting impressed Qs):

$$\mathbf{T}_{new} = \mathbf{T}_{old} + f(\lambda * \mathbf{T}_{new}, (1-\lambda) * \mathbf{T}_{old})$$

* Alias FWDBCK
† Alias FASTIC

where λ is a weighting factor: $0 \leq \lambda \leq 1$

Subroutine FORWRD uses a formulation where the parameter $\lambda = 0$. This method is called first-order *explicit forward differencing*. In formulations where $\lambda \neq 0$, it will be seen that $\mathbf{T}_{new}$ appears on the left and right hand side of the equation. Clearly, these methods require a either a simultaneous or iterative solution. Hence, formulations with $\lambda \neq 0$ are called *implicit* solutions. This is the method used by TRANSIENT, which is also called second-order implicit differencing, trapezoidal integration, or the Crank-Nicholson method.

Users unfamiliar with the control constants in SINDA/FLUINT can ignore the references to them in this section for the first reading. Control constants are described in Section 4.3 and Section 4.4. However, they are included here to assist in the use of this section as a reference source.

Solution routines are sometimes called “execution routines.” They operate on the network defined by the last BUILD and BUILDF commands. Their operation is cumulative: a call to a transient routine after a call to a steady-state routine will initiate the transient solution using the latest network state (i.e., that resulting from the prior steady-state solution) as initial conditions. If such cumulative action is not desired, the SAVPAR, SVPART, and RESPAR parametric routines can be used to save and restore prior states within a run.

Solution routines call VARIABLES 0, VARIABLES 1, VARIABLES 2, FLOGIC 0, FLOGIC 1, FLOGIC 2 and OUTPUT CALLS automatically. These routines were introduced in Section 4.1.

Within all logic blocks except OPERATIONS and PROCEDURE and RELPROCEDURE, the output variable NSOL can be used to determine which solution routine is currently active. This very useful variable identifies the current solution routine, which can be used to alter the instructions in user logic blocks (VARIABLES and FLOGIC) and in user subroutines:

NSOL= 0	STEADY (“FASTIC”)
= 1	STDSTL
= 2	TRANSIENT (“FWDBCK”)
= 3	FORWRD
= 4	(reserved)

This variable can be referenced, but should not be changed. For example, user simulation logic is often different in steady states than it is in transients. A simple check can be used to distinguish between the two:

```
IF (NSOL .LE. 1) THEN
        (steady state logic)
ELSE
        (transient logic)
ENDIF
```

4.2.1 FORWRD: Transient, Explicit

A note to the novice user: FORWRD lacks accuracy-based time step control, and in many cases requires more computational time to achieve the same accuracy. TRANSIENT (Section 4.2.2) should in general be used in preference to FORWRD.

Description: (See also Figure 4-6 and Figure 4-8.) This subroutine performs a transient thermal analysis by the explicit forward differencing* method and fluid analysis by implicit (backward) differencing (see Appendix E for a description of the fluid network methods). First, a heat balance equation is written about a node in forward finite difference form:

$$\frac{C_i}{\Delta t}\left(T_i^{n+1} - T_i^n\right) = Q_i + \sum_{j=1}^{N}\left[G_{ji}\left(T_j^n - T_i^n\right) + \hat{G}_{ji}\left\{\left(T_j^n\right)^4 - \left(T_i^n\right)^4\right\}\right]$$

where:

T_j^n . . . . . temperature of node j at the current time t
T_i^{n+1} . . . . . temperature of node i at the next time t + Δt
G_{ji} . . . . . linear conductor attaching node j to node i (e.g., KA/L)
$\hat{G}_{ji}$ . . . . . radiation conductor attaching node j to node i (e.g., $\sigma A_j F_{ji}$)
C_i . . . . . thermal capacitance of node i (e.g., $V \bullet \rho \bullet C_p$)
Q_i . . . . . source/sink for node i

This equation is the defining equation for diffusion nodes. As this equation is solved for node i, the heat flowing through the attached conductors is evaluated and used to adjust the source/sink on node j. (This heat is neglected for any one-way conductors in which node i is the upstream node.) The equation presented above uses only the temperature derivatives at the current time to predict the overall temperature change. This method is also called *Eulerian*, and is first-order accurate in time and space.

Because this equation can be solved explicitly for T_i^{n+1}, there is a limitation imposed upon the maximum time step due to stability considerations: the upper bound is the smallest of the ratios of the thermal capacitance to the sum of the linearized conductors ($C/\Sigma G$) for every diffusion node in the network. The term that is added in the sum of conductor for a radiation conductor to linearize it is:

$$\sigma A_i F_{ij}(T_i^2 + T_j^2)(T_i + T_j)$$

* The term “finite difference” in this section refers to the integration in time, and does not preclude finite element methods from being used to compose spatial terms (C, G).

For time steps in excess of this minimum ratio (called CSGMIN), the network temperatures may oscillate or diverge without limit. To prevent this, the routine computes CSGMIN and bases the time step on it. As the time step deviates from an infinitesimal quantity, so will the error imparted to the resulting temperatures. In many practical thermal applications, this time step criterion has been found to be adequate when used in combination with limits on the temperature change per time step (governed by DTMPCA as explained below).

Note that stability and accuracy are not related, and that keeping the time step less than CSGMIN guarantees stability but not accuracy. Therefore, use of TRANSIENT in combination with automatic time step control (DTIMEI=0.0) is recommended, since that time step controller is based on integration accuracy. There is no analogous error control available in FORWRD.

Another point to remember is that since the heat balance equation was formulated using the temperatures of nodes i and j at the start of the time step (t), those boundary conditions that are functions of time should use the old time (TIMEO) for consistency.

The VARIABLES 1 and VARIABLES 0 operations are performed once each time step before the diffusion node temperature calculations and the VARIABLES 2 operations are performed after the relaxation of the arithmetic nodes at the end of the time step. The OUTPUT CALLS are performed in increments of OUTPUT starting from the input value of TIME0.

The time step is computed first as DTIMEU = CSGMIN * 0.95/CSGFAC. (If fluid submodels are active, a maximum value of DTIMEU has already been calculated for this interval.) The value of CSGMIN used is that computed from the last integration step. A check is made after the calculation of the last diffusion node to determine if the time step used (DTIMEU) is less than the time step just computed. If not, then DTIMEU is adjusted downward and the diffusion node temperatures are recalculated. Values of CSGFAC less than one are ignored. The user may input a larger value if desired. DTIMEU is set to the smallest of the thermal and fluid (DTIMUF) time steps if fluid submodels are active.

In addition, a “look ahead” feature has been incorporated in the routine to alter the value of DTIMEU. If one time step would exceed the output time point, the time step is adjusted so it ends exactly on the output time point. In addition, the user may explicitly control how large or how small DTIMEU may be by using DTIMEH and DTIMEL. The routine checks to see if DTIMEU > DTIMEH; if so, then DTIMEU is set equal to DTIMEH; if DTIMEU < DTIMEL, an error message is printed and the problem terminates after calling OUTPUT CALLS. If no input values are specified, DTIMEH is set to infinity and DTIMEL is set to zero.

The user may also explicitly control the maximum diffusion node temperature change over a time step (and implicitly control the time step). The largest temperature change for the set of diffusion nodes is stored in DTMPCC with the correct sign. A check is made to insure that |DTMPCC| < DTMPCA. If not, DTIMEU is adjusted downward and the diffusion node temperatures are recalculated (without repeating fluid calculations). This check is always performed at the end of the diffusion node calculations. If no input value is specified, DTMPCA is set to 1.0E30.

After the diffusion node temperatures are calculated, the set of arithmetic nodes is then iterated to a steady-state solution. The diffusion node temperatures just calculated are then considered as boundary conditions for the relaxation of this set of arithmetic nodes. The arithmetic nodes are, by definition, steady-state nodes. The heat balance equation for node i on the k+1st iteration is:

$$0 = Q_i + \sum_{j=1}^{i-1} \left[G_{ji} \left(T_j^{k+1} - T_i^{k+1} \right) + \hat{G}_{ji} \left\{ \left(T_j^{k+1} \right)^4 - \left(T_i^{k+1} \right)^4 \right\} \right]$$
$$+ \sum_{j=i}^{N} \left[G_{ji} \left(T_j^{k} - T_i^{k+1} \right) + \hat{G}_{ji} \left\{ \left(T_j^{k} \right)^4 - \left(T_i^{k+1} \right)^4 \right\} \right]$$

where the superscript refers to the iteration.

This gives rise to a set of simultaneous equations which are solved in either a simultaneous (matrix inversion), or an iterative manner. For nonlinear problems, which are extremely common, even a simultaneous solution requires some iteration. The user must specify the convergence criterion (ARLXCA) for the solution and the maximum number of iterations (NLOOPT) to be performed. The largest temperature change from one iteration to the next is stored in ARLXCC during each iteration. Iterations will be performed until |ARLXCC| < ARLXCA or until the number of iterations (LOOPCT) is greater than NLOOPT. If there are arithmetic nodes in the problem, both ARLXCA and NLOOPT must be specified. To accelerate the convergence, a variation of Aitken's del-squared process is performed by default every third (or ITERXTth) iteration. This acceleration is only performed on those nodes that have exhibited a monotonic tendency over the previous three iterations, otherwise the acceleration actually works as a damping action. The user can control the change in the temperatures imparted by this acceleration process by use of ITERXT and EXTLIM. For example, values of ITERXT less than 3 have different meanings, invoking alternate acceleration schemes. If LOOPCT = NLOOPT, a warning message is printed and the problem continues.

Once the arithmetic node temperatures have been calculated, the largest temperature change over the time step is stored in ATMPCC. A check is made to insure that |ATMPCC| < ATMPCA; if not, DTIMEU is adjusted downward and both the diffusion and arithmetic node temperatures are recalculated. If no input value is specified, ATMPCA is set to 1.0E30. Note that the ATMPCA criterion can fail again on the second pass due to the fact that arithmetic node temperature changes can be time-independent. To avoid an infinite loop, the program will not attempt to reduce the time step twice in a row for this criterion, with the result that |ATMPCC| may occasionally be larger than ATMPCA. For this reason, the user should rely on DTMPCA rather than ATMPCA whenever possible.

The control constant BACKUP is tested for a non-zero real value after the calls to perform the VARIABLES 0, VARIABLES 1 and VARIABLES 2 operations. A non-zero real value in the former case will cause a recomputation of DTIMEU, TIMEN, and TIMEM. DTIMEU will again be compared against DTIMEH and DTIMEL and DTIMUF. BACKUP is then zeroed, the nodal Q values are returned to their SOURCE DATA values, and the complete time step solution is performed again. The user's logic should ensure that BACKUP is not perpetually set, and that some achievable criterion is used to allow the program to continue. At this point, the final fluid submodel calculations

have not been completed; fluid submodels do not recognize the BACKUP command. Fluid calculations are completed upon successful completion of the thermal calculations using the final time step value.

The control constant OPEITR is tested for a non-zero value after the completion of the VARIABLES 2 operations. If non-zero it will cause the OUTPUT CALLS operations to be performed. OPEITR is thus used to cause output at time points other than the normal output time points.

Restrictions: Values must be assigned to control constants TIMEND and OUTPUT and/or OUTPTF. There must be at least one diffusion node in the network unless fluid networks are present, in which case at least one boundary node must be present. The problem start time, TIMEO, will be zero unless it has been reset by the user or another transient routine has been called earlier.

Calling Sequence:

```
CALL FORWRD
```

4.2.2 TRANSIENT ("FWDBCK"): Transient, Implicit Second Order

Description: (See also Figure 4-5 and Figure 4-9.) This subroutine performs a transient thermal analysis by implicit forward-backward differencing and fluid analysis by implicit backward differencing (see Appendix E for a description of the fluid network methods). One heat balance equation is written about a diffusion node as a forward finite difference* equation and another as a backward finite difference equation. The resulting sum of these two equations is:

$$\frac{2C_i}{\Delta t}\left(T_i^{n+1} - T_i^{n}\right) = 2Q_i + \sum_{j=1}^{N}\left[G_{ji}\left(T_j^{n} - T_i^{n}\right) + \hat{G}_{ji}\left\{\left(T_j^{n}\right)^4 - \left(T_i^{n}\right)^4\right\}\right]$$
$$+ \sum_{j=1}^{N}\left[G_{ji}\left(T_j^{n+1} - T_i^{n+1}\right) + \hat{G}_{ji}\left\{\left(T_j^{n+1}\right)^4 \left(T_i^{n+1}\right)^4\right\}\right]$$

where:

T_j^n ..... temperature of node j at the current time t
T_i^{n+1} ..... temperature of node i at the next time t + Δt
G_{ji} ..... linear conductor attaching node j to node i (e.g., KA/L)
$\hat{G}_{ji}$ ..... radiation conductor attaching node j to node i (e.g., $\sigma A_j F_{ji}$)
C_i ..... thermal capacitance of node i (e.g., $V \bullet \rho \bullet C_p$)
Q_i ..... source/sink for node i

In effect, the equation presented above uses the average of the temperature derivatives at the current and next times to predict the overall temperature change. This method is also called *trapezoidal,* and is second order accurate in time and first order accurate in space. When applied to an orderly finite difference mesh of diffusion nodes and linear conductors, it represents the Crank-Nicholson approximation to the partial differential equation for conduction.

For an arithmetic node, the heat balance equation is, as always:

$$0 = Q_i + \sum_{j=1}^{i-1}\left[G_{ji}\left(T_j^{k+1} - T_i^{k+1}\right) + \hat{G}_{ji}\left\{\left(T_j^{k+1}\right)^4 - \left(T_i^{k+1}\right)^4\right\}\right]$$
$$+ \sum_{j=i}^{N}\left[G_{ji}\left(T_j^{k} - T_i^{k+1}\right) + \hat{G}_{ji}\left\{\left(T_j^{k}\right)^4 - \left(T_i^{k+1}\right)^4\right\}\right]$$

* The term "finite difference" in this section refers to the integration in time, and does not preclude finite element methods from being used to compose spatial terms (C, G).

The preceding equations give rise to a system of equations that describe the network. These equations, being implicit in T^{n+1}, are solved either by iterative relaxation or by simultaneous (matrix) solution until diffusion node temperatures change less than DRLXCA and arithmetic nodes change less than ARLXCA, and until the energy imbalances are tolerable (|FBEBALC| < FBEBALA), as described below.

Unlike the explicit method, there is no upper bound on the time step *due to stability considerations* for this implicit method. However, the user must realize that as the time step becomes larger, so will the *error* imparted to the resulting temperatures. Automatic time step prediction to control this error is available by default (DTIMEI=0.0) as described below.

Another point to remember is that since the heat balance equation was formulated using the temperatures of node i and node j at both the start (t) and the end of the time step ($t+\Delta t$), to remain consistent, those boundary conditions that are functions of time should use a mean value of these two time points (control constant TIMEM).*

By default (NVARB1=0), the VARIABLES 0 and VARIABLES 1 operations are performed once per time step. If NVARB1=1 for any submodel, then the VARIABLES 1 block for that submodel is called once per iteration step during the calculation of the new temperatures in a manner analogous to STDSTL or STEADY iterations. In other words, the VARIABLES 1 block will be called several times per time step if NVARB1=1. The VARIABLES 2 operations are performed after the iterations at the end of the time step. The OUTPUT CALLS operations are performed in intervals of OUTPUT starting from the input value of TIMEO.

The time step used in the calculation of the diffusion nodes is determined from the input value of DTIMEI. If DTIMEI = 0.0 (the highly recommended and defaulted choice), the time step is determined from an internal time-step calculation made after each time increment based on truncation errors† estimated from the trends in diffusion node temperatures. The control constant DRLXCA is used as a measure of the acceptable truncation error. Subject to other constraints (e.g., DTIMEH and output intervals), this time step is arbitrarily limited to the range (0.1*CSGMIN,1000*CSGMIN) due to historical reasons. *It should be noted that a time step smaller than 0.1*CSGMIN may be needed to maintain accuracy in some models*, requiring user intervention to permit an even smaller time step or to accept a larger error. The program will warn the user if it cannot achieve the required accuracy due to other time step constraints, or if the solution is being repeated too many times in an attempt to maintain the accuracy requirement or to avoid nonconvergence. These warnings are designed to occur less and less frequently, such that the user is duly notified but not annoyed. If such warnings occur, the user can either accept the solution, decrease DTIMEH to retain the desired accuracy, or accept greater error by increasing DRLXCA. If the user chooses a value of DTIMEH smaller than 0.1*CSGMIN, automatic time step control will be suspended, except that a periodic check (about every 7 time steps or so) will be made to warn the user if the accuracy criteria are still not being met.

---

* In theory, boundary functions should use an average of their values at TIMEO and TIMEO+DTIMEU, but a single value at TIMEM is a reasonable and convenient approximation.

† Since TRANSIENT is second order accurate in time, meaning that truncation errors are the third-order (proportional to Δt^3). It should be noted that three time steps are required before automatic time step control can begin, since each calculation of truncation error is based on the prior three steps.

As an alternative to automatic time step control, positive values of DTIMEI may be specified directly as the time step. Negative values of DTIMEI are a signal that the time step should be calculated as DTIMEU = CSGMIN/CSGFAC. *Nonzero values of DTIMEI are discouraged unless the answers are checked against additional runs that employ a much smaller time step.*

The predicted time step, DTIMEU, is subject to the following limiting adjustments. DTIMEU is reset if it is larger than the smallest fluid submodel time step (which is used if there are no thermal submodels). DTIMEU may be adjusted by a "look ahead" feature; that is, if a time step would exceed the output time point, the time step is adjusted to come out exactly on the output time point. In addition, the user may place limits on how large or how small DTIMEU is by using DTIMEH and DTIMEL. Then, if DTIMEU is greater than DTIMEH, DTIMEU is set equal DTIMEH; if DTIMEU is less than DTIMEL, then an error message is printed and the problem terminates. If no input values are specified, DTIMEH is set to 1.0E30 and DTIMEL is set to zero.

If MATMET=0) is chosen, then all temperature solutions are performed using iterative methods. If, for any thermal submodel, MATMET>0 is specified, then the first step is performed using matrix (simultaneous solution) methods, and any subsequent steps are performed using purely iterative methods. MATMET>0 is recommended for submodels that have unavoidably large conductors or that are dominated by thermal conduction, and which do not have complex sources or strong temperature dependencies. These are the default methods.

The largest temperature change from one iteration to the next for the diffusion nodes is stored in DRLXCC; that for the arithmetic nodes is stored in ARLXCC. Iterations will continue to be performed until |DRLXCC| < DRLXCA and |ARLXCC| < ARLXCA. DRLXCA and ARLXCA are both defaulted to 0.01 degrees (and are therefore unit dependent). Both criteria must be met before the relaxation is considered finished, as well as an energy balance check (|FBEBALC| < FBEBALA). If the iteration counter LOOPCT is greater than NLOOPT, a warning message is printed and that relaxation is considered complete. Experience has shown that LOOPCT in excess of about 10 is a sign that the time step size should be reduced.

To accelerate the convergence of the nodal temperatures, Aitken's del-squared process is used every ITERXTth iteration by default. This consists of determining the limit of the sequence of the last three successive temperatures for a node. This acceleration process is performed only on those nodes whose temperatures were monotonically increasing or decreasing over the previous three iterations, otherwise the acceleration actually performs damping. The user can limit the change in the temperatures imparted by this process by the use of ITERXT and EXTLIM. For example, values of ITERXT less than 3 have different meanings, invoking alternate acceleration schemes.

After the temperature solution is complete, the largest temperature changes over the time step are stored for each submodel. Those for the diffusion nodes are stored in DTMPCC, and those for the arithmetic nodes are stored in ATMPCC. A check is made to determine whether |DTMPCC| < DTMPCA and |ATMPCC| < ATMPCA are satisfied for each submodel. If not, DTIMEU is adjusted downward and the relaxation process is started from the beginning. If DTMPCA and/or ATMPCA are not specified, they will be set to 1.0E30. Note that the ATMPCA criterion can fail again on the second pass due to the fact that arithmetic node temperature changes can be time-independent. To

avoid an infinite loop, the program will not attempt to reduce the time step twice in a row for this criterion, with the result that |ATMPCC| may occasionally be larger than ATMPCA. For this reason, the user should rely on DTMPCA rather than ATMPCA whenever possible.

The VARIABLES 2 operations will be performed only after the successful check on the maximum temperature change over the time step.

After the calls to perform the VARIABLES 2 operations, the value of control constant BACKUP is tested. If BACKUP is a non-zero real value after the call for the VARIABLES 2 operations, the calculated temperatures will be replaced with their values at the start of the time interval, and the time step solution will be repeated. The user's logic should ensure that BACKUP is not perpetually set, and that some achievable criterion is used to allow the program to continue. At this point, the final fluid submodel calculations have not been completed; fluid submodels do not recognize the BACKUP command. Fluid calculations are completed upon successful completion of the thermal calculations using the final time step value.

After the VARIABLES 2 operations, the value of control constant OPEITR is tested. If it is non-zero, the OUTPUT CALLS operations will be performed. OPEITR is thus used to cause output at each time point.

Restrictions: There must be at least one diffusion node in the network if no fluid submodels are present. Values *must* be assigned to control constants TIMEND and OUTPUT. There must be at least one diffusion node in the network unless fluid submodels are present, in which case at least one boundary node must be present. A problem may not start at a time other than zero unless control constant TIMEO is appropriately set.

Calling Sequence:

```
CALL TRANSIENT
CALL FWDBCK            $ NICKNAME (old name)
```

4.2.3 STDSTL: Steady State (Fluid Pseudo-transient)

A note to the novice user: STDSTL is useful only in limited cases involving certain aspects of fluid flow models. STEADY ("FASTIC," Section 4.2.4) should otherwise be used in preference to STDSTL. However, since they treat thermal submodels identically, most of the documentation of these methods is listed in this first section and not repeated in the STEADY description.

Description: (See also Figure 4-7 and Figure 4-10.) This subroutine calculates the steady-state solution of a fluid/thermal network (see Appendix E for the fluid solution methods, which are the same for STDSTL as they are for FORWRD and TRANSIENT; see STEADY for an alternate fluid steady state approach). By default, the diffusion nodes and arithmetic nodes are solved by a "successive point" iterative method, meaning that the temperature for each diffusion and arithmetic node is adjusted such that energy flows through it are in balance. Alternatively, if MATMET=1 or 11 then the set of diffusion and arithmetic nodes in any submodel can be advanced simultaneously using a matrix (simultaneous) solution. All thermal submodels with MATMET=2 or 12 are solved together in a single matrix. (MATMET=12 is the default.)

One simultaneous solution or one iterative pass through all such nodes is called an iteration.

For the purely iterative method, the heat balance for the i^{th} node on the $k+1^{st}$ iteration is as follows:

$$0 = Q_i + \sum_{j=1}^{i-1} \left[G_{ji} \left(T_j^{k+1} - T_i^{k+1} \right) + \hat{G}_{ji} \left\{ \left(T_j^{k+1} \right)^4 - \left(T_i^{k+1} \right)^4 \right\} \right]$$
$$+ \sum_{j=i}^{N} \left[G_{ji} \left(T_j^{k} - T_i^{k+1} \right) + \hat{G}_{ji} \left\{ \left(T_j^{k} \right)^4 - \left(T_i^{k+1} \right)^4 \right\} \right]$$

where the superscript refers to the iteration count. The reader will recognize this equation as the equation used to find the temperatures of arithmetic nodes in transients. In STDSTL and STEADY, diffusion nodes are treated like arithmetic nodes.

When a model is updated using the default simultaneous methods, radiation terms are linearized and a sparse matrix solution (Appendix F) is employed. MATMET>0 is recommended for submodels that have unavoidably large conductors or that are dominated by thermal conduction, and which do not have complex sources or strong temperature dependencies. (This is the default.) If the system is perfectly linear, a single iteration is theoretically required. However, because of user logic and other complexities, two or three iterations may be required. If submodel-to-submodel conductances are large and/or nonlinearities are strong, several tens of iterations may be required, each performed using a simultaneous solution. Thus, the convergence acceleration discussion that follows applies to both iterative and simultaneous means of updating nodal temperatures.

To accelerate the convergence of the solution, a variation of Aitken's del-squared process is used by default (when ITERXT is equal to 3 or more). This process consists of performing ITERXT iterations and then determining the limit of the resulting temperature sequence. If this nodal temperature sequence is not monotonically increasing or decreasing, the acceleration process actually results in damping.* The user can control the change in the temperature imparted by this process by use of ITERXT or EXTLIM, which is the maximum change to apply as a result of this "extrapolation" process.

As an alternative to the extrapolator, the user choose to apply over- or underrelaxation parameter ω to the prediction of temperatures during the successive update of each node's temperature:

$$T_{new} = T_{old} + \omega*(T_{new} - T_{old})$$

When ITERXT is zero, the user provides a value for ω by setting it in EXTLIM (one value per submodel in which ITERXT=0). *EXTLIM therefore assumes a new meaning when ITERXT=0.* A value of ω greater than 1.0 (overrelaxation) accelerates the temperature changes in the models at the risk of nonconvergence in the event predictions oscillate, especially in cases with extensive user logic manipulations. A value of ω less than 1.0 is useful to forcing convergence when the predictions are oscillating or are unstable, but may slow convergence or cause STDSTL to stop prematurely due to lack of progress.

When ITERXT is 2, the code calculates the value of ω itself based on the following formula:

$$\omega = 2 / (1 + (1-\rho^2)^{1/2})$$

where ρ is the ratio of the current maximum temperature change, $|T_{new}-T_{old}|$, over the value from the previous iteration. This method, which always over-relaxes, provides a convenient alternative to the default extrapolator. It is especially useful in conduction-dominated problems with smooth user logic and properties that do not vary greatly with temperature. (These are precisely the type of problems in which matrix methods, MATMET>0, excel, so they do not need acceleration.) A slightly more aggressive method is applied when ITERXT=1, in which case the ρ term in the above equation is not squared, increasing the value of ω. EXTLIM is ignored when ITERXT=1 or ITERXT=2.

The user must specify the maximum number of iterations to be performed (control constant NLOOPS) in relaxing to steady state. He or she must also select the relaxation criteria that specify when the problem is at steady state. These criteria can consist of either the largest of the absolute values of the nodal temperature changes per iteration (DRLXCA for diffusion nodes and/or ARLXCA for arithmetic nodes), or the thermal energy balance values (EBALSA and/or EBALNA), or both. If the largest temperature change is used, the routine iterates until the criterion is met (or until both criteria are met if both types of nodes are present). The largest nodal temperature change for diffusion nodes is stored in DRLXCC; that for arithmetic nodes is stored in ARLXCC. The temperature change criterion is considered achieved when |DRLXCC| < DRLXCA and |ARLXCC| < ARLXCA for each submodel.

* Because each node is individually accelerated or dampened, this process is much more efficient than the over- and underrelaxation factors used in older versions of SINDA, which were applied blindly to all nodes. In effect, the program is both calculating and applying optimal over- and underrelaxation factors to each node.

After the temperature change criteria have been satisfied, if the control constant EBALSA is zero for each submodel, the relaxation iterations are terminated and the VARIABLES 2 and OUTPUT CALLS operations are performed. If EBALSA is greater than zero, then more iterations will be performed until the relative error in all energy flows to and from the entire model (available in the output variable EBALSC) is less than EBALSA. An analogous node-by-node check is performed for nonzero EBALNA based on absolute error in the energy flow for each node—*EBALNA has units of power whereas EBALSA is unitless*. The value against which EBALNA is being compared is available in the output variable EBALNC. Fluid submodels must also converge according to the RERRF and REBALF criteria. When all active criteria are met, a message is printed and the OUTPUT CALLS operations are performed.

Note that the temperature-change relaxation criterion must not be considered as a measure of the accuracy associated with the final temperature results: rather, it is nothing more than the maximum temperature change from one iteration to the next. Even though the temperature change per iteration tends to approach zero as the network approaches a steady-state condition, the magnitude of this number is not a sufficient condition for equilibrium. The necessary and sufficient condition for a steady state is that the energy entering each node equals the energy leaving that node, which implies that the energy flows at the network level are balanced.

VARIABLES 0, VARIABLES 1 and OUTPUT CALLS operations are performed initially to provide initial conditions, and the VARIABLES 1 operations are performed every iteration until the converged solution is achieved. Upon meeting convergence criteria both the VARIABLES 2 and OUTPUT CALLS operations are performed. Control constant ITEROT is checked at the end of each iteration. If LOOPCT is non-zero and an integer multiple of ITEROT, then OUTPUT CALLS operations are performed. If the iteration count LOOPCT exceeds NLOOPS, a message is printed and VARIABLES 2 and OUTPUT CALLS operations are performed.

“ENERGY STABLE BUT NOT BALANCED” is a message that sometimes appears at the conclusion of an otherwise apparently successful STDSTL (or STEADY) run. This message means that the EBALSA criteria was not strictly met, but that the program was unable to detect significant progress towards full compliance of that criteria: the solution was stalled out. This is usually caused by one of four factors described below along with recommended user actions:

1. One-way conductors are present in the model. If this is true, then the final solution has likely been produced. One way conductors “fool” a system-level energy check because they break symmetry: some nodes exist from the point of view of others, but that viewpoint is not reciprocal, so energy might seem to appear or disappear. This results in a energy inflow that will never equal an outflow, despite the fact that a valid solution has been produced. To be sure, use the EBALNA (node-level) criteria in addition to the EBALSA (model-level) criteria.

2. If fluid submodels are active, unmatched duplication factors (DUPN, DUPL, DUPI, and DUPJ) are mathematically analogous to one-way conductors, and result in analogous problems with system-level energy balance checks. Once again, the resulting answer is probably fine, but use a lump-by-lump (negative REBALF) check to be sure.

3. If fluid submodels are active, and if Mach numbers are high enough, then discontinuities in flow area (AF) can cause kinetic energy to appear or disappear from the network. Warnings at the start of the processor output file should be reviewed and flow areas (including perhaps AFI, AFJ) adjusted accordingly.

4. Excessively large conductances exist in the model, causing the default solution to stall out as temperature changes per iteration become imperceptible, and the internal extrapolator cannot distinguish trends. In this case, the program is warning that convergence is proceeding very slowly, and that many more iterations will be required to completely converge. The user has the choice of (1) reducing the convergence criteria (DRLXCA, ARLXCA, and/or EBALSA) and perhaps increasing NLOOPS, and then repeating the run for a better answer, (2) reducing excessively large conductances by reducing spatial resolution and/or combining adjacent nodes, (3) trying ITERXT=1 or ITERXT=2, (4) using MATMET>0 (MATMET=12 is the default) to invoke a simultaneous solution, or (5) accepting the resulting errors as adequate. Solutions (1), (3), and (4) may be used concurrently, but only (2) is a complete solution.

CPU Optimization with Multiple Submodels—STDSTL can take advantage of the thermal submodel organization to reduce the number of calculations (*but not total iterations)* required to achieve steady state. In essence, more CPU time can be spent where it is needed most. Those submodels meeting the most convergence criteria (temperature change, energy balance or both) are put in a dormant condition while iterations are performed on the remaining submodels. In the dormant state, a submodel's temperatures act as boundary temperatures for the remaining submodels. After ITHOLD iterations (default=0) in the dormant state, a dormant submodel is reactivated and it remains in the iteration loop until it again meets the convergence criteria. When the last submodel converges, all the dormant submodels are reactivated for at least one iteration loop in order to verify that convergence has indeed been achieved. Note that the use of nonzero ITHOLD may cause the final value of LOOPCT to appear higher than if ITHOLD were zero. This may give the illusion of having taken more computer time; the actual computer time will likely have been reduced because of dormant submodels. In fact, this is the sole purpose of ITHOLD.

DTIMES Option—If a number of different steady-state solutions are required, the control constant TIMEND is used. TIMEN is incremented by DTIMES when a problem terminates whether it is due to achieving the relaxation criterion or to exceeding the maximum number of iterations. TIMEN can be used in the VARIABLES 0 or VARIABLES 1 blocks to make boundary conditions a function of this pseudo time. If TIMEND is greater than TIMEN, the routine will continue to perform steady-state solutions until TIMEN equals TIMEND.

If desired, the STDSTL call can be followed by a call to one of the transient solution subroutines. In this manner, the steady-state solution becomes the initial condition for the transient analysis. Because of this typical usage, if TIMEND is greater than TIMEO yet DTIMES is zero, the program will perform one steady state solution at TIMEO assuming that the value for TIMEND is intended for a later call to a transient routine.

The DTIMES option represents a quick way to perform parametrics (see also Section 4.7.2 and Section 4.9). The user can use the problem time to represent a problem variable (diameter of a line, emissivity of a coating, conductance of a contact joint, etc.) In other words, the user replaces the meaning of "time" with some other parameter. Then, by either using "time"-varying conductor or

source options or other user logic, a single call to STDSTL (or STEADY) results in a series of parametric steady-state analyses. Refer to Sample Problem D for an example of such usage. This method has the benefit of enabling plotting of results as functions of “time” in Sinaps or other software. Unfortunately, negative problem times are not supported.

Calling Sequence:

```
CALL STDSTL
```

4.2.4 STEADY ("FASTIC"): Steady State

Description: (See also Figure 4-7, Figure 4-10, and Appendix E.) This routine finds a steady-state solution for thermal and fluid networks. *STEADY performs exactly the same calculations for thermal networks as does STDSTL, so that discussion (Section 4.2.3) will not be repeated here.* Fluid calculations in STEADY are fundamentally different from those in any other solution routine. The fluid network methods for other routines are described in Appendix E.

As the name implies, STEADY is intended primarily to find initial conditions for fluid submodels. Because of the primary FLUINT solution methods, rough initial conditions are equivalent to severe transient events which would require many time steps and iterations. STEADY takes rough-guess initial conditions and "smooths" them into a consistent set. Subsequent calls to other solution routines can then concentrate on real transients. STEADY can also be used as the primary solution routine.

STEADY performs a successive point relaxation similar to the solution method in STDSTL for thermal networks. A simultaneous hydraulic (pressure/flow rate) solution is performed first in each iterative pass, followed by iterative energy (enthalpy/heat rate/species concentration) solutions. Hydraulic solutions are always performed with a simultaneous one-pass method unless the matrix is ill-formed, and energy solutions are always completely iterative with two to three passes per iteration. If twinned paths are active, or if EI or EJ is nonzero for any path, then the simultaneous hydraulic solution becomes a coupled hydraulic and energy solution similar to that used in other solution routines.

In general, STEADY initially approaches a steady-state solution in fewer iterations than STDSTL. Not all modeling options are possible with STEADY, as noted below. Most models are served best by calling STEADY instead of STDSTL, but a few models may require the *subsequent* use of STDSTL or a steady (unchanging, time-independent) subsequent call to TRANSIENT. All STDSTL control constants apply to STEADY except RSMAXF and RSTUBF. RSSIZF is used only in a limited capacity. The defaults are the same for both routines.

Both STEADY and STDSTL take cautious first steps or preliminary iterations. The purpose of these cautious steps is to absorb inconsistencies and begin to approach a solution monotonically without falsely exceeding the property range limits. Convergence is not allowed during the preliminary iterations. Because of this, two consecutive calls to steady-state routines will require at least 2 iterations in the second call to confirm convergence found in the first. STDSTL usually converges immediately on a solution achieved by STEADY, but not always. The reasons for this apparent discrepancy include the fact that STDSTL supports all transient options (see below) whereas STEADY does not, and that STDSTL performs a simultaneous energy balance that immediately sweeps out cumulative errors left by STEADY iterative energy methods. Also, many models are simply unsolvable in STDSTL even after having been solved in STEADY, since STDSTL retains full transient hydraulics but neglects thermal masses: instant changes in a thermal model are often a poor assumption in parallel with fully transient hydraulics.

The routines HLDLMP, HTRLMP, and RELLMP can be used to control the results of STEADY (as well as other solutions, see Section 7.11.1.4). By making a tank or junction act temporarily as a plenum or *reference lump* with HLDLMP, the user should be able to customize the solution by

isolating parts of the network. For example, by holding the inlet and outlet of a subnetwork constant, the solution of the subnetwork becomes independent of the rest of the network. Similarly, HTRLMP can be used to hold the enthalpy of a lump.

Restrictions: During a STEADY solution, tank states and tube flow rates may change instantly—*in every way, tanks are treated like junctions and tubes are treated like STUBE connectors*. This imposes certain restrictions in program usage. First, VDOT and COMP terms are ignored, and twinned tanks are treated as a single combined junction. Second, ifaces are ignored. Third, species-specific suction is ignored. Fourth, at lease one lump must be either a plenum or a reference lump (state held by a previous call to HLDLMP). If the model contains no plena and no lump state has been held, a fatal error will result. Fourth, customizing QL values for tied lumps and many tube/STUBE values (e.g., FC, FPOW, AC, etc.) is not possible—changes to these parameters in FLOGIC 1 blocks are ignored. However, QL values for other lumps can be altered as with other solution routines.

Calling Sequence:

```
CALL STEADY
CALL FASTIC            $ EQUIVALENT (OLD NAME)
```

4.3 CONTROL DATA: Control Constant Input

CONTROL DATA blocks are used to exert user control over the problem solution by initializing a set of control "constants" that the program stores in memory under reserved variable names. The quotation marks are to indicate that these values are not constant at all, and *most* (but not all) may be initialized or changed at will in the user's logic blocks or via their defining expressions—the historical name "constants" is therefore a misnomer. Examples of these variables are TIMEND and NLOOPS, the problem stop time and maximum steady state iteration count allowed, respectively.

Because SINDA/FLUINT models can consist of more than one submodel, control constants of two forms: single-valued global control constants, and arrays of control constants with one constant per submodel. Single-valued constants apply to all submodels, such as ABSZRO. Multiple value control constants are arrays of values N long where N is the number of currently active* submodels. Examples of such constants include OUTPUT and OUTPTF, which are actually referenced internally (e.g., after logic translations) as OUTPUT(nt) and OUTPTF(nf), where nt is the thermal submodel sequence number and nf is the fluid submodel sequence number. If the same value is to apply to all submodels of the same type (i.e., thermal or fluid), that value may be initialized in CONTROL DATA, GLOBAL. This global value can then be overridden for each submodel if desired, as shown below. However, once processor execution begins, the control constants for each submodel are independent of each other however they were initialized in the data blocks, unless they were defined using the same register or register-containing expression.†

4.3.1 Defining CONTROL Data Values

Global control constants are entered under a single HEADER CONTROL DATA, GLOBAL record. *Note that this precludes naming a submodel 'GLOBAL'*. Any control variable name in Table 4-2 may be initialized in this block by a single statement. For instance, in a model consisting of five thermal submodels, the single statement OUTPUT = 0.1 would set five output time interval array elements to 0.1. Later, under a HEADER CONTROL DATA, FRED record, the statement OUTPUT = 0.01 would set the output time interval to 0.01 for submodel FRED. *Note that the GLOBAL block must precede the other control data blocks* because the last input overrides all previous inputs.

4.3.2 CONTROL Data Names and Functions

Table 4-2 is a list of all the SINDA/FLUINT program control "constants," a brief description of their function, whether they may be multi-valued (arrays) or not, and their defaults, if any. It is only necessary for the user to define all non-defaulted values if both steady-state and transient solutions are required. The use of some of the control constants can be quite involved and subtle, and usually dependent on their use with particular solution routines. The user is referred to Section 4.4 for more information on this subject.

---

* An active submodel is one whose name appears on a BUILD or BUILDF macroinstruction.

† In fact, registers represent a convenient means of manipulating the control constant values of multiple submodels during processor execution using a single parameter.

Table 4-2 Control Constant Definitions

Name	Definition	Options	Default
*Single-Valued Constants: Input under submodel **'GLOBAL'** only*			
TIMEO	Initial (problem start) time	Real	0.0
TIMEND	Stop (problem end) time	Real > TIMEO	**NONE**
NLOOPS	Maximum iteration count allowed (steady state)	Positive integer	1000
DTIMES	Pseudo time step (steady state)	Real	0.0
ABSZRO	Value of absolute zero in user's temperature units	Real[a]	-459.67 (ENG) -273.15 (SI)
PATMOS	Value of absolute zero in user's pressure units (FLUINT)	Real[a]	0.0
SIGMA	Stefan-Boltzmann constant	Positive real[a]	1.0
UID	Unit system identifier	SI or ENG (note: no quotes)[a]	ENG (English units)
ACCELX/Y/Z	Body force acceleration vector components in X, Y and Z directions	Real	0.0
*Multiple-Valued Constants: Input one data value per **thermal** submodel*			
ARLXCA	Maximum temperature change allowed for convergence of arithmetic nodes (degrees)	Positive real	0.01
NLOOPT	Maximum iteration count allowed (transient)	Positive integer	100
DRLXCA	Maximum temperature change allowed for convergence of diffusion nodes, and integration truncation error (degrees)	Positive real	0.01
EBALSA	Steady state submodel energy balance required for convergence: fraction of total model energy flow	Positive real 0.0 to 1.0	0.01
EBALNA	Steady state node energy balance required for convergence: maximum nodal imbalance in units of power	Positive real 0.0 = off	0.0
ATMPCA	Maximum temperature change allowed per time step for arithmetic nodes (degrees)	Positive real	1.0E30
DTMPCA	Maximum temperature change allowed per time step for diffusion nodes (degrees)	Positive real	1.0E30
DTIMEL	Minimum allowable time step (transient).	Positive real	0.0
DTIMEH	Maximum allowable time step (transient)	Positive real	1.0E30
DTIMEI	Specified time step (transient)	Positive real	0.0 (automatic time step calc.)
ITEROT	Iteration interval for OUTPUT CALLS (steady state)	Positive integer	0 (print upon convergence)
OUTPUT	Time interval for OUTPUT CALLS (transient)	Positive real	0.01*TIMEND
ITHOLD	Number of iterations a submodel can remain in semi-converged boundary state (steady state)	Positive integer	0
EXTLIM	Maximum allowable extrapolation temperature change if ITERXT is 3 or more, else the over/under-relaxation factor if ITERXT=0	Positive real	5.0
ITERXT	Number of iterations between extrapolation if 3 or more. Signals user-input over/under-relaxation factor if zero, or program-calculated over-relaxation factor if 1 or 2.	Integer, nonnegative	3
CSGFAC	Time step factor for transients	Real, > 0.0	1.0
BACKUP	Flag to repeat time step (transient)	Real, 0.0 or 1.0	0.0 (no repeat)
OPEITR	Flag to print after each time step (transient)	Integer, 0 or 1	0 (no print)
NSOLOR	Nodal solution order for iterations (MATMET=0): 0 - input order, 1 - back and forth, 2 - mixed order	0, 1, or 2	2 (mixed order)
NVARB1	VARIABLES 1 call control from TRANSIENT ("FWDBCK"). If 1, VARIABLES 1 called within each iteration (several per time step).	0 or 1	0 (one call at start of time step)

Table 4-2 Control Constant Definitions

MATMET	Used to select iterative or simultaneous methods for all solution routines (except FORWRD)	0, 1, 2, 11, 12	12 (single matrix, AMG-CG)
AMGERR	Used to trade speed vs. accuracy for the AMG-CG (MATMET=11 or 12) solution	Positive real 0.0 to 1.0, exclusive	1.0e-10
METAMG	Used to trade speed vs. memory for the AMG-CG (MATMET=11 or 12) solution	0, 1, 2, 3	0 (maximum speed)
SPARSEG	Used to make large models more sparse (and so require more but faster iterations), perhaps enabling MATMET	Positive real 0.0 to 10.0	0.001
FBEBALA	TRANSIENT ("FWDBCK") energy balance criteria for iterative (MATMET=0) solutions	Positive real	0.1
*Multiple-Valued Constants: Input one data value per **fluid** submodel*			
DTSIZF DTTUBF	Transient time step size factor; fractional change allowed in key variables (transient). DTTUBF is the fractional change allowed in tube flow rates.	Positive real 0.0 to 1.0, exclusive	0.1
DTMAXF	Maximum time step (transient)	Positive real	1.0E30
DTMINF	Minimum average time step (transient)	Real	0.0
OUTPTF	Fluid submodel version of OUTPUT	Positive real	0.01*TIMEND
OPITRF	Fluid submodel version of OPEITR	Integer, 0 or 1	0 (noprint)
RSSIZF RSTUBF	Same as DTSIZF for STDSTL pseudo transient relaxations. RSTUBF parallels DTTUBF.	Positive real 0.0 to 1.0, exclusive	0.5
RSMAXF	Same as DTMAXF for STDSTL; maximum artificial time step	Positive real	1 hour
RERRF	Maximum relative change allowed per iteration in PL, TL, XL, and FR (steady state)	Positive real	0.01
REBALF	Fluid submodel version of EBALSA. Unitless fraction. Negative signals lump-level, otherwise submodel-level	Real	0.01
ITROTF	Fluid submodel version of ITEROT	Positive real	0
ITHLDF	Fluid submodel version of ITHOLD	Positive integer	0
FRAVER	Fluid submodel characteristic (average) flow rate	Positive real	0. (internal)
RMSPLT	Fraction of time step to apply to twinned tank anticipation	Positive integer	0.1
RMFRAC	Fraction of mass/energy error to apply per time step	Positive integer	0.5
RMRATE	Minimum time step over which energy/mass is corrected	Positive integer	0.1 second

a. Not intended to be changed within logic blocks: a preprocessor option only.

4.3.3 CONTROL Data Formats

When a HEADER CONTROL DATA record is encountered, the preprocessor expects a series of one or more "NAME = value," fragments in each data field (columns 2 through 80) on the following records until another header card is encountered. Such fragments must not be split between records (lines). Table 4-3 is an example of the CONTROL DATA blocks for a two submodel, transient solution model.

When multiple CONTROL DATA blocks are required, they need not appear together. The preprocessor will gather them together before interpreting them. *The GLOBAL block must, however, appear only once and precede all the other CONTROL DATA blocks.*

Table 4-3 CONTROL DATA Block Example

```
HEADER CONTROL DATA, GLOBAL
    NLOOPS    =     50
    DRLXCA    =      0.01,  ARLXCA = 0.02   $ Note: closing comma understood
    ABSZRO    =  - 459.67
    EXTLIM    =     10.0
    TIMEO     =      0.2
    TIMEND    =      5.0
    OUTPUT    =      0.2
HEADER CONTROL DATA, FRIC1
    EXTLIM    =     1.0,    DRLXCA = 0.005
HEADER CONTROL DATA, FRIC2
    OUTPUT    =      1.0
```

4.4 Control Constant Usage

The various SINDA control constants may be divided into two operational groups, as follows:

1) Program-calculated control constants (i.e., those calculated by the program for optional use by the user, including DRLXCC etc.)

2) User-specified control constants (i.e., those supplied by the user to control the program, such as NLOOPS and TIMEND).

Program-calculated control constants contain values output by the network solution routines as results of the solution routine calculations. They may be examined by users at their discretion, and may be used as criteria for controlling the operations in any of the user logic blocks. The following lists show the names of the program-calculated control constants.

Program-calculated single-value constants:

```
LINECT,  LOOPCT,   MLINE,   MMODS,   NMACT,
NNOD,    PAGECT,   TIMEM,   TIMEN,   TIMEO
```

Program-calculated multi-value constants:

```
smn.ARLXCC, smn.ATMPCC, smn.CSGMAX, smn.CSGMIN,
smn.DRLXCC, smn.DTIMEU, smn.DTMPCC, smn.EBALNC,
smn.EBALSC, smn.ESUMIS, smn.ESUMOS, smn.NARLXC,
smn.NARLXN, smn.NATMPC, smn.NATMPN, smn.NCGMAC,
smn.NCGMAN, smn.NCSGMC, smn.NCSGMN, smn.NDRLXC,
smn.NDRLXN, smn.NDTMPC, smn.NDTMPN, smn.NEBALC,
smn.NEBALN, smn.FBEBALC,fsmn.DTIMEF
```

where:

`smn` . . . . . thermal submodel name
`fsmn` . . . . . fluid submodel name

Many of the above variables are output only, meaning that the code calculates them for reference and they cannot be specified by the user. For example, DRLXCA is the allowed value, whereas DRLXCC is the calculated (output) value, as signaled by the last character (“A” vs. “C”). The same is true of other input/output pairs such as ATMPCA/ATMPCC and EBALSA/EBALSC.

User-specified control constants are used to influence calculations within the network solution routines. The following lists show the names of the user-specified control constants.

User-specified single-value constants:

```
ABSZRO, DTIMES, NLOOPS, SIGMA, TIMEN, TIMEND, TIMEO, PATMOS,
ACCELX, ACCELY, ACCELZ
```

User-specified multi-value constants:

```
smn.ARLXCA,  smn.ATMPCA,  smn.BACKUP,  smn.CSGFAC,
smn.DRLXCA,  smn.DTIMEH,  smn.DTIMEI,  smn.DTIMEL,
smn.DTMPCA,  smn.EBALNA,  smn.EBALSA,  smn.EXTLIM,
smn.ITEROT,  smn.ITERXT,  smn.ITHOLD,  smn.NLOOPT,
smn.OPEITR,  smn.OUTPUT,  smn.NSOLOR,  smn.NVARB1,
smn.MATMET,  smn.AMGERR,  smn.METAMG,  smn.SPARSEG
smn.FBEBALA, fsmn.DTSIZF, fsmn.DTMAXF, fsmn.RSSIZF,
fsmn.RSMAXF, fsmn.RERRF,  fsmn.REBALF, fsmn.ITROTF,
fsmn.OPITRF, fsmn.DTMINF, fsmn.ITHLDF, fsmn.DTTUBF,
fsmn.RSTUBF, fsmn.OUTPTF, fsmn.FRAVER, fsmn.RMSPLT,
fsmn.RMFRAC, fsmn.RMRATE
```

Internal control constants:

```
NDTMPC,NDTMPN. . . . . the model name and node number associated with  DTMPCC
NATMPC,NATMPN. . . . . the model name and node number associated with  ATMPCC
NDRLXC,NDRLXN. . . . . the model name and node number associated with  DRLXCC
NARLXC,NARLXN. . . . . the model name and node number associated with  ARLXCC
NCSGMC,NCSGMN. . . . . the model name and node number associated with  CSGMIN
NCGMAC,NCGMAN. . . . . the model name and node number associated with  CSGMAX
NEBALC,NEBALN. . . . . the model name and node number associated with  EBALNC
ESUMIS . . . . . Energy into a submodel
ESUMOS . . . . . Energy out of submodel
```

Figure 4-8 shows the general flow for the explicit integration routine FORWRD for each time step.* The arithmetic nodes are solved iteratively until |ARLXCC| < ARLXCA. On the other hand, Figure 4-9 shows that the implicit routine TRANSIENT solves the entire network iteratively each time step. In addition, TRANSIENT also checks for |DRLXCC| < DRLXCA and |FBEBALC| < FBEBALA before terminating the loop. Both classes of routines, however, perform checks on the maximum temperature change permitted during the entire time step and reduce the time step and repeat the calculations if the checks fail.

Steady state routines do not normally operate as a function of time. They perform iterations until the relaxation criteria ARLXCA and DRLXCA are satisfied (Figure 4-10), whereupon they begin to examine the best criterion for a steady state solution, the relative system energy balance. When the difference between the energy entering the network and the energy leaving it is less than or equal to EBALSA times the energy coming into the system, the iterations stop. If the iteration count had exceeded NLOOPS before the energy balance was achieved, the solution would have terminated at that point. Note that EBALSA is a fractional number, typically 0.01 or 0.02, and represents a relative (unitless) measure of imbalance.

* This discussion is limited to the thermal network solutions. For a discussion of the simultaneous fluid network solution, see Appendix E.

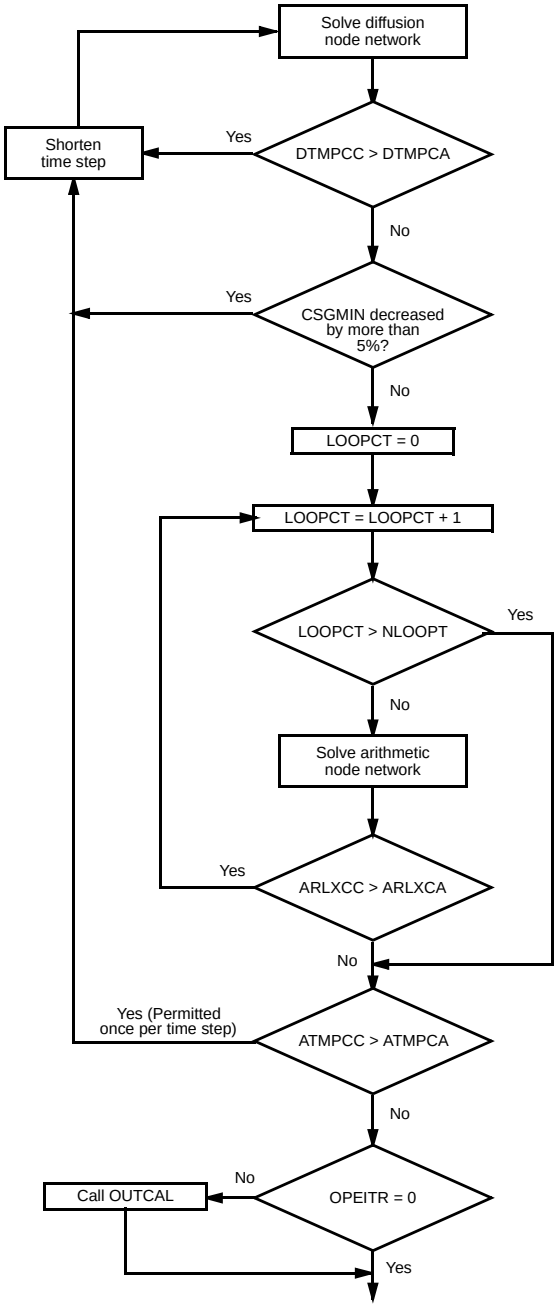


Figure 4-8 Basic Flow in FORWRD During Each Time Step

If “global” is used as a submodel name, this disables translation of the next keyword including the control constants listed above. There is no global OUTPUT or OUTPTF variable: OUTPUT CALLS, GLOBAL is controlled as a function of the OUTPUT and OUTPTF values of all active fluid and thermal submodels (Section 4.1.3.7).

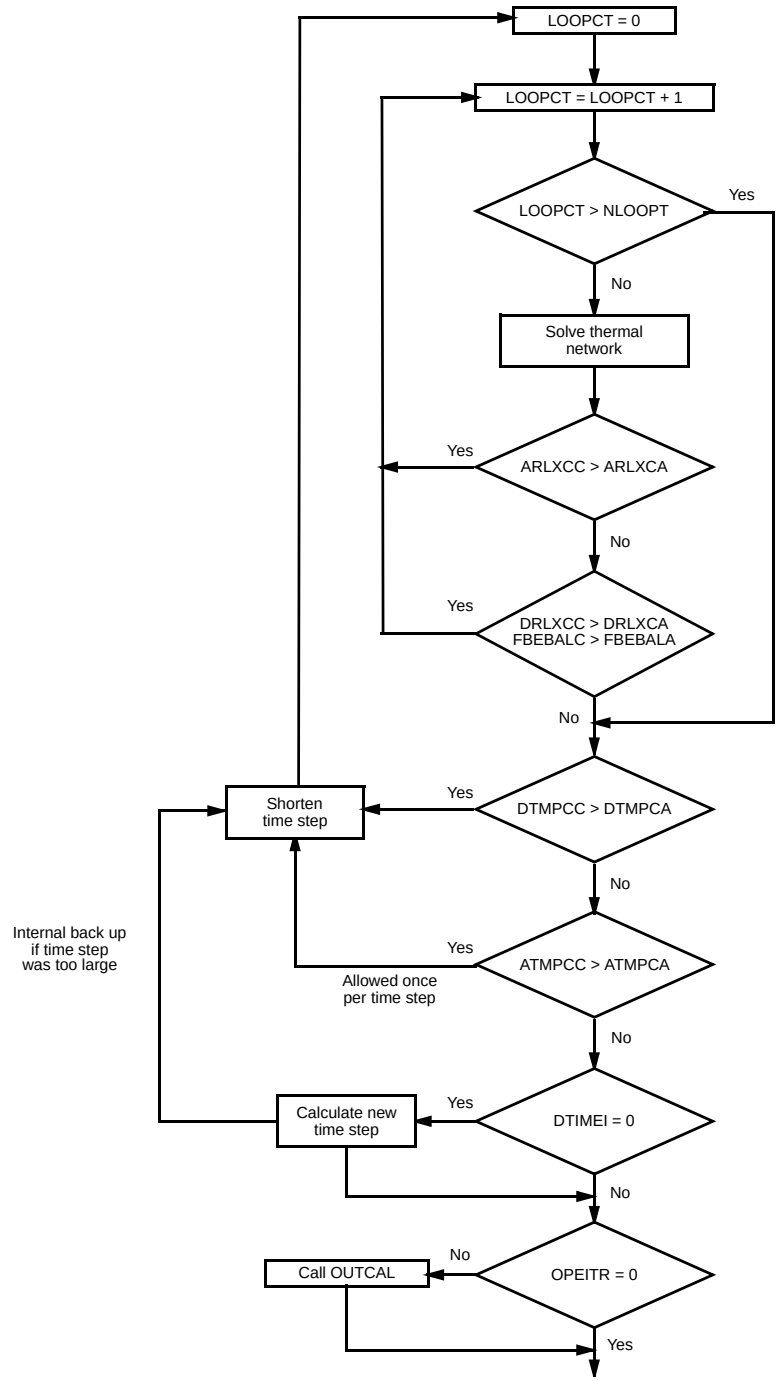


Figure 4-9 Basic Flow in TRANSIENT (“FWDBCK”) During Each Time Step

For transient solutions, the relaxation criteria are specified in terms of temperature change per iteration, and the temperature change criteria are specified in terms of temperature change per time step. If the former (DRLXCA and ARLXCA) are not satisfied, then further iterations, up to a maximum of NLOOPT, are performed. If the latter (DTMPCA and ATMPCA) are not satisfied, then

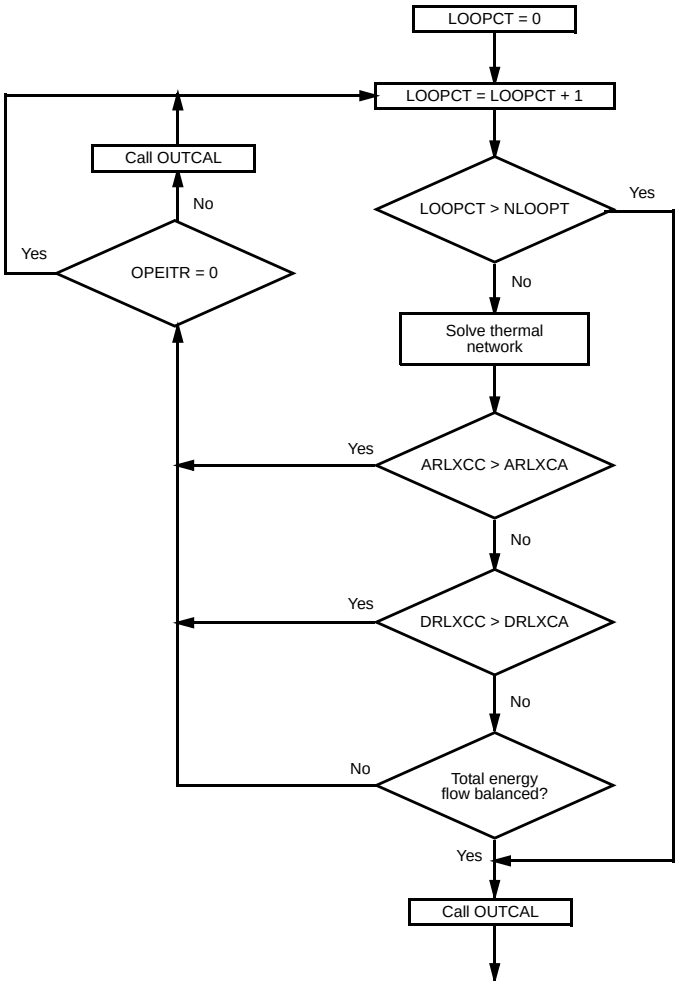


Figure 4-10 Basic Flow in STDSTL and STEADY (“FASTIC”)

the time step is shortened and the calculations are repeated. In addition, DRLXCA is used as the acceptable truncation error per time step for the automatic time step control feature (which is active by default when DTIMEI=0.0).

The points above will prove useful in understanding the following discussions of the individual, user-specified control constants. Table 4-2 summarizes the constants used by the various network solution routines and shows the default values assigned to each if they are not specified by the user.

Table 4-4 User-Specified Control Constants Required by Solution Routines

Name	Steady State STDSTL, STEADY	Transient 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

4.4.1 SINDA Control Constant Summary

ABSZRO (Value of absolute zero in user units)—This control constant is usually used to define temperature units and is usually set to 0.0, -459.67, or -273.15 although any number may be used as long as there are no fluid submodels active. ABSZRO is used internally in all execution routines as an offset factor for all temperatures (both nodes and lumps). The default value depends on the unit system identifier, UID, and is -459.67 if that flag is not used. *ABSZRO should not be changed during processor execution (e.g., in logic blocks).*

AMGERR (Allowable normalized error in the AMG-CG iteration)—This control applies to the MATMET=11 or MATMET=12 (AMG-CG) method, and provides a trade-off between speed and accuracy. The default value of 1.0E-10 (10^{-10}) results in nearly the same answers as YSMP (MATMET=1 or 2): it forces the AMG-CG internal iteration to continue until the answers are virtually the same as those produced by a direct YSMP solution. Larger values of AMGERR (up to 10^{-6}) result in even faster execution, though these increases are usually modest.

Large values of AMGERR are not particularly dangerous, since SINDA ultimately controls the accuracy of the solution. However, values much larger than about 10^{-4} to 10^{-5} might cause SINDA to invoke extra iterations to achieve the required accuracy (e.g., LOOPCT=2 instead of LOOPCT=1 for a transient time step), which would likely defeat any benefit gained by faster matrix inversions.

ARLXCA (Allowable Arithmetic Node Relaxation Temperature Change)—This control constant can be specified for any solution routine in which an arithmetic node is present. If ARLXCA is not specified, the default is 0.01 degrees (and so is unit dependent). ARLXCA represents a temperature convergence criterion for the arithmetic nodes during each iterative calculation. Iterations proceed until either all convergence criteria including ARLXCA have been satisfied, or until the maximum number of iterations have been attempted (NLOOPT or NLOOPS). ARLXCA has units of degrees. Although the default is 0.01, its recommended value is dependent upon the magnitude of expected temperatures. The 0.01 value signifies fifth decimal place "accuracy"* for absolute temperatures in the hundreds, but only third to fourth place accuracy for cryogenic temperatures. The corresponding output variable, against which ARLXCA is compared, is ARLXCC. (See also DRLXCA.)

ATMPCA (Allowable Arithmetic Node Temperature Change)—This control constant may be optionally specified by the user for the implicit or the explicit transient integration routines. The default is 1.0E30. ATMPCA represents the maximum allowable arithmetic node temperature change during one time step. It is compared to the calculated temperature change which is stored in control constant ATMPCC. If ATMPCC is greater than ATMPCA, the time step, DTIMEU, is shortened to:

$$DTIMEU = X * DTIMEU*(ATMPCA/ATMPCC)$$

where X = 0.95 for FORWRD and 0.5 for TRANSIENT.

* ARLXCA is a convergence criterion, not an accuracy criterion. It measures how quickly a model is changing during the solution, and not how accurate the answers are.

and the computational procedure is then repeated with the smaller time step. Specification of ATMPCA prevents a rapid temperature change between time steps, with the value to be specified dependent upon the problem. ATMPCA should be used to prevent missing important temperature-dependent variations, and so should be chosen by the user according to the rates of change of properties with temperature. *However, it is not always possible to obey the limit imposed by ATMPCA* since arithmetic node temperature changes can be independent of time. In order to avoid an infinite loop, *the program will not reset the time step twice in a row because of this criterion.* Thus, the user may occasionally notice a value of ATMPCC that is higher than ATMPCA.

For these reasons, DTMPCA should be used in preference to ATMPCA, which has been preserved for backwards compatibility with older models.

BACKUP (Backup Switch)—Control constant BACKUP provides the user with the means for utilizing any numerical solution subroutine as a predictor program. All of the numerical solution subroutines set control constant BACKUP to zero just prior to the call on VARIABLES 2. Then, immediately after the return from VARIABLES 2, a check on BACKUP is made. If BACKUP is nonzero, all temperature calculations for the just completed time step are eliminated; the old temperatures (temperatures calculated at the previous time step) are placed in the temperature locations; and control is routed to the start of the computational sequence.

It should be noted that the user must provide the necessary criterion and check in VARIABLES 2 if the iteration is to be repeated. Thus, if the iteration is to be repeated, BACKUP must be nonzero and a criterion that can be met in the next pass must be established to avoid an infinite loop.

Finally, note that only thermal submodels are affected by BACKUP. Fluid submodels cannot be reversed once solved due to memory considerations.

CSGFAC (Time Step Factor)—This control constant may be optionally specified by the user for the transient solution routines. It provides the user with some degree of control over the computer time step. It defaults to one.

For subroutine FORWRD, which is conditionally stable, CSGFAC is a time step divisor (DTIMEU=CSGMIN/CSGFAC); a value of CSGFAC *greater than one* is input to obtain greater accuracy. (Values less than one will be ignored.)

For subroutine TRANSIENT, which is unconditionally stable, CSGFAC is ignored unless DTIMEI is specified as a negative value, in which case the time step becomes CSGMIN/CSGFAC and automatic time step calculation is otherwise overridden.

DRLXCA (Allowable Diffusion Node Relaxation Temperature Change)—This control constant can be specified for the implicit routine TRANSIENT and for the steady state routines STDSTL and STEADY. The default value is 0.01 degrees (and so is unit dependent). The corresponding output variable, against which DRLXCA is compared, is DRLXCC.

For steady state solutions, DRLXCA serves the same purpose for the diffusion nodes as the control constant ARLXCA serves for the arithmetic nodes. Thus, the discussion on ARLXCA holds equally true for DRLXCA. Separate relaxation criteria for diffusion and arithmetic nodes are available because they provide greater computational flexibility and facilitate the coupling of two distinct networks.

DRLXCA serves a dual purpose in TRANSIENT when the automatic time step predictor is enabled (by letting DTIMEI default to zero). In that case, in addition to helping determine when the implicit solution at the next time step has been found by iterative relaxation, DRLXCA is used as a measure of the acceptable truncation error per time step. Truncation error is the numerical error introduced by the assumption that higher order terms can be neglected over small intervals.* The interval or time step size is adjusted such that this error term is approximately equal to the minimum of DRLXCA in all active submodels.

While most users use the same value of DRLXCA in both TRANSIENT and the steady state routines (STDSTL and STEADY), this is not always a good practice. The reason is that while the error in temperatures for steady state routines in on the order of DRLXCA and ARLXCA, this same error applies to each time step taken in TRANSIENT. *Thus, the total error in TRANSIENT is cumulative using DRLXCA as a criterion.* While the automatic time step predictor (used when DTIMEI=0.0) coupled with the DTMPCA option usually eliminates such problems, the user should consider using a smaller value of DRLXCA in TRANSIENT than in STDSTL or STEADY.

DTIMEH (Maximum Time Step Allowed)—This control constant may be optionally specified for the explicit and the implicit transient routines (FORWRD and TRANSIENT, respectively). DTIMEH represents the maximum time step allowed during the integration process. If DTIMEH is not specified, it is set to 1.0E+30. When DTIMEI is not input with TRANSIENT, then the smaller of DTIMEH, OUTPUT, DTMAXF, and OUTPTF will be the upper limit on the time step used. One of these control constants should be used to make certain that the time step is not allowed to be so large that fast changes in boundary conditions are missed. *Strictly, solutions using FORWRD should only be accepted if the same answers result from a second run made using a smaller time step. DTIMEH represents an easy means of imposing this lower time step.* (See also DTMPCA.) In TRANSIENT, if cautions are produced regarding integration accuracy, the user should apply a lower value of DTIMEH, or at least one that is lower than CSGMIN, the smallest time characteristic.

DTIMEI (Input Time Step for TRANSIENT)—This control constant is optional for TRANSIENT (“FWDBCK”) and is not used by FORWRD. If DTIMEI is set to 0.0 (the default), TRANSIENT will use the routine FBECHK to choose a time step. *This is the suggested mode of operation; nonzero DTIMEI should be considered only after having tried zero DTIMEI with an appropriate value of DTIMEH. Solutions using nonzero DTIMEI should only be accepted if the same answers result from a second run made using a smaller time step.*

During execution, FBECHK will vary the time step to maintain the same level of accuracy, measure by the value of DRLXCA. It will take large time steps when temperature changes are slow and short time steps when any temperature change is rapid. It may back-up and repeat a time step if the error is too large, or in case of nonconvergence (LOOPCT>NLOOPT) if diffusion nodes were

* In TRANSIENT, this error is proportional to the cube of the time step times the third partial derivative of temperature with respect to time.

at fault. In almost all cases, this will result in shorter execution time and more accurate results than if positive DTIMEI is specified. The concurrent use of DTMPCA is recommended. Also, note that DTIMEI is ignored if fluid models are active and if they require a smaller time step.

Positive values of DTIMEI represent an arbitrary time step, but the governing criteria should be minimum computational time with satisfactory temperature accuracy. Remember that *implicit* means unconditionally *stable* for any time step, not unconditionally *accurate.* In fact, implicitness guarantees nothing about the accuracy of the results. It only guarantees that they will not diverge if the "real" solution does not diverge. Thus, using an arbitrary time step will result in arbitrary accuracy; nonsensical values may result if DTIMEI is too large.

Negative values of DTIMEI also signal that the automatic time step prediction is to be overridden, but instead of a constant, user-specified time step, a time step proportional to CSGMIN (smallest characteristic nodal time constant) is taken as dictated by the CSGFAC constant: DTIMEU = CSGMIN/CSGFAC. Thus, values of CSGFAC larger than one cause a smaller time step, and values less than one cause a larger time step.

DTIMEL (Minimum Time Step Allowed)—This control constant is optional for the transient solution routines. DTIMEL represents the minimum time step allowed. If the calculated time step is less than DTIMEL, the run terminates with an error message printout. Note DTIMEL is ignored in TRANSIENT if the user inputs the time step using positive DTIMEI. If fluid submodels are active, the fluid control constant DTMINF should be used instead of DTIMEL because that control constant is more tolerant of occasional small time steps.

DTIMES (Steady-state Pseudo Time Step)—This control constant is optional for the steady-state solution routines. DTIMES represents an artificial time step which will be taken until TIMEO is incremented to TIMEND. A complete steady-state analysis will be run at each time interval, representing a "transient" analysis based on steady-state assumptions—the system is assumed to react infinitely fast to changes in boundary conditions. DTIMES defaults to zero, meaning that only one steady-state analysis is required at the current time value (TIMEO). See also Section 4.7.2.

DTMPCA (Allowable Diffusion Node Temperature Change)—This control constant may be optionally specified by the user for the transient solution routines. The default is 1.0E30. DTMPCA represents a maximum allowable diffusion node temperature change between one time step and another. If the maximum diffusion node temperature change, which is stored in DTMPCC, is greater than DTMPCA, the time step is shortened to:

$$DTIMEU = X * DTIMEU*(DTMPCA/DTMPCC)$$

where X = 0.95 for FORWRD and 0.5 for TRANSIENT.

and the temperatures are reset to their former values. The computational procedure is then repeated with the smaller time step. DTMPCA serves the same purpose for the diffusion nodes as control constant ATMPCA provides for the arithmetic nodes. Unlike ATMPCA, DTMPCA is always obeyed and should be the primary tool for making sure that rapid temperature-dependent variations are not missed. A value of 10°F (5.5°C) has been found to be appropriate in most spacecraft thermal analyses.

EBALSA (System Energy Balance Convergence Criterion)—This control constant is used in STDSTL and STEADY but not in the transient solution routines. The corresponding output variable, per thermal submodel, against which EBALSA is compared is EBALSC. The default value of EBALSA is 0.01, and values smaller than 10^{-6} should be avoided. Note that the value of EBALSA is expressed as a fraction. This fraction represents an acceptable *relative* energy balance of the system, and is therefore unitless: it is independent of the numerical value of the energy flow through the system. Iterations will continue until the total system energy balance is within the fraction EBALSA of the energy into the system, or until NLOOPS has been reached. Mathematically, the energy balance is achieved when:

ABS(TSUMIS-TSUMOS) .LT. EBALSA * ABS(MAX(TSUMIS,TSUMOS))

where:

TSUMIS. . . . . Energy into total system (sum of ESUMIS for all submodels)
TSUMOS. . . . . Energy out of total system (sum of ESUMOS for all submodels)

This energy flow check occurs at the master model level only; *checks at the submodel level are neglected due to intermodel conductors.* (The values of ESUMIS and ESUMOS for each submodel are therefore rarely equal.) Furthermore, the system may be considered converged if the total energy flows (TSUMIS and TSUMOS) are unbalanced but are not changing and are not growing towards each other. A caution is produced in such cases,* which often arise from one-way conductors or path/tie duplication factors, in which case the message can be ignored. However, this message is also encountered when huge conductors are used, and the program has detected negligible progress toward a solution. In the latter case, the user should either reduce the convergence criteria (perhaps using EBALNA instead) and continue, use matrix inversion methods (MATMET>1), or consider changes or corrections to the model such as combining nodes to eliminate the huge conductors, or starting with better initial conditions. See also “ENERGY STABLE BUT NOT BALANCED” page 61 of this section.

EBALNA (Nodal Energy Balance Convergence Criterion)—This control constant is used in STDSTL and STEADY but not in the transient solution routines, and is analogous to EBALSA but occurs at the nodal level rather than the master model level. The corresponding output variable, per thermal submodel, against which EBALNA is compared is EBALNC. Iterations will continue until the largest value of |EBALNC| for all nodes is less than EBALNA of the energy into that node, or until NLOOPS has been reached. *The default value is 0.0, meaning that nodal level checks are skipped.*

Unlike EBALSA, EBALNA is *not* expressed as a fraction, and therefore has units of power. This control constant represents an acceptable *absolute* energy balance of the node. In other words, the value of EBALNA is dependent on the numerical value of the energy flow through the node.

EXTLIM and ITERXT (Convergence Acceleration Schemes)—These two control constants control the behavior of the temperature extrapolation/dampening routine or control the application of over- and under-relaxation factors.

* “ENERGY STABLE BUT NOT BALANCED.” See also page 61 in this section.

By default (ITERXT=3), all the solution routines perform temperature extrapolations to speed convergence (this extrapolation is limited to arithmetic nodes in FORWRD). This approach is normally faster, safer and less model-sensitive than the "damped" acceleration/deceleration method used in older versions of SINDA. For nodes that are monotonically approaching a limit, the extrapolator accelerates them towards that limit. For nodes that are oscillating in temperature, the extrapolator provides damping.

EXTLIM is the maximum temperature change allowed as a result of an extrapolation. The default value of EXTLIM is 5.0. *Note that this is unit-specific and should be tailored for each model. If a model is oscillating, convergence can sometimes be achieved by reducing this value.*

ITERXT is the number of iterations which must be performed before an extrapolation is performed. The default value of ITERXT is 3. (This is also the minimum value of ITERXT if the extrapolator/damper is to be used: a value less than 3 denoted other methods as described below.) If a model is oscillating, convergence might be facilitated by increasing this value. Entering a value greater than NLOOPS or NLOOPT will eliminate extrapolation completely.

Alternatively, values of ITERXT below 3 (0, 1, or 2) signal the use of over/under-relaxation methods instead of the extrapolator/damper. An over- or underrelaxation parameter ω accelerates or slows changes to each node's temperature during the successive update:

$$T_{new} = T_{old} + \omega*(T_{new} - T_{old})$$

The parameter ω can be set directly by the user by setting ITERXT=0, which *changes the meaning of EXTLIM* to imply the user has provided a value of ω in that parameter. A value of ω greater than 1.0 (overrelaxation) accelerates the temperature changes in the models at the risk of nonconvergence in the event predictions oscillate, especially in cases with extensive user logic manipulations. A value of ω less than 1.0 is useful to forcing convergence when the predictions are oscillating or are unstable, but may slow convergence or cause STDSTL to stop prematurely due to lack of progress.*

Normally, the default extrapolator/damper is more efficient and robust than a user-provided over- or under-relaxation factor. However, a useful alternative is to let the code calculate the value of ω based on the following formula:

$$\omega = 2 / (1 + (1-\rho^2)^{1/2})$$

where ρ is the ratio of the current maximum temperature change, $|T_{new}-T_{old}|$, over the value from the previous iteration. This method, which always over-relaxes, is invoked by setting ITERXT=2.† It is especially useful in conduction-dominated problems with smooth user logic and properties that do not vary greatly with temperature. A slightly more aggressive method is applied when ITERXT=1,‡ in which case the ρ term in the above equation is not squared, increasing the value of ω. Note that EXTLIM is ignored when ITERXT=1 or ITERXT=2.

* If oscillations between two sets of answers is detected for most nodes in the solution, the program will issue a warning and automatically set ITERXT=0 and ω=0.5 (EXTLIM=0.5).

† This ITERXT=2 option is referred to as "Classical SOR" in the Thermal Desktop Case Set Manager.

‡ This ITERXT=1 option is referred to as "Aggressive SOR" in the Thermal Desktop Case Set Manager.

FBEBALA (TRANSIENT Energy Balance Criterion)—TRANSIENT ("FWDBCK") solves the temperature at the next time point, T^{n+1}, implicitly: several iterations are required each time step to estimate the next set of temperatures. If MATMET=0 (purely iterative solution) and if large conductances are present, then TRANSIENT might converge prematurely on the temperature values at the next time point without a check on energy imbalance to supplement the checks on temperature change (DRLXCA, ARLXCA).

FBEBALA defaults to 0.1, which means that a maximum nodal power imbalance of approximately* 10% is tolerated each time step. The output variable FBEBALC can be used to assess the current magnitude of this error term. In other words, the code continues to perform closure iterations on each time step until |DRLXCC| < DRLXCA, |ARLXCC| < ARLXCA, and |FBEBALC| < FBEBALA.

Setting FBEBALA lower than 0.1 reduces the power imbalance allowed: FBEBALA=0.01 means 10 times lower imbalance tolerated per time step than the default: approximately 1% imbalance. Such tight tolerances can cause significant increases in solution cost, along with nonconvergence warnings and associated decreased time steps (for the default DTIMEI=0, meaning automatically calculated time steps).

As a side effect of nonzero FBEBALA, EBALNC and NEBALC are updated during TRANSIENT if MATMET=0. However, note that EBALNC will contain an *approximate* power imbalance in this case.

FBEBALA was introduced in Version 4.8. To return to the methods used in V4.7 and earlier (i.e., no energy balance check for TRANSIENT closure iterations), set FBEBALA=0.0. Setting FBEBALA=0.0 turns off this check (a huge value has a similar effect), but if those checks are suspended then consider a call to COMBAL (Section 7.6.12) to investigate the magnitude of the error via imbalance data printed to the output file. Using simultaneous solutions (nonzero MATMET), decreasing DRLXCA, or using FORWRD instead all reduce the error, though usually also by increasing solution costs. Decreasing excessive nodal resolution and/or eliminating overly large G values (which imply negligible temperature gradients between nodes) are by far the best solutions.

ITEROT (Iteration Output interval)—This control constant serves as a switch that controls the calling of subroutine OUTCAL for thermal submodels in steady-states. If set nonzero, OUTCAL will be called every ITEROT iteration steps. Setting ITEROT to 1 before NLOOPS is reached (say at NLOOPS-LOOPCT=10) is a common method for investigating nonconvergence in troublesome models. For example, in CONTROL DATA:

```
ITEROT = (LOOPCT >= NLOOPS-10)? 1 : 0
```

* This meaning cannot be taken literately due to the use of various approximations necessary to minimize computational cost.

ITHOLD (Number of Iterations to Hold semi-converged submodels)—This control constant can be used purely for computational efficiency, and its effects can be eliminated by setting it to be zero (the default). It governs the maximum number of solution steps that more completely converged thermal submodel can be suspended from a STDSTL and STEADY solution. The goal is to have all submodels reach final convergence at the same time without wasting computational effort. Those thermal submodels achieving the greatest degree of convergence (by satisfying the most convergence criteria—temperature change or energy balance) are temporarily dropped from the computation cycle for a maximum of ITHOLD (default = 0) iterations and then reactivated. In such a dormant state, calls to a submodel's logic blocks (namely, VARIABLES 1 and OUTPUT CALLS) continue even though the nodes in the submodel are not being updated.

Final convergence occurs when *all* submodels (thermal and fluid) satisfy *all relevant* criteria. ITHOLD works in concert with ITHLDF, the equivalent control constant for fluid submodels.

While it usually results in reduced solution costs (which unlike old SINDA is no longer directly measured by the final value of LOOPCT, as explained below), users with convergence problems should let it default to zero, especially in the presence of extensive user logic that updates values in one submodel as a function of parameters in other submodels.

Note that the use of ITHOLD can easily increase the total number of iterations, LOOPCT, required to achieve convergence over that required when ITHOLD=0. *This should not be mistaken as an increased cost.* Rather, measurement of the total CPU time would reveal that while LOOPCT increased, the total CPU time might *decrease* using ITHOLD since not all submodels were active for all iterations.

MATMET (Matrix Methods Signal)—In prior versions, the default was a purely iterative solution (MATMET=0). The current default is MATMET=12: all nodes' equations collected into a single matrix and solved simultaneously using AMG-CG (Appendix F.2).

The values of MATMET have the following meaning:

0	Thermal submodel to be solved iteratively
1	Thermal submodel to be solved simultaneously (per time step or per steady state solution pass) using the YSMP method
2	Thermal submodel to be solved simultaneously with all other thermal submodels that also have MATMET=2, using the YSMP method
11	Thermal submodel to be solved simultaneously (per time step or per steady state solution pass) using the AMG-CG method
12	Thermal submodel to be solved simultaneously with all other thermal submodels that also have MATMET=12, using the AMG-CG method.

When MATMET=0, during each iterative pass of STDSTL, STEADY ("FASTIC"), and TRANSIENT ("FWDBCK"), nodes are solved one at a time treating other nodes temporarily as boundary conditions. In problems with large conductors, with long conductive paths, and with poor initial conditions, such iterative methods can be slow or can stall (see "ENERGY STABLE BUT NOT BALANCED" page 61 of this section) despite the accelerations attempted via EXTLIM and ITERXT. As an alternative, all the nodes in a submodel can be solved simultaneously using a sparse matrix solver if MATMET=1 or 11. Collections of submodels are solved together (in a single matrix)

if MATMET=2 or 12. Although convergence of conductance-dominated problems is normally tremendously accelerated by this approach, the cost per iteration is higher than with iterative methods and several iterations (each using a simultaneous solution in STEADY and STDSTL) are still required to adjust for radiation, temperature-dependent conductivities and sources, and other user logic manipulations.

MATMET has no meaning in FORWRD. Otherwise, when MATMET>0 simultaneous solutions are applied to every pass (LOOPCT=1,2, ...) in steady state or transient (TRANSIENT or "FWDBCK") solutions. When MATMET>0, normally only two or three iterations are required in TRANSIENT (unless NVARB1=1). In STEADY or STDSTL, normally about ten iterations are required for thermal-only problems.

Even if all submodels elect MATMET=1 or 11, the entire set of nodes is not solved simultaneously, just each submodel. This design exploits the facts that (1) most conductors *between* submodels are weaker than the ones *within* submodels, and that (2) matrix methods work best for smaller problems. Therefore, MATMET=1 or 11 should not be used (at least not perpetually--see next paragraph) if strong connections exist between submodels. *This restriction includes strong ties to fluid submodels in steady state solutions* (but not in transient solutions). In fact, the advantages of breaking up a single matrix into sets of smaller matrices essentially vanishes for AMG-CG methods (MATMET = 11 or 12).

If strong interconnections exist between thermal submodels, elect instead MATMET=2 or 12. All submodels employing MATMET=2 or 12 will be solved together (in a single matrix) as a group. Use of MATMET=2 and MATMET=12 cannot be mixed: a single collected matrix is used.

In rare cases, especially for small models, MATMET=2 may actually be faster than MATMET=12. Otherwise, MATMET=2 and MATMET=11 are mostly available for completeness. Therefore, the following suggestions apply:

Use the default MATMET=12 unless problems arise.
Consider small or zero SPARSEG if some portions of the network are suspected of being radiatively isolated (poorly connected). Increasing damping (ITERXT=0, EXTLIM < 1.0) may also be required if the simultaneous solution oscillates between two sets of answers (and therefore doesn't converge, or does so slowly).

Use MATMET=0 to investigate network construction problems or logic.
Simultaneous solutions tend to obscure errors by propagating their effects through-out the network instantly, whereas iterative methods make it easier to isolate problems, perhaps using ITROUT=1 with limited NLOOPS to watch changes iteration by iteration.

One trick is to start a steady state run with MATMET>0 to jump near the final solution, then to reset MATMET=0 to complete the convergence. An alternative scheme is to turn on the matrix solution every N (say, 10 or 100) iterations. In both cases, note that while matrix solutions can be turned off at any time, they cannot be turned on unless MATMET>0 was elected in CONTROL DATA such that the preprocessor allocated the necessary memory.* Therefore, set MATMET=1 or 2 in either global or submodel control data if it ever to be elected during the course of the solution.

When MATMET=1 or 2 (YSMP method) is set in the preprocessor, workspace is allocated for performing the matrix solution based on estimations: the exact amount of storage required cannot be predicted in advance. The amount of storage can be quite large for huge models (with thousands or tens of thousands of nodes). If insufficient memory has been allocated, the matrix solution will fail and the program will reset MATMET=0 and will return to an iterative solution. The user can elect to override the internal calculations of workspace by setting the NWORK parameter in HEADER NODE DATA (Section 2.12), as prompted by the program. Alternatively, the user can call YSMPWS (no arguments) at the end of OPERATIONS to report actual memory utilization (Appendix F), perhaps then electing to lower the value of NWORK to reduce memory requirements in future runs.

For MATMET=2, the workspace allocation is the sum of all the NWORK values of participating submodels, whether defaulted or specifically input in NODE DATA.

Memory allocation is not required for MATMET=11 or 12 (AMG-CG method).

If fatal problems arise during the MATMET=11 or 12 solution, SINDA/FLUINT will revert to MATMET=1 or 2, respectively.

While radiation terms are linearized and handled rather well (albeit iteratively) by the matrix methods, too many weak radiation terms tend to fill the matrix excessively (i.e., reducing its sparseness), significantly slowing the solution and increasing memory requirements while not contributing significantly to the predictions. Therefore, it is strongly recommended that the user eliminate insignificant radiation terms* before using matrix methods. If the resulting matrix is too dense and MATMET=1 or 2 has been used, a caution will be issued and the program will automatically revert to MATMET=0: iterative solutions.

For the default AMG-CG method (MATMET=12), trade-offs between memory and speed, and between accuracy and speed, can be controlled via the METAMG and AMGERR controls, respectively.

---

* MATMET can be changed from 0 (in CONTROL DATA) to 1 or 11 (in logic blocks or expressions) without regard to the above restrictions. However, MATMET=2 or 12 cannot be elected unless initially specified as such in CONTROL DATA. Since MATMET=12 is the default, this concern only applies if the user has overwritten the default.

* See, for example, the Bij cutoff factor in Thermal Desktop® and RadCAD®: GRAPHICAL (GEOMETRIC) CONCURRENT ENGINEERING on Page xliii.

METAMG (memory vs. speed trade-off control for AMG-CG method, integer)—The default for this coarsening aggressiveness is 0, meaning "maximum" memory allocated in exchange for the greatest solution speed when MATMET=11 or 12. In really large models with radiation, available memory may be more of a limiting factor than is CPU speed, at least on some machines. Paging virtual memory can significantly raise execution times. In this case, the user can choose to use less memory in exchange for reduced solution speed by raising METAMG from the default zero to 1, 2, or 3 (the maximum). In some models, choosing METAMG=3 instead of the default METAMG=0 reduces memory requirements *for the AMG-CG solution* by about half, at a cost of reduced execution speed (also about half). Since AMG-CG memory requirements are normally modest compared to the rest of SINDA/FLUINT, and since other housekeeping processes can consume considerable memory, increasing METAMG should only be attempted after exhausting other measures,* including shutting down unnecessary applications before executing SINDA/FLUINT.

NLOOPS and NLOOPT (Number of Iteration Loops)—NLOOPS is the maximum number of iterations for both thermal and fluid submodels in STDSTL and STEADY. NLOOPT is the maximum number of iterations per time step for the transient routines TRANSIENT and FORWRD. The output constant LOOPCT contains the current (or final) iteration count in all routines.

The default for NLOOPT is 100. The default for NLOOPS is 1000. For a large steady state problem, it is not unusual to have NLOOPS equal to several hundred to a thousand if MATMET=0 (iterative methods). If the default matrix methods (see MATMET) and/or only fluid submodels are employed, then the number of steps can be on the order of 100 to 500.

For the implicit routine TRANSIENT, NLOOPT should be no larger than 100 if DTIMEI is defaulted to zero. In fact, in TRANSIENT a value of LOOPCT beyond about 10 is an indication of an attempt to take a time step that is too big. If DTIMEI=0.0 and the program is predicting the time step, then failing to converge in NLOOPT iterations requires the program to back up and reattempt the solution with a smaller time step.

NLOOPT may be specified for the explicit routine FORWRD since it is used for the arithmetic nodes. A value of 100, in combination with an ARLXCA value of 0.01, is usually adequate. In general, a trial and error procedure is required to arrive at suitable values for ARLXCA, DRLXCA and NLOOPT.

NSOLOR (Solution Order Flag)—This flag controls the order in which nodes are solved during an iteration. In STDSTL and STEADY, if MATMET=0 then nodal temperatures are solved with iterative passes. During each iteration, the temperature of each node is changed such that energy flows through that node balance. One pass through all nodes is considered an iteration. TRANSIENT uses a similar scheme to update arithmetic node temperatures within each time step.

By default, the solution sequence is varied (NSOLOR=2), which is not only computationally efficient, it makes the results less sensitive to changes in input order. In the default sequence, the input order is used initially, followed by the reverse direction, followed by a forward pass through every other node and then a backward pass through the nodes missed on the first pass. (Actually, the sequence is somewhat more complicated to describe, since the same solution order is used three

* In addition to model reduction, consider increased SPARSEG values and increased RadCAD B_{ij} cut-off values.

consecutive times prior to an extrapolation as governed by the ITERXT constant). To return to the methods used in versions prior to Version 3.2, use NSOLOR = 0, which solves nodes in input order. To try a two-step method (alternative forward-backward), choose NSOLOR = 1.

NSOLOR need not be the same value in all thermal submodels. It may be changed at any time within logic. Frequent changes, however, are counterproductive since they can conflict with the extrapolator/accelerator logic.

NSOLOR is used in TRANSIENT but not FORWRD, which only iterates arithmetic nodes.

NVARB1 (VARIABLES 1 Call Control in TRANSIENT)—VARIABLES 1 is called once per steady state iteration (in STEADY and STDSTL), providing an opportunity for inserting temperature-dependent changes, modeling control systems, or for performing other frequent updates. In TRANSIENT, however, in which each iteration is somewhat analogous to a miniature steady-state solution, VARIABLES 1 is only called once at the beginning of the time step by default (NVARB1=0). This represents an assumption that temperature-dependent changes are small over the course of the time step (e.g., that conductances and capacitances can be calculated once and then held constant over the time step, instead of updated during the course of the iterative solution of the implicit equations). Perhaps more importantly, it means that logic (especially control system simulations) written for steady-state routines may not function analogously when submitted to a transient analysis.

As an alternative, the user may elect to have the VARIABLES 1 block called each iteration in TRANSIENT by setting NVARB1 to 1 for any or all submodels. This gives slightly higher fidelity answers for temperature variations, and more complete implicitness. However, it can also lead to nonconvergence problems depending on user logic.

If NVARB1 is set for some but not all submodels, there is a slight potential for problems if nodal heat rates (Qs) from a model that is using NVARB1=1 are updated in VARIABLES 1 (perhaps using a temperature-variable source or a call to HEATER) of another submodel that uses NVARB1=0, or vice versa. *Either use NVARB1 globally, or be avoid setting the Qs for one submodel in the VARIABLES 1 of another submodel. See also VARIABLES 1, GLOBAL: the global VARIABLES 1 block.*

The user may turn off NVARB1 (i.e., set to zero) within VARIABLES 1 itself, terminating further calls within that time step. It may be turned back on later for subsequent time steps.

CAUTION—Use of HEATER (Section 7) in VARIABLES 1 with NVARB1=1 can be problematic unless the time steps are comparatively small compared to the characteristic heating time of the control node (governed by its capacitance, the control deadband, and the heat rate applied). High heat rates, small deadbands, and low capacitances must be avoided or small time steps will result (and even then nonconvergence will persist). Note that if NVARB1 were equal to zero, the resulting integration would have been in error anyhow since it would not be capturing switching cycles. However, the automatic time step controller is not allowed to take time steps smaller than CSGMIN, and that parameter does not account for Q rates or for changes in those rates, and therefore may not represent a true minimum for accuracy requirements. Other control systems may experience similar convergence problems, as they would in a steady-state analysis.

OPEITR (Output Each Iteration)—This control constant serves as a switch that controls the calling of subroutine OUTCAL for thermal submodels in transients. If set nonzero, OUTCAL will be called at each time step (the term “iteration” is a misnomer), as shown in Figures 4-8, 4-9, and 4-10. OPEITR is mostly used as a model debugging tool.

OUTPUT (Time Interval for Activating OUTPUT CALLS)—This control constant defaults to 0.01*TIMEND if thermal submodels are present, meaning approximately 101 calls to the OUTPUT CALLS block will be made during a transient. Normally, the output interval is gauged by the length of the run and the expected temperature response characteristics. As a “rule-of-thumb” the output interval lies between CSGMIN and CSGMAX, with OUTPUT being several times larger than CSGMIN. The values of CSGMIN and CSGMAX can be obtained from the output subroutines. When using TRANSIENT, the user should remember that OUTPUT and DTIMEH set the upper limit on the time step used (see discussion on control constant DTIMEH), unless fluid submodels are active, in which case even smaller intervals may result.

There is no global OUTPUT variable: OUTPUT CALLS, GLOBAL is controlled as a function of the OUTPUT and OUTPTF values of all active thermal and fluid submodels (Section 4.1.3.7). If “global” is used as a submodel name, translation of the next keyword is disabled.

SIGMA (Stefan-Boltzmann Constant)—This control constant provides a convenient means of input for the Stefan-Boltzmann constant, however any number may be used. SIGMA is used internally in all execution routines as a multiplication factor for all radiation conductors. It is never stored in the conductance (G) array. When SIGMA is not specifically input, it has the default value of 1.0. *SIGMA should not be changed during processor execution (e.g., in logic blocks)*. The user will find the built-in expression constants *sbcon* (0.1712E-08) and *sbconsi* (5.6693E-08) to be useful when defining SIGMA (see Section 2.8.4)

SPARSEG (Matrix Inversion Sparsification Control)—Simultaneous solutions (matrix methods: MATMET>0) resolve energy propagation inefficiencies present in iterative methods, but they can require excessive memory for large models.* This is especially true when radiation is present, because the coefficient (“conductance”) matrix becomes dense: each node “sees” many other nodes, albeit with weak interactions. Such weak terms are therefore automatically detected in large models in order to prepare a more sparse matrix. No terms are truncated or neglected in the sparsification process.

A more sparse matrix not only uses less memory, it is faster to invert. The trade-off is that the reduced matrix must be inverted a few more times than the full matrix (as evidenced by a higher value of LOOPCT in both steady state and TRANSIENT solutions as SPARSEG grows), but the net effect is usually an increase in computational speed. The computational savings can be significant for very large models with radiation: well over an order of magnitude reduction of both CPU time and memory.

* This is a concern for YSMP (MATMET=1 or 2) but memory is not usually a concern for AMG-CG (MATMET=11 or 12).

Small models do not benefit from this treatment. Therefore, the sparsification algorithm begins to apply for matrices* with over 100 nodes, and reaches full strength for matrices with over 1000 nodes.

The control constant SPARSEG regulates the amount of sparsification. By default, SPARSEG is 0.001, meaning that the conductances (linear or radiation) weaker than 0.1% of the linearized total conductance on any node will be removed from the coefficient matrix of very large models. While the answers will be the same within accuracy limits, the resulting LOOPCT will be larger in both steady state and TRANSIENT solutions, but the cost of the inversions performed at each LOOPCT will be less.

While the implicitness of the final solution is not affected, the reorganization of terms means that each iterative inversion is slightly more explicit, and *that* means that instabilities can result if SPARSEG is chosen to be too large. *Therefore, a SPARSEG value of 10% is employed as a maximum limit.*

TIMEND (Problem Stop Time)—The use of this control constant is self-explanatory. TIMEND must be specified as larger than TIMEO, otherwise an error message is printed and the run terminated. For the explicit routines, if TIMEND is not larger than TIMEO a time step of zero will result and the "TIME STEP TOO SMALL" error message will be printed. TRANSIENT will print the message "TRANSIENT TIME NOT SPECIFIED." If a user has some criterion for terminating an analysis, but not the run, he or she may accomplish this by setting TIMEND=TIMEO when the criterion is met. This will return control to the OPERATIONS block.

If TIMEND>TIMEO in a steady-state run, a nonzero value of the DTIMES constant is expected, and a caution will be otherwise issued.

TIMEO ("Old" Time or Problem Start Time)—This control constant represents the "old" time at the beginning of the current computation interval. It will be updated by the program after each time step is completed, however, *its initial value may be set by the user and will be interpreted as the problem start time*. TIMEO may be set negative, but must not be larger than TIMEND. If not specified by the user, its initial value will be 0.0.

4.4.2 FLUINT Control Constant Summary

Certain SINDA/FLUINT global constants (TIMEO, TIMEND, NLOOPS, ABSZRO) apply to both thermal and fluid submodels. Other global control constants apply only to fluid submodels: PATMOS, ACCELX, ACCELY, and ACCELZ. PATMOS is the pressure analog of ABSZRO. ACCELX/Y/Z is the body force acceleration vector, as described earlier.

Another class of control constants are output by the program for use by the user in the logic blocks. For fluid submodels, there is only one such constant for each submodel: DTIMUF, analogous to DTIMEU. Note that this variable is only current in FLOGIC 2 blocks; because the current time

* For MATMET=1 or 11, a matrix contains the nodes within a single thermal submodel. For MATMET=2 or 12, the matrix contains all the nodes in the thermal submodels using the MATMET=2 or 12 designation: perhaps all nodes in the model. MATMET=12 by default: a single matrix for all nodes in the model.

step cannot be computed until all user manipulations have been completed, the value of DTIMUF in FLOGIC 0 and 1 represents the previous time step; it is initialized to zero for the very first call to FLOGIC 0 or 1.

User-input, submodel-specific control constants are also available. These variables are summarized in the bottom of Table 4-2 and Table 4-4. Note that each of these constants may vary by submodel.

DTMAXF—Maximum transient time step allowable. No fluid submodel will be integrated with a time step larger than DTMAXF. The default is 1.0E30 (essentially infinite). It is recommended that the default value *not* be used. A large characteristic limit should be used instead. Note that this is the fluid analog of DTIMEH, and that there is no analog for DTIMEI, which will be used as a maximum (if positive) when fluid submodels are active.

DTMINF—Minimum *average* transient time step allowable. Transient integration will terminate after output calls if the time step is *persistently* below this value. The default is 0.0. Because of the relative volatility of fluid time step sizes, DTMINF is compared with a running average of the time step instead of the current time step. Note that this is the fluid analog of DTIMEL, although unlike that constant, DTMINF tolerates occasional wandering below the limit. No checks are performed in the first 20 time steps, and the program will only halt if the average time step is below the limit and does not appear to be increasing.

DTSIZF—Time step size factor. This parameter can be made to vary from 0.0 to 1.0 (*exclusive*) to control the size of the transient time step. The default is 0.1, which means that no important (time-dependent) network variable should change by more than about 10% during the next time step, or that no variable will change more than about 100.0*0.1/(1.0-0.1) = 11% past a transition. Increasing DTSIZF (say to 0.2 or 20%) will result in faster and probably less accurate solutions, while decreasing DTSIZF will result in slower and more accurate solutions. The user should be warned that instabilities may result if DTSIZF is too large (above about 0.5)—*increasing DTSIZF can sometimes even cause smaller time steps to be taken* for this reason. If there are no time-dependent elements, a time step of DTMAXF will be used for that submodel. The smallest time step of all active thermal or fluid submodels will be used.

DTTUBF—Time step size factor for tube flow rates. This parameter can be made to vary from 0.0 to 1.0 (*exclusive*) to control the size of the transient time step. The default is 0.1, which means that no tube flow rate (or slip flow velocity if twinned) should change by more than about 10% during the next time step. If tubes are being used for their stabilizing effects only, consider setting DTTUBF to 0.5 to allow greater time steps. DTTUBF is ignored if there are no tubes.

FRAVER—This control constant helps FLUINT decide when a flow rate is too small to consider important. By default (FRAVER=0.0), the average flow rate will be calculated internally. This value will be used to judge when a flow rate is essentially zero (less than 1.0E-9*FRAVER), and is also used to help prevent small time steps caused by paths whose flow rates are extremely small compared to the average. In systems that start from nearly zero flow rates, however, FLUINT cannot anticipate that the flow rate will eventually be greater than they are now, and therefore cannot decide that the "trickles" it is currently measuring are not important to the user. In such situations, the user can specify FRAVER to be a typical or characteristic value (lb_m/hr or kg/s, depending on UID) of interest to them, thus helping to skip over periods with small flows.

ITHLDF—Number of steady-state iterations a fluid submodel can stay dormant. This is the fluid analog of ITHOLD. If a fluid submodel converges before other submodels, it is suspended from the solution sequence and held in a boundary state until other submodels catch up or ITHLDF iterations, whichever comes first. This suspension process is analogous to the actions of the DRPMOD/ADDMOD sequence—the submodel is present as a boundary condition for other submodels, but is not updated. In such a dormant state, calls to a submodel's logic blocks (namely, FLOGIC 0, 1, and 2 and OUTPUT CALLS) continue even though the lumps in the submodel are not being updated.

ITHLDF=0 means no such suspension will occur (the default). Like ITHOLD, *this is purely a CPU time savings measure, and its use usually results in an increase in the number of total iterations (LOOPCT) required even though total CPU time might actually decrease since the average cost per iteration decreases.* The suspension and reactivation of fluid and thermal submodels are handled together, such that it is possible (and in fact usual) to have only one fluid and/or thermal submodel active during any one STEADY or STDSTL iteration.

ITROTF—Steady-state iteration output interval. Analogous to ITEROT, this is the number of steady-state iterations between calls to output routines. A value of 0 will cause output routines to be called only after STDSTL is finished. A useful debugging trick is to set ITROTF to 1 in the last few iterations. For example, in FLOGIC 0 (see also ITEROT for an analogous example in CONTROL DATA using expressions instead of Fortran):

```
IF(LOOPCT .GE. NLOOPS - 10) ITROTF = 1
```

This trick may help explain the behavior of models that refuse to converge.

OPITRF—Transient output flag analogous to OPEITR. Setting OPITRF=1 will produce output at each time step. This should be used for debugging purposes only because of the large resultant files. For example, OPITRF might be turned on if the time step falls below an acceptable level. (DTMINF fulfills a similar purpose.)

OUTPTF—Output time interval for transient runs, analogous to OUTPUT for thermal submodels. This control constant defaults to 0.01*TIMEND if fluid submodels are present, meaning approximately 101 calls to the OUTPUT CALLS block will be made during a transient. Time steps will be adjusted if necessary to produce results at the times indicated by OUTPTF or OUTPUT.

There is no global OUTPTF variable: OUTPUT CALLS, GLOBAL is controlled as a function of the OUTPUT and OUTPTF values of all active thermal and fluid submodels (Section 4.1.3.7). If "global" is used as a submodel name, translation of the next keyword is disabled.

PATMOS—Value of absolute zero pressure in user units. This control constant is used to define pressure units and is usually set to 0.0 (the default value, meaning psia or absolute Pascals), or, -14.696 (meaning psig), or -101325.0 (gauge pressure in Pascals). PATMOS is used internally in all execution routines as an offset factor for all lump pressures. *PATMOS should not be changed during processor execution (e.g., within logic blocks).* PATMOS can be any value, but should be consistent with UID for clarity.

REBALF—Relative error acceptable in steady-state fluid solutions for submodel or lump energy balance. The default is 0.01 or 1% (values smaller than 10^{-6} should be avoided). This is a convergence criterion for fluid submodels in STDSTL and STEADY. Steady-state relaxation steps will continue (subject to NLOOPS and other control constants) as long as the relative error in the submodel energy balance is greater than |REBALF|. This relative error is defined as the difference between energy inputs and outputs divided by the energy inputs (absolute value). Energy flow is calculated from lump QDOT values (whether user supplied or FLUINT supplied) and flow rate-enthalpy products in and out of plena. At steady-state, the net energy storage in the submodel should approach zero.

Nonpositive REBALF has special meanings. First, a zero value signifies that the energy balance criterion should be skipped. Second, negative values signal that the energy balance criterion should be performed on a lump-by-lump basis instead of a submodel-level basis.

A message such as "ENERGY STABLE BUT UNBALANCED" may appear if path duplication factors are used, in which case it can be ignored. To be sure, the problem can be rerun with negative REBALF as a check. If flow velocities are high, flow areas should be checked for continuity as well. See also page 61 in this section.

RERRF—Relative error acceptable in steady-state fluid solutions for key variables. The default is 0.01 or 1% (values smaller than 10^{-6} should be avoided). This is a convergence criterion for fluid submodels in STDSTL and STEADY. Steady-state relaxation steps will continue (subject to NLOOPS and other control constants) as long as the relative change in value of key network variables is greater than RERRF. Key network variables include path flow rates and pressure differentials, and lump states. Relative error is defined as the change in value between relaxation steps divided by the present value (absolute value).

RMFRAC and RMRATE—When a tank transitions from two-phase to single phase liquid or vice versa, FLUINT replaces the governing equations if liquid is treated as incompressible (for fast solution speeds). This change in equations can generate a slight error in the conservation of mass and energy. Also, when a pair of twinned tanks is first separated in anticipation of phase change (Section 3.25.7, and as controlled by RMSPLT below), slight errors are created. To avoid extremely small time steps and backed up solutions, such errors are accumulated at the time of the transition and spread over the course of the next few time steps in order to conserve mass and energy over the long term. RMFRAC and RMRATE govern this re-application of accumulated error. RMFRAC is the fraction of error to apply to the current time step, defaulting to 50% or 0.5. Values much higher than this may result in small time steps or unstable transitions, but values too small don't correct errors within an appropriately fast time scale. RMRATE is the minimum time scale over which to apply corrections, working with RMFRAC to prevent corrections from being reapplied too soon or from being delayed too long. Its default is 0.1 second (2.777778e-5 hours).

RMSPLT—As described in Section 3.25.7, when a pair of twinned tanks is in the combined (homogeneous) mode and is single phase, then if the opposite phase is injected into the tank, it may need to split before it reaches a threshold value of void fraction. This "premature" splitting generates errors in mass and energy that must be corrected over the course of the next few time steps (see RMFRAC and RMRATE above), but without this splitting the tanks would not react appropriately. Splitting only occurs if the tank would exceed the void fraction thresholds within the next time step. RMSPLT governs the anticipation splitting decision: if the tank would transition in more than 10% (0.1, by default) of the next time step, then it is split now in anticipation. Use smaller values of RMSPLT to avoid anticipation logic, larger values to invoke it more frequently.

RSMAXF—Maximum artificial time step allowable with STDSTL. This is the steady state analog of DTMAXF. No fluid submodel will be integrated with an artificial time step larger than RSMAXF. The default is one hour. *It is highly recommended that the default value not be used*, especially for networks with many time-independent elements. A smaller characteristic value should be used instead.

RSSIZF—Steady-state relaxation step size factor. Analogous to DTSIZF, RSSIZF governs the allowable change in important network parameters (that would be time-dependent during a transient) during any one steady-state relaxation step in STDSTL. The default is 0.5 (50% change), although it can vary from 0.0 to 1.0 (*exclusive*). The user should use a small value if many alterations are made in the user logic blocks or nonconvergence is observed, and may use a large value for faster convergence if few alterations are made. RSSIZF is ignored if there are no time-dependent elements. If no time-dependent elements are present, an artificial time step of RSMAXF will be used. Because these time steps are artificial, different submodels are integrated with different time steps, which is not true for transient runs.

RSTUBF—Steady-state relaxation step size factor for tube flow rates. Analogous to DTTUBF, RSTUBF governs the allowable change in tube flow rates (that would be time-dependent during a transient) during any one steady-state relaxation step in STDSTL. The default is 0.5 (50% change), although it can vary from 0.0 to 1.0 (*exclusive*). RSTUBF is ignored if there are no tubes.

4.4.3 Time and Time Step Control Constants

This subsection, while entirely redundant, serves to provide a list of control constants relating to problem time. These constants are summarized in Table 4-5, which denotes whether they are provided by the user ("input") or calculated by the program ("output"). This section does *not* describe time step control, which is detailed elsewhere (Section 4.2.1, Section 4.2.2, Section 4.4.1, and Section 4.5.2).

A call to FORWRD or TRANSIENT begins integration at TIMEO and proceeds to TIMEND at an appropriate time step. Although each thermal and fluid submodel advances at the same time step, this step is stored on a per submodel basis as DTIMEU for thermal submodels, and DTIMUF for fluid submodels. In other words, DTIMEU is equal to DTIMUF at the end of a time step.*

During a transient, the current problem time, TIMEN, is updated. During a time step, it represents the next time point, such that DTIMEU = TIMEN-TIMEO. TIMEO is also updated to reflect the starting interval of each time step. In TRANSIENT, a constant TIMEM is updated to contain the average time between TIMEO and TIMEN, as needed to support certain time-varying updates.†

A call to STEADY ("FASTIC") or STDSTL by default solves for the network at TIMEO, the start time, and time does not advance during the network solution. TIMEN, the current time, is set to TIMEO.

If DTIMES is nonzero, the problem time is incremented in STDSTL or STEADY from TIMEO to TIMEND, performing a steady-state solution every DTIMES time units with TIMEN being incremented. This option is useful for analyzing systems with little mass or capacitance, or at least fast time constants compared to boundary conditions. It is also useful for executing parametric analysis by treating some parameter as "time."

* Differences may appear during the course of a time step (in routines other than FLOGIC 2 and VARIABLES 2) since the program may still be updating these values. Also, caution should be used since the last time step used may have little to do with the next time step to be taken, and in fact DTIMUF and DTIMEU are initially zero.

† In a strict mathematical sense, TIMEM is meaningless since the user should instead provide an average of values at TIMEO and TIMEN, rather than a value at the average time TIMEM. However, TIMEM represents a convenient and often adequate short-cut.

Within STDSTL, fluid submodels are solved using pseudo-transient methods, and hence each fluid submodel may have its own internal time step DTIMUF. This “fake” time step, or “relaxation step” does not cause an advance in problem time, and is not equal to DTIMEU which has no meaning in STDSTL.

Table 4-5 Summary of Time and Time Step Control Parameters

Name	Definition	Input or Output	Type	Applicable Solutions
TIMEO	Initial time	input	global	all
TIMEND	Stop time	input	global	transients, unless DTIMES is nonzero
TIMEN	Current time	output	global	transients, unless DTIMES is nonzero
TIMEM	Current mean time in interval	output	global	TRANSIENT
DTIMEH	Highest allowable time step	input	thermal	transients
DTIMEL	Lowest affordable time step	input	thermal	TRANSIENT
DTIMES	Pseudo time step (steady state)	input	global	steady states
DTIMEU	Last time step	output	thermal	transients
DTIMUF	Last time step	output	fluid	transients and STDSTL
DTMAXF	Maximum allowable time step	input	fluid	transients
DTMINF	Minimum affordable time step	input	fluid	transients
RSMAXF	Maximum relaxation step size	input	fluid	STDSTL

4.5 FLUINT Execution and Control

Steady-state analyses of all active submodels (whether fluid or thermal) are performed by calling STDSTL or STEADY. Similarly, transient analyses are performed by calling FORWRD or TRANSIENT. Note that both FORWRD and TRANSIENT use the same solution method for fluid submodels even though the methods are different for thermal submodels. In fact, STDSTL also uses almost the same fluid solution method as that used in transients, but uses large artificial time steps to relax each network towards a time-independent solution.

STEADY (aka, “FASTIC”), which employs the same thermal method as STDSTL, uses an iterative approach for fluid models. STEADY treats tanks like junctions and tubes like STUBE connectors. Because of this treatment, some options (such as tank volume factors, twinned tanks, and ifaces) are either ignored or approximated. In general, STEADY approaches steady-state solutions very quickly but may not converge easily, while STDSTL approaches solutions slowly but enables all model options to be active. Unfortunately, the incompatible combination of full transient thermohydraulics with instantaneous thermal response (because of the treatment of diffusion nodes as arithmetic) in STDSTL means a full transient solution (e.g., TRANSIENT or “FWDBCK”) over a fake time interval is often a better choice for combined thermal/fluid steady states when advanced fluid modeling options are needed. Otherwise, STEADY is almost always the best choice for steady state solutions.

Three user logic blocks are available for each fluid submodel: FLOGIC 0, FLOGIC 1, and FLOGIC 2. A GLOBAL, submodel-independent, version of each of these FLOGIC blocks is also available. These are similar to VARIABLES 0, 1 and 2 but do not exactly correspond to them. FLOGIC 0, the primary location for most user logic, is called before each solution interval (meaning one time step for transient analyses or one relaxation solution for steady-state analyses). FLOGIC 1 is called after FLUINT has automatically calculated all required element descriptors (such as any QDOT, connector GK and HK, tube FC, etc.) to fulfill any requested or implied calculations. This call (FLOGIC 1) occurs before the time step (or steady-state relaxation step) is calculated, and is the last chance to alter the parameters before the solution begins. FLOGIC 2 is called after the solution has been performed and the network variables have been updated. In other words, the FLOGIC 0/1/2 sequence can be thought of as initialize/tailor/wrap-up. The parameters available to the user in these blocks are described in Section 4.1.3.10.

In order to allow complete access to all model parameters (for heat transfer or control simulation calculations), both fluid and thermal parameters are available in any logic block (fluid or thermal). Preferentially, however, the user should only alter fluid parameters in FLOGIC blocks, and similarly should only alter thermal parameters in VARIABLES blocks.

4.5.1 Fluid Submodel Steady-State Convergence

As noted above, the fluid network methodology in STDSTL is very similar to the transient methods used in FORWRD and TRANSIENT. In other words, the program takes artificial time steps, integrating the solution towards a time-independent state. In STEADY, the time-independent equations are solved iteratively until no significant changes are noticed. Unlike all other solution routines,

STEADY normally* solves the hydrodynamic equations separately from the energy and species concentration equations during each solution step. Both STDSTL and STEADY, however, use the same methods to guide and detect convergence.

There are two types of convergence criteria: the relative change in a certain parameter between iterations (RERRF), and the relative error in the energy flow through either a lump or the entire submodel (REBALF). "Relative" means that the error is expressed as a unitless fraction of the relevant parameter.

The following parameters are subject to the first type of check, in roughly the following order:

1) Lump PLs (absolute value, skipped for 9000 fluids)
2) Lump TLs (absolute value)
3) Lump DLs
4) Lump constituent concentrations (mass fractions)
5) Path FRs (unless the flow rate is judged insignificant)
6) Path pressure differentials (unless the flow rate is judged insignificant)
7) Path enthalpy differentials (for twinned paths, or paths with nonzero EI or EJ)

To be converged, the relative changes in all of the above parameters must be less than RERRF, which has a default value of 0.01 (1%). Typically, the first three criteria are met quickly, while the last two take more iterations to satisfy. The program loops through the elements performing these checks, and the first lump or path to fail this check is listed in the "CONVERGENCE STATUS" message that precedes most output routines (e.g., LMPTAB, PTHTAB as described in Section 7.11.8). This should not be interpreted as meaning that only this element failed the test, or that its failed by the largest margin—it was simply the first. Furthermore, lumps are checked before paths. Thus, a model that lists a lump density change as the reason for nonconvergence is not as close to a final solution as is one that lists a path pressure differential as the reason.

Upon successfully satisfying the relative change criteria, the program will attempt to satisfy the energy balance criteria, as governed by REBALF. If REBALF is zero, this check is omitted. If positive, the energy flow for the entire fluid submodel is checked by summing QDOTs into nonplenum lumps and energy flows in and out of plena. In this context, "plena" includes lumps that have been held with a call to HLDLMP (Section 7.11.1.4). The submodel may be considered converged if the energy flows balance within REBALF, or if they do not balance but are stationary and show no progress toward balancing. This latter case is normally caused by path duplication options, which can artificially magnify one the energy flows if a virtual subnetwork is not adiabatic. It can also be caused by high flow velocities and discontinuous flow areas. In such cases, the convergence message will be accompanied by a warning that "energy flows are stable but unbalanced." (Costly algorithms would be required to calculate how many times a lump *really* appears in the master model from having been duplicated.)

* When twinned paths are used, or when EI or EJ is nonzero for some path, then STEADY must solve both energy and hydrodynamic equations simultaneously.

If REBALF is negative, then the energy balance check is performed on a lump-by-lump basis, with |REBALF| being compared with the largest imbalance of any nonplenum lump. Unlike its analog EBALNA, which has units of power, negative REBALF is interpreted as a unitless (fractional) error in the energy rates for each lump.

The output routine LMPTAB (Section 7.11.8) can be used to help detect problem areas in convergence. The last two columns for each lump list the current net rates of mass and energy increase for all lumps, including plena. Upon convergence, these rates should be very small for all nonplenum lumps. Lumps with large mass and energy rates indicate possible problem areas, especially in STD-STL which uses matrix methods for simultaneous solutions. In STEADY, the energy rate for a lump may simply be off because it has not been updated recently during the iterative solution. (The STEADY energy balance uses a mixing algorithm to vary the order in which lump energy iterations proceed.)

Note that work terms are ignored in the energy balance. Work on or by the system from VDOT (volume changes) or ifaces are ignored because VDOT must eventually be zero to achieve steady-state (although the user may manipulate VDOT values during the relaxation process). Work by body forces and frictional losses are also neglected in the energy balance because they are neglected in the model itself.

Actually, the real solution procedure is considerably more complicated than it would appear from the above discussion. To help avoid property range errors and to absorb very rough initial conditions, the steady state routines start very cautiously in a sequence called “PRELIMINARY ITERATIONS” in the output message. During these iterations, the network is pushed very gently towards a solution, and the rate of convergence is compared with 10 times the values of RERRF and REBALF (i.e., 10*0.01 = 0.1, or 10% by default). Once the network satisfies these very coarse criteria, the program starts to push harder and harder, until the network is able to satisfy the real convergence criteria and the program is not being artificially gentle. (The variable degree of “pushing” or “gentleness” is achieved by scaling the driving terms, namely path HK values, in the hydrodynamic solution from their full values.) Thus, if NLOOPS is exceeded and the convergence status says “PRELIMINARY ITERATIONS: 10*RERRF NOT SATISFIED,” the user should be advised that the network is very far from being converged. On the other hand, if the solution is pushing harder and the network is responding well, a message such as “CONVERGENCE PENDING” can result.

As a result of the above methods, it takes at least two iterations to confirm convergence of a model that has already been solved. If thermal submodels are active, four iterations are required as a minimum.

4.5.2 Fluid Submodel Time Steps

In transient routines as well as in STDSTL, FLUINT calculates its own maximum allowable time step. In STDSTL, this is the step size used, and it may be different in each fluid submodel. In other routines, this time step is then compared with the step required by other submodels, the step required for output intervals, etc. The minimum of all such limits is used as the time step in transient routines. Control constants RSSIZF, DTSIZF, RSTUBF, and DTTUBF are used to help determine

the allowable time step per submodel. Constants DTMAXF, RSMAXF, OUTPTF, etc. are used to impose other limits on the time step. DTIMUF contains the result of the time step calculation for each submodel, and the output header describes its causes.

This section focuses on the first process: determining an allowable step size based on current network status, expected changes, and previous behavior. In the following discussion, “RSSIZF” may be substituted for “DTSIZF” when referring to STDSTL. DTSIZF is used as the generic term. Similarly, reference to DTTUBF also implies RSTUBF.

Background: Integration Strategies—FLUINT uses a first-order implicit time step integration that is performed in parallel with whatever method is used to integrate the thermal networks. Heat rates between thermal and fluid models are held constant to conserve energy. If all property domains and derivatives, friction coefficients, heat rates, etc. truly remain constant over the time interval, then the solution is fully implicit and an arbitrarily large time step can be taken. Since these parameters in fact often vary, a best estimate is made of the time step that can be made without excessive changes in such parameters.

Extensive logic is employed to estimate this time step and to check predicted changes against the previous step. While this feedback method successfully avoids time steps that are too small (from the mathematical standpoint if not from the user’s standpoint), the only way to be absolutely sure that this estimated time step is not too big is to proceed to integrate the equations and solve for the next network state. If unforeseen changes in operating regimes, boundary conditions, or other parameters are excessive, then the solution may back up and repeat itself with a smaller time step.

In FLUINT, the selected strategy is to spend about 10% to 20% of the cost per solution to make a good and somewhat conservative estimate of the time step, and to avoid backing up more that 10% (but usually less than 1%) of the number of steps.

Time Step Predictor—The FLUINT time step predictor faces two challenges. First, because of the implicit nature of the solution and the instant nature of many elements, time steps can only be estimated because anything can happen during the subsequent time step. Second, accuracy is crucial. If the time step is too big, errors can result; if it is too small, the program can appear to halt with no output.

Unfortunately, there is no fluid equivalent of SINDA’s CSGMIN, and there is no maximum time step due to stability criteria (although tubes can exhibit a close analogy to such a mathematical limit). The maximum time step is therefore estimated such that the assumptions implied in the current solution step are substantially true throughout the next time interval.

This estimation takes the form of imposing two kinds of limits: *percent change limits* and *transition limits*. Percent change limits keep flow rates, densities, constituent concentrations, and heat rates from varying excessively during the time interval. Transition limits, on the other hand, keep major assumptions from being violated. For example, when a tank moves from all-liquid to two-phase or the reverse, almost all assumptions, governing equations, and property derivatives change. To step too far past such a transition would result in lower accuracy at the very least, and execution errors at the worst. To try to stop exactly at the transition point is too demanding and can result in perpetually decreasing time steps. Therefore, in such cases the program attempts to step just slightly past the transition.

The key to the time step prediction is the estimation of anticipated changes. This estimate must be as accurate as possible without requiring the full network solution. Some readers may be surprised to find that up to 20% of the CPU costs are devoted strictly to time step estimation, and not to the solution itself. Because of the criticality of the selected time step, this is money well spent.

The methods involved in this estimation are among the most complex methods in FLUINT, and will only be introduced here. Detailed knowledge is not required to use the code, but an overview is provided to assist the analyst in understanding program behavior. The time step prediction procedure is as follows.

First, allowable limits (both change limits and transition limits) are calculated based on current network status.

Second, a *local solution* is performed for each time-dependent element (i.e., tanks and tubes). This local solution assumes that adjoining network elements will not change over the next time interval. The collective results of these predictions are saved in memory for comparisons with subsequent network-level solutions during the next time step.

Third, the results of this local solution are corrected using previous network behavior—a feedback loop exists. This feedback correction is somewhat subtle and perhaps counterintuitive. If the local solution from the previous time step does not agree with the current local solution, then an unforeseen change is presumed to have occurred (e.g., a valve has closed, a tank has evolved vapor or become hard-filled with liquid, etc.) which invalidates extrapolations using previous network states. In such cases, the current local solution is used and previously stored data is ignored. Otherwise, if the previous and current local solutions agree, then extrapolations of previous network-level trends can be used and local solutions ignored. A degree of confidence is assigned to the local solution compared to network-level extrapolations. One of the key purposes of this feedback control is to prevent *sticking*: continually small time steps with no appreciable progress made in the integration. Sticking would occasionally result if the program depended only upon local solutions to guide the time step prediction. Unfortunately, while this feedback helps somewhat in the prevention of catastrophically large time steps, it cannot completely guarantee their absence.

Therefore, the fourth and final step is to proceed with the integration and then check the results for signs of excessive time steps. If instabilities are detected or if changes are greater than expected, the solution will repeat itself with a smaller time step. If too many such back-ups are required, the time step will be limited to improve integration efficiencies, resulting in a message that the time step is limited by “EXCESSIVE BACK UP LIMIT.”

Tube Limits—No tube may change flow rates by a fraction greater than DTTUBF (10% by default). The local solution is estimated by taking the allowable flow rate change divided by the current derivative of flow rate with respect to time—the end point lumps are assumed to not change. To avoid getting stuck on flows through side passages, this allowable flow rate change is not allowed to be smaller than a small percent of the submodel average flow rate. If the time step is limited by a tube, the status message lists that tube, and lists the reason as “FLOW RATE CHANGE LIMIT.” For models where the stabilizing behavior of tubes is desired, but not the small time steps, consider setting DTTUBF=0.5.

For tubes for which AC*FR has become a large positive number (e.g., those at the inlet of condensers), an inherent stability limit exists which is flagged in output headers as “STABILITY LIMIT.” This limit, while fortunately rare, can only be avoided by changes to the model to reduce the rate of deceleration (condensation, suction to side flow passages, rate of increase of flow area) or increase the frictional of head (K factor) losses within the tube. If such changes are inappropriate and the time step is unacceptably small, consider STUBEs instead.

For twinned tubes, the rate of change of the slip velocity (i.e., vapor velocity minus liquid velocity) is also constrained to changes no greater than DTTUBF. If this is the cause of the time step limit, the output headers will describe it as a “SLIP FLOW CHANGE LIMIT.”

ORIFICE Limits—Orifices used to model pressure regulating valves, or valves that open or close (nonzero MODA and nonzero AORI_T) may also limit the time step, with messages such as “ORIFICE OPENING RATE LIMIT” or “ORIFICE CLOSING RATE LIMIT.”

Tie Limits—The time step can be limited by heat transfer ties to diffusion nodes unless the ties are on single-phase junctions. Because the heat rates are assumed constant over the solution interval, this limit prevents the temperature difference from changing by an unacceptable amount. Because SINDA also assumes some responsibility for the time step of the tied node, this limit is relatively liberal.

The UA for the tie is summed into the SINDA CSGMIN calculation to appropriately burden the SINDA time step calculation. (Caution: the large effective UA values typical of HTUS and HTNS ties can cause reduced CSGMIN. If this is the case, consider tying to arithmetic nodes instead.) The local solution is calculated by estimating the derivative of the temperature difference with respect to time. Unlike tube and tank limits, no feedback control is currently applied to tie limits because (perhaps unlike the elements to which they connect) ties are strictly a time-independent element. If a tie limits the time step, the reason is listed as a “NODE TEMPERATURE CHANGE LIMIT,” and the ID of the tie is listed.

Tank Limits—The limits for tanks are numerous, and vary according to the type of tank. The local solution assumes that adjacent lumps (and most adjacent flow rates) stay constant over the interval, and a reduced set of simultaneous equations is solved.

Transition limits include movement in and out of the saturation dome, the initial accumulation or extinction of constituents, and pressure gradient limits imposed by nearby capillary devices (whose behavior can change drastically when the pressure difference is near the capillary limit). Such limits are denoted “PHASE TRANSITION LIMIT” or “CAPILLARY TRANSITION LIMIT.” Also, limits are imposed on the percent change in lump mass, constituent concentration, density, volume, temperature (including limits due to any ties), energy (relative to the critical point if one exists), etc. In many of these limits, the default change limit may be less than DTSIZF, in which case the change limit actually used will be proportional to the value of DTSIZF. For example, the percent density change in a liquid tank is limited to 0.006*DTSIZF, which corresponds to 0.06% for the default value of DTSIZF. Accuracy would be questionable if the change were greater than this limit. Changes due to such effects are noted as “DENSITY CHANGE LIMIT,” “VOLUME CHANGE LIMIT,” etc. In tanks with XL=0.0 (all liquid), the temperatures and densities are directly linked, which can result in a “TEMP./DENSITY CHANGE LIMIT.”

If pressure changes in a lump are likely to affect the flow rate in an adjacent connector so much that the flow rate could easily reverse, change flow regimes, or affect the tank itself, then the time step might be reduced even though connectors themselves are time-independent. This conservative measure avoids oscillations and potentially significant errors. When this limit is encountered, the reason is attributed to the tank, and is listed as "FR CHANGE LIMIT IN PATH ___." The time step is not restricted, however, if the connector flow rate is unstable since the instability is likely to persist even at diminishingly small time steps. Furthermore, time steps are never reduced by more than 50% due to this cause.

An analogous limit may be placed on the enthalpy change of a tank if twinned STUBEs flow out of it, in which case the message will read "FR CHG LIM IN TWIN PATH ___." Similar limits are attempted for paths with nonzero EI or EJ, since their flow rates will also depend on the enthalpy (density or temperature) changes within the tank. If this is the case, the message will read "FR CHG LIM, EI/EJ, PATH ___." Once again, time steps are never reduced by more than 50% due to this cause.

Ftie Limits—Fties do not normally limit time steps directly. Rather, they influence the time step via their impact on the lump's total source term, QDOT.

Iface Limits—Ifaces do not normally limit time steps directly, although their effects on the volumes and pressures of endpoint tanks can indirectly affect time steps. However, when an iface is approaching a volume limit (VHI, VLO, VSUB, VZRO), the time step may be restricted to prevent stepping past such a drastic change in behavior. These are strictly transition limits, not change limits.

Other Limits—Often, the time step status line reports "TIME STEP GROWTH LIMIT." This message is due to the fact that the time step is constrained to grow by not more than 100% (a factor of two) each step. This limit means that some limit has previously held back the time step, but that it is now growing towards a new level. This previous limit is printed in the output routine header message. Also, for systems with no tanks or tubes, or for which the anticipated changes are negligible, the time step can be set very large and the message "NO LIMIT—TIME INDEPENDENT" printed. In practice, however, the user must intervene and place a reasonable upper limit on time step using DTMAXF.

Changes Made in User Logic—Only the effects of the latest pass through the user logic blocks (e.g, FLOGIC 0, VARIABLES 1, etc.) can be included in the time step controller; no ability exists to foresee future changes. Therefore, *it is the user's responsibility to limit the time step* (using DTMAXF or DTIMEH) *as needed to capture any important variations generated within the logic blocks*. This caution applies to time- and temperature-varying parameters as well as valve closures, etc.

4.5.3 FLUINT Trouble Shooting

It is much easier to make a nonsensical fluid model than a nonsensical thermal model. Unfortunately, FLUINT cannot always determine whether or not a model makes physical sense until it attempts to solve it, and the failure modes of nonsensical models do not always leave clues as to what modeling error was made. This section describes common mistakes and modeling problems

encountered with FLUINT. See also Section 3.20, Modeling Tips for Efficient Use.

Property Routine Errors—Excursions beyond the limits of the property library may cause either temporary or fatal problems. (Note that error messages indicating such problems will occasionally use English units even if SI was specified since many standard library fluid internal routines work strictly in English units.) Because such excursions are common during steady-state runs, FLUINT will first attempt to reset the properties to their limits and continue. However, too many or too severe errors will force an abort. Before suggesting ways of avoiding these limits, the reader should be familiar with the property range limits (see Appendix C for library fluids, and Section 3.21 for user fluids).

Some of the common sources for these errors (assuming that the system is not expected to operate outside of the range limits and is not overconstrained) include:

1) The default value of MCH=-2, the path choked flow methodology flag, may require metastable fluid properties and low throat pressures (below the range limits). Consider suspending choked flow detection and modeling (MCH=0), using the Dyer method (MCH=-3), or at using equilibrium expansions (MCH>0). Refer to Section 3.18 (and subsections Section 3.18.1.3 and Section 3.18.1.5 in particular) for details.

2) Large flow resistances (long lengths, small diameters or flow areas, large K factors) combined with unyielding pump models including VFRSET and MFRSET connectors, or overconstrained flow rates. In such cases, the pressures may remain above or below acceptable limits in order to attempt to force the flow through the sections with large resistances. Excessively large lengths (TLEN values) alone can cause these problems for *compressible fluids with UPF less than unity*: the axial resolution is too coarse to resolve the internal pressure gradient.

3) Constant heat source terms combined with inadequate flow rates. If the QDOT value on a lump is nonzero and the flow rate through the lump becomes too small, the fluid in the lump will become either excessively superheated or excessively cold. This error is common if flow rates drop suddenly in tied junctions, as noted below, or if the flow rates oscillate by too large a magnitude. Flow rates can be steadied by eliminating competition among parallel paths, by adding tubes to constrain the changes, etc.

   If the flow rates are much lower than expected, check for (1) excessive resistances or large resistances compared to parallel paths, (2) pump degradation because of two-phase flow, and (3) depriming of a capillary device. Note that VFRSET and PUMP models will quickly degrade if vapor is present at the inlet. While this is a realistic concern in most transients, it may be simply a pest in some steady-state runs. In the latter case, use a plenum or call HLDLMP to maintain subcooling in the inlet lump or use an MFRSET model.

4) All or a portion of the network has a defined, constant volume (even though it may be two-phase) and heat is added or subtracted until a pressure over or under limit is reached. Again, this may correspond to an actual problem in the real system.

5) An invalid state has been specified in a call to CHGLMP, CHGLMP_MIX, or in FLOW DATA. Check current values of ABSZRO and PATMOS. For FLOW DATA errors check PLADD, PLFACT, TLADD, and TLFACT, and check to see what the default PL, TL, and XL values had been defined in previous LU DEF subblocks. Another common source of related errors is calling property routines (e.g., VSV and VTS) with values of temperature and pressure that are not in absolute units, or with fluid identifiers (the last argument) improperly defined or not allowed to be translated.

6) If the flow is compressible, the default value of UPF=0.5 for tubes and STUBE connectors may cause instabilities that drive the downstream pressure too low. Setting UPF to a higher value (say, UPF=1.0) avoids this numerical problem.

The user should note that “temperature too high” and “temperature too low” messages can be monotonously common for tied junctions when the flow rate can vary, especially in two-phase flow where heat rates are high and flow resistances can change quickly. These can be ignored unless they are persistent.

Oscillations and/or Hysteresis—There are many possible ways of arriving at different solutions when starting from different initial conditions, or getting long-term oscillations in the solution. The causes of these phenomena are sometimes physically based (as with capillary devices) and are sometimes artificially induced by mathematically sharp limits or one-to-many relationships that frequently occur in pressure drop and heat transfer correlations. Some causes are introduced purposely to enhance stability (e.g., flow regime mapping and capillary prime).

By the nature of its fully implicit solution and by using small artificial hysteresis in certain device models, FLUINT avoids most short term oscillations. Most of the large term oscillations and hysteresis are either physically based or are usually of no consequence to the analysis requirements. However, a few of the causes of these effects are outlined here for the user’s understanding. More importantly, the user should be aware of similar effects caused by his or her own logic.

1) Capillary devices (CAPIL connectors and CAPPMP models) may exhibit fundamentally different behavior depending on the presence or absence of a capillary interface. Once deprimed, certain conditions must be met in order to reprime as is the case with real systems. This may be the cause of arriving at different answers from different directions. Note also that FLUINT uses small artificial hysteresis in the capillary pressure limit and the tank qualities to maintain stability over the short term. This hysteresis is on the order of 4% (real devices can exhibit hysteresis on the order of 50%), and hence should not be noticeable under normal circumstances. The routine SPRIME (Section 7.11.1.7) called from FLOGIC 0 can be used to force the device to stay primed, and the RC, XVH, and XVL descriptors of CAPILs and CAPPMP can be adjusted to make deprime less likely. One common cause of transient oscillation is the deprime of a capillary device due to excess pressure gradient where the liquid side is a subcooled tank (in the absence of a WICK iface). Refer to Section 3.5.3, Section 3.9.3, and Section 3.11.2 for more details.

2) Heat transfer and pressure drop correlations can cause both hysteresis and oscillations. For example, accurate modeling of the transition region for single-phase flow means that one friction factor can be achieved at three different Reynolds numbers: (1) a laminar and (2) a turbulent number where small increases in Reynolds number corresponds to decreases in the friction factor, and (3) a transition number where the friction factor increases with increasing Reynolds number. While this presents no problem for quasi-steady solutions, it may cause some oscillations for fast changing solutions. As another example, heat transfer coefficients are orders-of-magnitude greater in two-phase single-constituent flows than single-phase or multiple-constituent flows. Therefore, when heated past the saturation limit, a lump will tend to stay in the two-phase regime because of the greater amount of heat acquired in this mode. The converse is true for condensation—some oscillations may occur as a lump transitions back and forth between a fast-cooling two-phase state and a slow-cooling single-phase state. Again, such effects may be visible during fast changes and steady-state solutions, but usually damp out for gradual changes.

3) In diabatic two-phase ducts, the spatial accelerations (forces due to velocity gradients) can cause long term oscillations, especially when such ducts are in parallel. Density (void fraction) oscillations are also common when using twinned paths. These accelerations are responsible for most oscillatory behavior in real evaporators and condensers. However, the user should not take such oscillatory predictions too literally because of the many approximations inherent in such analyses.

4) As described in Section 3.16, flow regime determination logic that is invoked by specifying IPDC=6 (the default) favors the current regime when there is doubt. This can result in substantial hysteresis.

5) Transitions from two-fluid to one-fluid modeling (slip flow to homogeneous with twinned paths, and nonequilibrium to perfectly mixed with twinned tanks) purposely contain hysteresis for stability.

6) Choked flow simulation and detection methods (Section 3.18) purposely introduce hysteresis for stability, although there is some physical basis for two-phase flow.

Many problems with oscillations in two-phase modeling can be avoided (1) by using UPF=0.5 instead of 0.0 or 1.0, (2) by using C options instead of U or D in LINE and HX macros, (3) by using duplication options to avoid resolving interactions amongst parallel passages, and (4) by avoiding twinned paths if slip flow modeling is not necessary. The first two modeling decisions cause pressure drop and/or heat transfer calculations to be based on more than one lump or path, thereby lessening the impact of a change in one element and avoiding analytical trouble caused by severe gradients. The remaining decisions avoid unnecessary detail.

Many oscillations can be caused by rough user logic. Refer to the section on Modeling Tips for more details (Section 3.20)

Matrix Solution Errors (YSMP)—Errors encountered during matrix inversion have error messages containing “YSMP” (Yale Sparse Matrix Package). This usually signals that at least one portion of the network is over or under constrained (see “Smoothing sensitive systems” under Modeling Tips), or that a junction loop exists (see Section 3.3.2). The cause of the error is printed in terms of the equation (mass or energy) and the lump in which the problem was detected.

Such errors can occur when all or an isolated portion of the network contains only hard-filled rigid tanks (COMP=XL=0.0) and/or junctions, or when the flow rates are overconstrained by too many “stiff” (GK=0.0) connectors. A plenum or “soft” tank (COMP> 0.0 or XL > 0.0) must be present to provide a reference pressure or to absorb volume changes. At least one plenum is required for single-phase liquid models. Note that while an error message will be written if the entire network encounters such a condition, no such predictions are possible when a portion of the network is isolated from an otherwise satisfactory network by closed valves or other stiff connectors, as described below.

An isolated portion of the network is not just a region completely surrounded by closed valves. It is more loosely defined as any section of the network whose boundaries have prescribed flow rates (which includes the case of a prescribed zero flow rate corresponding to a closed valve). Paths that can prescribe flow rates include VFRSET and MFRSET connectors, ORIFICE and CTLVLV and CHKVLV connectors when closed, and any primed capillary connector (CAPIL connectors, NULL connectors in CAPPMP models). Such connectors are said to be *stiff* because their flow rate is insensitive to pressure gradient changes: GK = 0.0.

Examples of such overconstraining include more than one of the above connectors on a single line with no side branches, plena, or soft tanks in between. Note that a three-way control valve model should always leave one branch unconstrained to avoid such problems.

Another cause of YSMP failure is an undersized workspace. If this is the case, FLUINT will so indicate in a processor error message, and will describe how to remedy the problem using the NWORK keyword on the HEADER FLOW DATA subblock (see Section 3.2.2).

The workspace required by the YSMP package cannot be known beforehand, and insufficient allocation can result in an aborted run. Furthermore, the actual usage can vary significantly with seemingly small changes to the model. It is also a function of the solution routine used (STEADY requires about a third of the workspace required by other solution routines) and the changes that occur in the model during a transient integration. Therefore, the preprocessor automatically estimates the required allocation, then applies considerable conservatism to avoid failed runs.

If the user believes the workspace allocations to be excessive and is willing to assume the increased risk of an aborted run in exchange for smaller memory requirements, the NWORK keyword can be used to turn off the automatic calculation and to reduce the allocation even if no aborts have occurred. The YSMPWK routine (which has no arguments) should be called at the end of OPERATIONS to report actual memory usage. Such information from previous runs can then be applied to future runs.

4.6 Overview of Processor Output Operations

SINDA/FLUINT produces both binary and text (ASCII) output files of various types. The methods for declaring the file names and controlling their content are described elsewhere; *this section is wholly redundant.* However, because these descriptions are provided in several different locations within this manual, this section provides a centralized summary that is especially helpful to new users.*

There are many types of output files serving various uses, but they all can be classified as one of two categories: binary or text. Text files can be opened by editing programs and viewed by the user, whereas binary files are compact representations that can only be read by programs intended for that purpose (namely, SINDA/FLUINT itself, its graphical user interfaces, or postprocessors). Almost all of these files are named in OPTIONS DATA (Section 2.7.1).

By default, SINDA/FLUINT produces no processor output at all. It is the user's responsibility to declare what information is needed in the form of output routines added to any (or each) submodel's OUTPUT CALLS logic block (Section 4.1.3.7). Often, either the GLOBAL output call block or that of a single submodel (either thermal or fluid, or perhaps one of each) is chosen to contain these output instructions, and the routine calls are made using options such as 'ALL' that request the output from all active submodels.

4.6.1 Text Output

Text files can be opened with editing programs (e.g., Microsoft's Notepad, WordPad, or Word), and viewed by the user. The main file is the OUTPUT file, although any number of user files may also be opened.

4.6.1.1 The OUTPUT File

The OUTPUT file is named in OPTIONS DATA (Section 2.7.1). It will contain summary messages and warnings in a 132 column format. It will also contain the results of any user-requested text-based output routines, such as NODTAB, TPRINT, and LMPMAP.

The OUTPUT file can be changed during a run using the CHGOUT utility (Section 7.4.19).

4.6.1.2 OUTPUT File Content: Common OUTPUT CALLS Options

To invoke output to the OUTPUT file, one or more routines must be invoked in the appropriate OUTPUT CALLS logic block (Section 4.1.3.7). In fact, *most* output routines write to the OUTPUT file. Section 7.4 describes SINDA (thermal submodel) outputs, and Section 7.11.8 describes FLUINT (fluid submodel) outputs.

* A template input file, template.inp, is available in the *samples* subdirectory for SINDA/FLUINT-only users. It contains a typical set-up for an empty model. The template function for new Sinaps® models fulfills the same function, as does the Case Set Manager in Thermal Desktop®.

Common routines include:

NODTAB. . . . . A tabulation of current node data
LMPTAB . . . . . A tabulation of current lump data
PTHTAB . . . . . A tabulation of current path data

An example OUTPUT CALLS block might appear as follows:

```
header output calls, udaman
      call nodtab
      call submap
```

In addition, the user may write their own messages to the OUTPUT file using the variable NOUT, which contains the Fortran unit number for the OUTPUT file. For example:

```
      write(nout,*)' Starting the transient ...'
```

4.6.1.3 Output Frequency

By default, each active submodel's OUTPUT CALLS will be invoked at the start and end of each steady state solution, and every 1% of TIMEND for each transient solution. This frequency can be adjusted using the ITROUT and ITROTF control constants for steady-state, and OUTPUT and OUTPTF control constants for transients as described in Section 4.4.

If an OUTPUT CALLS, GLOBAL block is defined, note that it is invoked any time any submodel-level block is called (even if empty) after all such submodel-level blocks are controlled. In other words, the global-level block responds to any and all submodel-level control constants such as OUTPTF or ITROUT.

To suspend the initial echo of data for a steady-state or transient, the user can use the value of LOOPCT as a signal to return without doing any output operations the first time:

```
header output calls, udaman
      if(loopct .eq. 0) return      $ turn off initial echo
      call nodtab
      call submap
```

When using parametric sweeps or advanced solutions such as optimization or statistical sampling (Section 5), other steps may be necessary to limit output, including integer registers used as flags to disable/re-enable output as desired. Alternatively, output calls can be moved to OPERATIONS (Section 4.1.3.1), or to the OUTPUT CALLS of special submodels whose frequency is reduced or which are empty/fake and which are built (BUILD or BUILDF, Section 4.1.1.1) just to control output.

4.6.1.4 User Files

In addition to the main OUTPUT file, any number of user files may be opened (and in fact, used for input as well as output). Two files, USER1 and USER2, are predefined though more can be added via the USRFIL utility (Section 7.4.17). USER1 and USER2 file names may be provided in OP-

TIONS DATA (Section 2.7.1), and may be read or written to (as Fortran unit numbers NUSER1 and NUSER2) in any logic block, though usually OUTPUT CALLS will be used (Section 4.1.3.7).

As an example, text output normally sent to the OUTPUT file can be temporarily redirected to a user file (though line counts will be lost for printer control):

```
ntest = nout      $ store the value of NOUT temporarily
nout  = nuser2    $ send text to the USER2 file instead
call nodtab       $ a node summary, redirected to USER2
nout  = ntest     $ output resumed to the OUTPUT file
```

Binary user files may be opened using a user-supplied Fortran OPEN statement. A special UNITONLY mode of USRFIL is available to locate an unused unit number for such purposes.

4.6.2 Binary Output

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.*

A relatively small number of output routines produce binary (machine readable) instead of ASCII text. Most of the above discussion (Section 4.6.1) is applicable to binary as well as text output, and so won't be repeated in this subsection. For example, binary output routines are normally invoked from the OUTPUT CALLS and are therefore subject to the same frequency considerations. However, each binary routine produces content that is directed to a special file, usually unique to that routine.

The most common routine is the SAVE routine, which writes to the SAVE file or folder named in OPTIONS DATA as summarized in Section 4.6.2.1. However, other binary routines exist too, including:

SAVE . . . . . Principal binary output routine, writes to the SAVE file (summary in Section 4.6.2.1, details in Section 7.4.5). Writes instead to a CSR folder named in SAVE if USECSR has been invoked.

RESAVE . . . . . For restart purposes (Section 7.4.3 and Section 4.7), writes to the RSO file, which may be renamed as the RSI file for subsequent runs. The RSI file is read by the RESTAR routine (Section 7.5.4). Each of these "files" may alternatively be a CSR folder if USECSR has been invoked.

CRASH. . . . . For crash recovery, overwrites the same snapshot to the file or folder named in the call (Section 4.7.4).

SAVEDB. . . . . Cumulative saves to the REDB file or folder, read by the RESTDB routine

SAVPAR . . . . . Saves to an internal (temporary) file for resetting within one run (Section 7.4.6 and Section 4.7) using the RESPAR routine (Section 7.5.3)

SVPART . . . . . Selective version of SAVPAR

These files use a common format, enabling important usage scenarios as described in Section 4.6.2.2.

4.6.2.1 The SAVE File or Folder

The SAVE file is named in OPTIONS DATA (Section 2.7.1). If USECSR has been invoked, the file becomes a Compressed Solution Results (CSR) folder instead.

Each time the SAVE routine (Section 7.4.5) is called, it adds a sequential "snapshot" or *record* in binary format. The content of each snapshot can be adjusted, but usually (as long as disk space allows) all possible data is saved:

```
call save          $ shortened form of CALL SAVE('ALL',0)
```

The primary purpose of the SAVE set is to provide data for external postprocessing programs such as Sinaps® (page xlii), Thermal Desktop® (page xliii), EZXY® (page xlvi, perhaps via the Microsoft Excel™ interface). Usually, each call to SAVE is produced at a different time point, such that the postprocessing programs will initially assume that each record represents a different time point. (This assumption can be overridden for parametric sweeps, as an example, where each record might represent a solution at a different value of a particular register.)

The SAVE file or folder can be changed during a run using the CHGSAVE utility (Section 7.4.19).

4.6.2.2 Different Files with a Common Format

The format of the traditional (single) SAVE file is not *proprietary* per se, but it is undocumented since changed between SINDA/FLUINT versions. The format of a CSR folder, on the other hand, is documented and can almost always be used even if the format changes, since the changes often take the form of newly added parameters. **The traditional single-file format is being obsoleted: call USECSR to use the newer CSR folder format.**

Whether a file or folder style, these formats are intentionally exactly *the same for all binary input/output routines.* For example, an RSO output can be post-processed just as if it were a SAVE output, and the SAVE file or folder can be named as an RSI file or folder, and then used to restart a run.

Important usage scenarios enabled by this interchangeability include:

1. A run in progress can be recovered using the results of any routine: SAVE, RESAVE, CRASH, or SAVEDB.
2. Any file or CSR folder can be used for postprocessing. For example, one might save a lot of data (e.g., the 'ALL' option) infrequently (e.g., in a submodel with a large OUTPUT or OUTPTF value) to a RESAVE location, and at the same time one might save a little data (e.g., just the 'T' option for nodal temperatures) frequently (e.g., in a separate submodel's OUTPUT CALLS using a small OUTPUT or OUTPTF value)
3. Any binary file or folder from a prior run can be named as an REDB file or folder, and then more data from a new run can be appended using SAVEDB.

4.7 Multiple Case/Multiple Run Operations

The SINDA/FLUINT system includes the capability to perform parametric analyses during a single run as well as restarting subsequent runs from saved data. These two capabilities are independent of each other. Parameter runs can be made with or without restart and a run may be restarted with or without the parameter options being used. The one restriction that applies to both of these capabilities is that neither works on a per submodel basis. The entire set of active and inactive submodels will be both saved and retrieved, although the user can be selective about the *type* of data.

With regard to restarts, *the structure of the network must be the same when the data is retrieved as it was when it was saved*. The values of any network parameters may be changed, but no network elements, user constants, arrays etc. may be added or deleted, and the input order cannot change. Otherwise, a fatal error will be detected, with a message detailing the reason for the collision. However, collision checks cannot detect changes in input order. (An alternate restart routine is available to circumvent such collision checks if the user wishes to assume responsibility for potentially catastrophic failures in exchange for greater modeling power.) Parametric capabilities, of course, are not affected by this restriction because they are performed within a single run.

This section describes means of saving and retrieving data both between runs and within a single run. Registers are an important facilitator for performing parametric and sensitivity studies. Section 2.8 describes how to make sweeping changes to the model itself between runs, and Section 4.9 describes how to make these changes within a single run.

4.7.1 Multiple Cases Within a Single Run: SAVPAR, SVPART, RESPAR

Frequently, a user will want to run multiple cases of the same basic thermal or flow problem to see, for example, how his or her model responds to changes in boundary conditions. This is generally an efficient way to operate because preprocessing, compiling and loading operations are not repeated. However, since all network solutions in SINDA/FLUINT are cumulative, a separate means is used to return some or all of the network to a previous state. This parametric run capability is accomplished using subroutines SAVPAR, SVPART, and RESPAR.*

SAVPAR represents a means of taking a *complete* "snapshot" of a model and its status at any time within the processor execution. This snapshot may then be restored for further analyses through the use of RESPAR. SAVPAR saves the entire network state, which forces the erasing and rewriting of the entire network state in a subsequent call to RESPAR. An alternate to SAVPAR, SVPART, is available for selecting which properties of a network (e.g., temperature only, capacitances only) should be saved to avoid overwriting other parameters.

When SAVPAR is called it writes out all the current values of the T, C, Q, G arrays, all fluid submodel data, and the values of the control constants, register values, user constants (numbered, but not named) and user arrays. Each call to SAVPAR will return a unique record number. These record numbers serve as identifiers for the snapshots, and should be stored in registers, user variables,

* Although intended for support of the Reliability Engineering module, the routines SAVEDB and RESTDB, documented in Section 5.23 and Section 7.14.2, can also be used for this purpose.

or in a user array for subsequent use in subroutine RESPAR, which requires that record number as an argument. Incorrect arguments will either result in a collision error message or restoration of the wrong snapshot. After a RESPAR call, all the data in the processor has been restored to the values it had when the corresponding SAVPAR or SVPART was called.

SAVPAR writes out a lot of data. For this reason it should probably never be called from a VARIABLES or FLOGIC block. SAVPAR is most suited for calls from OPERATIONS, PROCEDURE, or OUTPUT CALLS. It should only be called from a VARIABLES or FLOGIC block if the number of calls are restricted by user logic.

One possible mode of parametric operation is as follows:

1) save all data: CALL SAVPAR(NTEST)
2) execute solution routine: CALL TRANSIENT
3) restore all data: CALL RESPAR(NTEST)
4) change data values as required: RTG.C10 = RTG.C10*1.1
5) reset time to zero, and repeat 1 through 4 as required

SVPART operates analogously to SAVPAR, but allows data to be saved selectively. (Refer to the arguments listed for RESAVE in Section 4.7.3, or to the description of SVPART in Section 7.) This not only reduces the amount of data saved, it prevents the erasure of data that was not meant to be changed as part of the parametric analysis.

These three routines are documented in Section 7.

4.7.2 Multiple Cases Within a Single Run: DTIMES

Often, steady-state analyses are adequate for sizing a parameter or for investigating performance sensitivities to variations of it. In these cases, a convenient means of running parametric analyses can be exploited by treating the independent variable as “time.”

As described in Section 4.4.1 and Section 4.4.3, a single call to a steady state routine (STDSTL or STEADY*) will result in multiple steady-state solutions if DTIMES is nonzero and TIMEO is less than TIMEND. The problem time (TIMEN) will be advanced from TIMEO to TIMEND in intervals of DTIMES, with a complete steady-state solution being performed at each point.

While the DTIMES option is intended to be used to analyze pseudo-transients, or systems whose time constants are orders of magnitude less than those of time-varying boundary conditions, it may also be used to execute parametric analyses if the independent variable of interest (e.g., a linear dimension, a material property, etc.) is substituted for the problem time. Specifically, the minimum value of a parameter is specified as TIMEO, the maximum as TIMEND, DTIMES is chosen to provide resolution within that range, and a steady-state run is initiated.

* If fluid submodels are active, use STEADY (aka, “FASTIC”) instead of STDSTL for such parametrics.

The variations in the independent parameter must then be imposed on the network. User logic can be written in VARIABLES 0 or FLOGIC 0 to apply the current value of TIMEN to the network parameter(s) of interest (e.g., G values of conductors, Q values of nodes, etc.). The easiest way to impose these updates is to have the inputs defined on the basis of registers, set the value of the pertinent register to be TIMEN in VARIABLES 0 or FLOGIC 0 followed by a call to UPREG, as explained in Section 4.9.

Otherwise, without registers Fortran DO LOOPs can be used. Alternatively, the constants in SIV conductors (for example) can be specified as numbered user constants, and the values of these constants changed with a single Fortran statement, thereby indirectly changing all relevant conductors. Where possible, it is convenient to use time-varying conductance (TVS, Section 2.14) or time-varying source (TVS, Section 2.13) options, and let the program update the variables automatically.

As an example, consider a support strut in a vacuum environment. The low-conductivity strut holds a cryogenic vessel suspended from a warm shroud or chamber wall. It is desired to find the value of emissivity (e.g, the coating properties) that minimizes the heat leak into the vessel.* The emissivity is defined as a register and used in the definition of the radiation conductances. A single call to STDSTL is made, with TIMEO=0.0 (zero emissivity) and TIMEND=1.0 (emissivity of unity). In VARIABLES 0 or 1, the emissivity-defining register is set to TIMEN. DTIMES, which must be nonzero, is chosen to be 0.05, resulting in 21 steady state solutions.

An important benefit of this approach is ease of postprocessing. Most postprocessors, including EZXY® and Sinaps® (separate manuals), are designed to provide tables and plots from SAVE or RESAVE (RSO) files and folders (Section 2.7 and Section 7.4) as functions of time (by default). By treating an independent variable as “time,” postprocessing results is much more convenient.

Each steady-state solution will cause OUTPUT CALLS to be called (as a minimum) at the beginning and the end of each solution. To avoid repetitive outputs (e.g, the same values output at the end of one solution being repeated at the start of the next solution), add the following line at the start of OUTPUT CALLS for the relevant submodels (or in the OUTPUT CALLS, GLOBAL block):

```
IF(LOOPCT .EQ. 0) RETURN
```

This causes all subsequent operations in that block to be skipped at the start of each steady-state run.

4.7.3 Restart Operations Between Runs

Restart operations involve saving the data values from one run and then reusing them in another run. For example, one run could perform a one hour simulation and save the appropriate data values. Another run could read in these values and perform the second hour of the simulation. Alternatively, steady-state results could be saved as for use as initial conditions in different transient analyses.

* Experienced users will find the Solver, described in Section 5, more convenient for such tasks, with the emissivity of the strut being named as a design variable and the heat leak into the tank defined as the OBJECT.

The restart capabilities are invoked through two routines: RESAVE and RESTAR.* To use restart one or both of the following lines must appear in OPTIONS DATA:

```
RSO  =  filename or foldername
RSI  =  filename or foldername
```

The RSO line must appear in order to create a restart file or folder. The RSI line must appear in order to read data from a previously created restart file or folder. Both lines must appear in order to read one restart set and create another in the same run.

The restart file or folder (RSO) is written using RESAVE routine. This routine has one argument: a string composed of one or more of the following letters:

T . . . . . to save temperatures
C . . . . . to save capacitance
Q . . . . . to save Q values
G . . . . . to save conductor values
A . . . . . to save user arrays
U . . . . . to save user constants (numbered and named)
N . . . . . to save control constants
L . . . . . to save lump PL, TL, XL, and QDOT (for plotting)
P . . . . . to save path FR (for plotting)
F . . . . . to save most of remaining FLUINT data (*necessary for restarts*)
M. . . . . to save advanced (mixture, etc.) FLUINT data (*necessary for restarts*)
R . . . . . to save registers (values but not expressions)
ALL . . . . . to save all of the above

Alternatively, the user can use a minus sign prefixing the list of letters to mean "save all *except* the following."

Examples:

```
CALL RESAVE('ALL')     $ save everything†
CALL RESAVE('TCQ')     $ save Ts, Cs, and nodal Qs
CALL RESAVE('TCQGAUN')
CALL RESAVE('-R')      $ all but register data saved
```

When RESAVE is called it will write a record number to the output file with the current problem time, as illustrated by the following example:

```
RESTART RECORD NUMBER 1250 WRITTEN FOR TIMEN = 1.75
```

* Although intended for support of the Reliability Engineering module, the routines SAVEDB and RESTDB, documented in Section 5.23 and Section 7.14.2, can also be used for this purpose. Uniquely, SAVEDB can be used to add to a previously created SAVE, RSO, or REDB file. (All of these files use the same format: a file created for one purpose can be used for another in a later run.)

† Omitting the argument (e.g., "CALL RESAVE") is equivalent to using 'ALL'

RESAVE will print a different record number each time it is called. This record number must be used as an argument in any subsequent runs calling RESTAR. Calling RESTAR with the appropriate record number will restore the data that was saved at that time. If a number of different saves were made with RESAVE, a number of different restarts can be made by calling RESTAR with different arguments. The same restart can be repeated several times or a series of different restarts may be performed from the same RSI data set. Incorrect RESTAR arguments will either result in a collision error message or restoration of the wrong snapshot.

RESAVE writes out a lot of data. For this reason RESAVE should probably never be called from VARIABLES or FLOGIC blocks. RESAVE is most suited for calls from OPERATIONS or OUTPUT CALLS. It should only be called from a VARIABLES or FLOGIC block if the number of calls are restricted by user input logic.

An example of the restart operation is as follows:

A) Run 1
   1) perform solution routine: CALL TRANSIENT
   2) save data in OPERATIONS or OUTPUT: CALL RESAVE('ALL')

B) Run 2 (assume *for this example* that previous RESAVE created record 15944)
   1) restore data CALL RESTAR(15944)
   2) change data TIMEND = 5.0
   3) use data CALL TRANSIENT

Caution—The actions of the routines DRPMOD, PUTTIE, and PUTPAT are not preserved from a previous run. In other words, RESTAR does not affect either the thermal submodel dormant/awake status nor the location of the ties and paths.

Caution—Named user constants are never saved nor retrieved: they are intentionally invariant since they are intended for use in controlling restart and parametric operations.

Caution—The current *expressions* defining data, including those defining registers, are not saved. When a model's data is retrieved and the read-in value of a register is different from the current value, that value will overwrite any expression used to define the register.

Guidance—When RESTAR is used within a PROCEDURE or RELPROCEDURE block, as called from within parametric sweep routines (PSWEEP, DVSWEEP, RVSWEEP), design space scanning routines (DSCANLH, DSCANFF), the Solver, or reliability engineering modules (SAMPLE, DSAMPLE, RELENG), then after restoring the data and before returning control to the user, any expressions containing key registers (e.g., swept variables, design variables, random variables) are automatically updated to avoid conflicts with the driver routines.

4.7.4 Crash Files: Abort Recovery

Crash files (or Crash *folders* if USECSR has been invoked) are an alternative restart option that are particularly suited to helping the user recover from unexpected termination of the run, while avoiding the huge files that would be generated by repeated calls to SAVE or RESAVE.

Crash files or folders are created and updated via calls to a single routine, CRASH, which 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 file sizes 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 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 empty block enables the user to customize the frequency of calls to CRASH.

Crash files are closed after each call, and are reopened with STATUS='UNKNOWN' on the next available unit number. 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 or folder name input as the RSI specification in OPTIONS DATA.

CRASH files and folders are not named in OPTIONS DATA. The sole argument to the CRASH routine is the file or folder name to be used, entered either in single quotes or as a CARRAY reference. On case-sensitive machines (e.g., Unix), use the CARRAY option to preserve lower case letters in file names. Otherwise, the file or folder name will be capitalized.

Examples:

```
CALL CRASH('OHNO')      $ LOCAL FOLDER NAME
CALL CRASH(UCA10)       $ FOLDER NAME INPUT AS CARRAY 10
```

Caution—Calls to CRASH may appear in more than one place in the model *as long as the file or folder names are identical.* Otherwise, fatal errors may result upon attempting to restart.

4.8 Controlling Submodel States

One of the most powerful enhancements of SINDA/FLUINT over prior versions of SINDA is the ability to use submodels. **Users of old SINDA-style codes should pay particular attention to this section!** Submodels are not simply organizational tools, nor are they merely methods for avoiding node and conductor number collisions. The user can perform many manipulations at the submodel level, permitting a wide range of analyses not possible with old SINDA. Many clever tricks are available to the user who masters these manipulations.

While this material is also presented elsewhere in several sections, it is important enough to be collected and summarized in this section. Refer also to the sample problems (separate volume), especially the first sample problem dealing strictly with thermal submodels.

BUILT vs. NOT BUILT—The BUILD and BUILDF commands define the list of currently active thermal and fluid submodels, respectively. This list is called the *configuration.* These commands may be repeated as necessary within OPERATIONS, PROCEDURE, and/or RELPROCEDURE.

Any submodel not built into the current configuration is treated as an adiabatic boundary condition (e.g., as if it does not exist) from the viewpoint of any other submodel referencing nodes or lumps in that submodel. Any submodel that is *never* built is treated as if it were never input—the input blocks for that submodel are not even preprocessed and its arrays, constants, and network parameter data (T, PL, UA, etc.) are not available. If the submodel is eventually built but is not present in the current configuration, its arrays and data are available in logic blocks but are not updated or accessed by any solution routines. Thus, the first time a submodel is built, it will contain all values input in the data blocks. The next time a submodel is built it will contain the values it had from the last time it was built. The only way to remove a submodel from the current configuration is to build a new configuration that omits it.

DORMANT vs. NOT DORMANT—Thermal submodels may be dynamically placed in a dormant state (“dropped”) or re-awakened (“added”) using the DRPMOD and ADDMOD routines. DRPMOD (Section 7.6.10) is used to drop a submodel from the calculation sequence; subroutine ADDMOD (Section 7.6.11) reverses that action. Dormant models are not the same as inactive models (i.e., those not currently built). Whereas nodes in inactive models do not exist from the viewpoint of active models, nodes in dormant models are treated by other submodels as if they were boundary nodes. Hence, a dormant model may be said to be in a boundary state. Like BUILD and BUILDF commands, DRPMOD and ADDMOD may only be used in OPERATIONS, PROCEDURE, and RELPROCEDURE. (Caution: the actions of DRPMOD are not preserved from previous runs—the actions of RESTAR or RESTNC do not affect dormant status.) 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.*

There is no fluid submodel equivalent to DRPMOD and ADDMOD. However, the HLDLMP and RELLMP routines (Section 7.11.1.4) perform an analogous function on a per-lump basis and may be called each iteration (rather than only within OPERATIONS, PROCEDURE, or RELPROCEDURE). (The thermal submodel equivalent to these routines are HTRNOD and RELNOD.) Of

course, an entire fluid submodel could be placed in a boundary state with repeated calls to HLDLMP, and a single node could be placed in a boundary state by isolating it in one submodel and then calling DRPMOD.

4.9 Dynamic Registers: The Underlying Spreadsheet

The most powerful enhancement of SINDA/FLUINT since the advent of submodels is the ability to use registers. This usage has been significantly expanded in this version by the inclusion of processor variables (SINDA/FLUINT input and output parameters such as “smn.T22”), which enables complex interrelationships to be defined between various inputs, and between inputs and outputs. **Users of old SINDA-like codes or older versions of SINDA/FLUINT should pay particular attention to this section!**

There are many important reasons for making copious use of registers to define a model algebraically. This section describes one such reason: the ability to easily make sweeping changes in a model during the execution of the processor, greatly facilitating parametric analyses and sensitivity studies.

4.9.1 Introduction and Importance

By themselves, registers (as described in Section 2.8) allow a user to make sweeping changes in a model not only *between* runs, but also *during* a single run. Such a capability (of making “dynamic” or on-the-fly changes to a model) enables an analyst to easily perform parametric variations within a single run with minimal user logic. Sizing and sensitivity studies are very important, yet are often neglected because they are time-consuming, especially when performed in a consistent manner. Sensitivity studies are especially needed since they help the analyst not only better understand the model, but more importantly the design itself.

These features can also be used to create complex interrelationships between inputs and also between inputs and outputs, simplifying modeling tasks and reducing user logic, and making a model much more variable.

The processor “remembers” via an internal database every place that a register or processor variable was used in the definition of a model, and knows how to propagate changes to registers and variables throughout the model *while the processor solution is proceeding*. By default, the propagation of updates happens automatically and frequently, but the user can optionally assume extensive control of the database and over the updating of parameters.

For example, consider the bar problem presented in Section 1.5.2, the purpose of the model being finding the time (to the nearest minute) at which the middle node exceeds 200 degrees. Assume that the user wanted to get a plot of this event duration as a function of the bar’s specific heat, holding other properties constant. The user might assemble such a plot by making repeated runs, with each run producing a single point on the plot. Or, the user could write logic (perhaps a Fortran DO loop) inside of OPERATIONS to run through a series of values. However, in the latter case the user must then add a nested loop to change the capacitance values for the 3 nodes. This additional logic is not necessary if the registers defining the nodes are adjusted dynamically.

To perform a parametric variation of the specific heat (C_p) of the bar sample problem within one run, a DO loop is added to OPERATIONS as shown in Table 4-6. The calls to SVPART and RESPAR save and retrieve, respectfully, the initial conditions such that each transient starts from

the same point (Section 4.7.1). Logic within TRANSIENT automatically *updates the entire model on the basis of the current value of the registers*. In this problem, only CP was changed, and that value was only used in the definition of the capacitance of three nodes.

Table 4-6 Three Node Bar Model Using Dynamic Registers

```
HEADER OPTIONS DATA
TITLE HEATED BAR SAMPLE PROBLEM, WITH REGISTERS
       OUTPUT  = BAR.OUT
       MODEL   = TEST
 HEADER REGISTER DATA
     DENS    = 0.3
     CP      = 0.2
     CON     = 0.5/12.0
     EMIS    = 0.1
     THICK   = 0.1
     WIDE    = 1.0
     LONG    = 3.0
     AREA    = THICK*WIDE
     VOLUME  = AREA*LONG
   HR2MIN  = 60.0
   RESOL   = 3.0
 HEADER NODE DATA, SUB1
   GEN 10,3,5,70.0,(DENS*CP*VOLUME)/RESOL
   -99, -460., 0.0
 HEADER CONDUCTOR DATA, SUB1
   GEN 1015,2,505,10,5,15,5,CON*AREA/(LONG/RESOL)
   -2099, 20, 99, EMIS*AREA*sbcon/(HR2MIN*144.0
 HEADER CONTROL DATA, GLOBAL
      TIMEND = 1000.0, OUTPUT = 1.0
 HEADER SOURCE DATA, SUB1
      10, 10.0/(btuhrwat*HR2MIN)
 HEADER OPERATIONS
 BUILD TEST, SUB1
       CALL SVPART('-R',NTEST)       $ SAVE I.C.
       DO 100 CP = 0.1, 1.0, 0.1     $ TEST RANGE
              CALL RESPAR(NTEST)     $ GET I.C.
   100         CALL TRANSIENT
 HEADER OUTPUT CALLS, SUB1
       IF(T15.GE.200.0) THEN
           CALL TPRINT('SUB1')
           TIMEND = TIMEN
       ENDIF
 END OF DATA
```

While such logic may be trivial in this sample problem, it can become very complex in the more typical 1000 to 10,000 node problems. For example, perhaps an uncertainty such as a bond joint conductance is being evaluated, and many such joints exist in the system. If the areal conductance

of such a bonded joint had been defined as a register named “Hbond” and that register had been used throughout the model wherever that type of joint was found, then the following logic is all that would be needed to adjust* the entire model at any point in the solution:

```
Hbond = newValue
```

Analogously, the user might be developing their own heat transfer correlation (or correlating a heat transfer coefficient to test data).

These examples illustrate the importance of having automated changes made in a manner consistent with how the user originally defined the problem: in terms of the basic dimensions and properties.

Registers can be used in the definition of nodes, conductors, source terms, fluid system variables, and even array data. Almost every data value in SINDA/FLUINT can be defined as an algebraic expression that may include a register and/or a reference to a processor variable such as “smn.T22” or “loopct” (and therefore will be variable if the value of that register or processor variable changes). In fact, *processor variables themselves might be defined on the basis of registers and other processor variables, enabling a complete spreadsheet-like functionality underlying the entire SINDA/FLUINT program*. As long as users are consistent in the definition of the model, then they need not remember where in the model those definitions were used, nor how to access and change them in logic.

The bar example presented in this subsection is extremely simplified. Actually, the user can choose to exert extensive control not only over the update process, but also over the registers and the places in which they were used. For example, the user can request that only thermal data be updated, or only node data in a certain submodel, or only the data within a specific array, etc. The user can also disconnect and reconnect all or part of the model from being automatically updated if desired. Users can even check to see if a particular value is defined via an expression containing a register or processor variable (vs. a hard-wired value or an expression that has no registers and hence evaluates to a hard-wired value). Registers themselves can be changed by providing a new expression instead of a fixed value, and registers can be updated independently from the model if desired.

* Actually, unless the user specifically calls UPREG after the update, the adjustment of the model is deferred until the next solution routine or until the end of the current logic block.

4.9.2 Nomenclature

Defining some nomenclature is helpful in the explanations that follow:

Processor Variable
: Any translatable SINDA/FLUINT network or control parameter (Section 4.1.4), such as “smn.T22” or “tlen403” or “drlxcc” or “fr#this.”

Formula . . . . . An expression containing at least one register or processor variable, and perhaps consisting of a single register, such as ‘fred’ or ‘2.0*cond*thick*width/length’ or ‘2.0*pi*cond*length/ln(Do/Di).’ If an expression does not contain a register nor a processor variable, then its value is constant unless updated via other SINDA/FLUINT options.

Parameter . . . . a SINDA/FLUINT variable defined using a formula, and therefore potentially subject to being updated if a change is made to one or more of the registers or processor variables in the formula. A processor variable might itself be defined using a *formula* and therefore becomes a *parameter* as well, providing complex interrelationships that are resolved in spreadsheet fashion.

4.9.3 Affected Variables

Almost every variable in SINDA/FLUINT can be defined by a formula and are therefore subject to be modified automatically or as requested by the user. It is therefore simpler to list the variables that *can’t* be modified in this manner, even if they were defined using formulas. Variables that are *not* subject to automatic updating during processor execution include:

1. Variables that can’t be defined by an expression in the first place. These are detailed in Section 2.8, and include items such as integer element identifiers (node, macro, tie, etc.), increments on element identifiers in macrocommands, nonnumeric data such as STAT flags (which are character data), etc. The preprocessor will produce an error if it encounters such an illegal use of an expression.

2. Fluid property data (FPROP blocks). While formulas containing registers *can* be used to define these, once processor execution starts they remain invariant. Hence, processor variables cannot be used in those formulas.

3. Thermal unit conversion factors: fac, tcon. These values may be defined by an expression, but they will remain constant during processor execution. This does not imply that values to which they are applied (boundary node T, diffusion node C, conductor G) can’t be variable, just that the conversion factors themselves are not variable once the processor execution starts. Hence, processor variables cannot be used in those formulas either.

4. Fluid unit conversion factors: TLFACT, PLFACT, etc. Once again, these values may be defined by an expression, but if they will remain constant during processor execution and so cannot contain references to processor variables. This does not imply that values to which they are applied (plenum TL and PL, MFRSET SMFR) can't be variable, just that the conversion factors themselves are not variable.

5. Archaic (undocumented) options such as the CAL feature are assumed to contain only numeric constants or constant expressions. In other words, those features are only supported for backwards compatibility.

And finally and most importantly:

6. ***Variables that are normally output to the user, and not input to the program, will not be updated even if their* initial *values have been specified using formulas. For this reason, any formulas used to define such variables cannot contain references to processor variables.***

For example, the *initial* temperature T of diffusion and arithmetic nodes may be defined by an expression or even a register-containing formula, but during processor solution these definitions are "forgotten" since they are assumed to only be true for initialization purposes, even if these temperatures are later held using DRPMOD. In other words, automatic updates will not affect the T of diffusion nodes and arithmetic nodes even if the initial T was defined using registers. The use of formulas in the T of a boundary node, however, *will* become a parameter since it is not truly an initial condition so much as a boundary condition. In other words, it will be subject to automatic updates because it is not a normal output variable--it is not normally updated by SINDA/FLUINT solution routines.

Actually, there are few such variables, which include the thermodynamic state variables of tanks and junctions (TL, PL, XL, XGx, XFx -- even if these variables are later held using HLDLMP), and path flow rates (FR, but not the input variable SMFR of an MFRSET connector). Thermodynamic state variables of plena will become parameters, but see the cautions regarding such usage in Section 4.9.3.3.

Control constants such as ABSZRO, SIGMA, PATMOS, and TIMEO may be initialized using formulas, but are not updated automatically and hence cannot reference processor variables.

Tank volumes (VOL) represent a special case since they are normally input variables, but in some case (specifically, if VDOT, COMP, or ifaces are used) can become output variables. The processor monitors this variable and will issue a one-time caution if the possibility of conflict is detected.

4.9.3.1 Special SOURCE DATA Rules

Because of their cumulative nature, special rules apply to formulas used in source data options. For example, 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.

Also, a restriction exists with respect to multiple source declarations for the same node. No more than one SIT, DIT, CYC, TVS, TVD, or PER option can be used on any single node if those duplicate options *both* contain formulas. (This restriction is likely to never be experienced.)

4.9.3.2 Derived Expressions

To the greatest extent possible to encourage the use of registers, SINDA/FLUINT attempts to preserve formulas. For example, if a LINE macro were used to define a duct, and only the duct diameter DHS were given as a formula (say, '*diam*') and no AF, AFI, AFJ, or AFT values were provided nor defaults values defined previously, then the program will assign the formula "pi*diam^2/4" to AF and AFT, and would also assign tank volumes (if any) values based on formulas instead of hard-wired numbers. This scheme is intended to enable the user to later change *diam* in the processor, and have all related variables adjusted automatically.

The user should therefore be careful regarding default values, which may cause SINDA/FLUINT to use them instead of derived expressions. This might cause later changes to be inconsistent. For example, if *diam* had been used to define DHS in the above case but AF had been defaulted to "3.0E-4" in a prior PA DEF block, then the code would use a variable diameter but a fixed flow area, assuming that is what the user intended. Expressions can be used in default blocks (PA DEF, LU DEF) as well, which helps avoid this problem. Or, the AF could be defined in terms of the DH itself (such as "pi/4*DH#this^2") to preserve the relationship between the two inputs.

To preserve expressions and to create derived expressions internally, SINDA/FLUINT must create potentially lengthy derived expressions, especially if formula were used in the definitions of increments. Since expressions are limited to 132 characters total, the total length limit may be exceeded and an error will be produced. In such cases, the user will need to add new registers to simplify the derived expressions. For example, the user might create a new register called *area* and set it to "pi*diam^2/4" which results in simpler (or at least shorter) derived expressions.

4.9.3.3 Formula Defining Plenum States

If the TL, PL, or XL of a FLUINT plenum is specified using a formula, then the code will use internal calls to CHGLMP to adjust the fluid thermodynamic state if registers or processor variables dictate a change to TL, PL, or XL.

Unfortunately, it is not possible to know or control the order of these calls. For example, consider a case with pure fluids where TL=*paramT* and PL!=*paramP* yet XL=0.5 (requiring a saturated state). If both *paramT* and *paramP* were changed simultaneously, it is possible the PL is updated before TL instead of the reverse, and thus the command to update PL was superseded in apparent contradiction of the "PL!" rule. Similarly, if XL were changed parametrically, the TL value might be overwritten.

Also, the code does not internally call CHGLMP_MIX. This means that simultaneous adjustments to more than one species' XG or XF mass fraction may not be possible.

In other words, due to the complexities of thermodynamic states, unless single-phase fluids are involved or mixtures in which only a single species is adjusted at a time, the user should consider avoiding the formula-based method of defining updates to plena states in favor of explicit calls to CHGLMP and/or CHGLMP_MIX in OPERATIONS or in FLOGIC 0.

To continue the above example, if the plenum in question was #100 in submodel "subm," then the following would be used to set the plenum's state, rather than FLOW DATA based expressions:

```
CALL CHGLMP('subm',100,'TL',paramT,'XL')
CALL CHGLMP('subm',100,'PL',paramP,'XL')
```

4.9.4 Changing Registers in the Processor (e.g., in Logic Blocks)

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 a replacement statement or via a call to a subroutine:

```
regname  = 10.0E-4                    $ replacement
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. A routine called CHGREG (Section 7.12) is therefore available in the more rare instance where the user needs to change the underlying expression defining a register, and not just the current value.

Changing the value (or expression) of a register does not by itself cause the immediate update of any other variable (including other registers). Rather, the program automatically invokes the spreadsheet-like propagation of any changes that have been made *at discrete intervals* after user logic blocks have been executed, as described next.

4.9.5 Automatic Spreadsheet Propagation

By default, the program will automatically propagate changes in registers and/or processor variables at discrete intervals during each steady state iteration or transient time step. This update is frequent and global, but it occurs *after* user logic blocks have been executed, and not *during* their execution. The automatic propagation of updated values *typically* happens after FLOGIC 1 and also after VARIABLES 1 such that the network is "fresh" before a solution step occurs.

Internally, the code periodically calls UPREG, a routine that propagates changes to registers and processor variables to any parameter defined by a formula (including registers and processor variables themselves). The user can elect to perform additional updates using UPREG or subset versions that limit the extent of the updates (Section 7.12, summarized below in Section 4.9.6). Thus, the user can elect to propagate changes within a logic block as well. The user can also elect to disconnect parameters from the update process altogether.

In fact, the user can turn off automatic updates (internal calls to UPREG, except those in the Solver or the Reliability Engineering module) using the NOUPDATE option in OPTIONS DATA (Section 2.7.2), and thereby assume complete control of the process, as described next.

4.9.6 Assuming Control of the Update Process

Neither resetting the value of a register as shown above nor calling CHGREG causes the immediate update of either registers or the parameters defined on the basis of registers. To update only the registers (but not the parameters defined using registers), use the CALREG (Section 7.12) routine. To update everything, call UPREG (Section 7.12).

By default, UPREG is called internally by the program at several points to make sure the interrelationships are "fresh" before a network solution proceeds. The user can augment this automatic update with his or her own update commands at any time, can control updates by disconnecting parameters, and if fact can turn off all automatic updates (except those performed by the Solver and the Reliability Engineering module) using the NOUPDATE option in OPTIONS DATA (Section 2.7.2).

In other words, *the propagation of changes from registers to the parameters upon which they are based is only performed at discrete intervals, or at the request of the user,* who has complete control over when to update and what to update.

For most applications, the default update system is adequate. The following options can be quite customizeable, but are only needed for very specialized modeling needs.

The simplest routine, UPREG, updates all registers and all parameters, and is often all that is needed to augment or replace the default update system. Section 7.12.1 describes other routines that can be used to selectively update parts of the model, including:

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

For example, UPREGT and UPREGF can be used if needed to comply with the guideline that thermal parameters should be updated only from VARIABLES blocks and fluid parameters should only be updated from FLOGIC blocks. Underlying registers and expressions will be updated, but only the subset of parameters defined by these routines will actually be changed during the spreadsheet iterations.

Actually, all 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.

There is rarely any problem or penalty, however, from calling these routines too often or from duplicating any calls. This lack of a penalty for repeated calls is due to the checks performed by the UPREG routines to avoid unnecessary effort if little has changed. However, these same checks may prevent the update of parameters that have been changed independently of their defining formulas, as noted in the caution below.

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. Such a discrepancy can arise if the user has overwritten the parameters since the last update (perhaps due to a call to RESPAR or RESTAR, as explained in Section 4.9.7).* See Section 4.9.6.3 for means of forcing the updates.

4.9.6.1 Sequence and Location of Updates

Updating parameters via dynamic registers is no different from having made the changes to many variables “manually” in user logic. The same restrictions and guidance apply. However, since it is much easier to make sweeping changes to models when using registers and formulas, and in a manner that “hides” the extent of the changes made, the user should perhaps exercise a little caution.

First, it should be realized that changes made through updating parameters may overwrite changes the user has recently made, and that changes the user makes afterwards will similarly overwrite changes made via the automatic process.

Also, depending on where the call to the update routines 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 formulas, 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.*

When lump volumes are updated, an internal call to CHGVOL is made. When plenum thermodynamic variables (PL, TL, XGx, etc.) are updated, internal calls to CHGLMP will be made.

When integer inputs such as IPDC or ITERXT are governed by formulas, the resulting real values are rounded to the nearest integer.

Just as with any other change to SINDA/FLUINT modeling parameters in logic blocks, ***no checks are made to detect illegal or out of range inputs.*** Although some program options can trap such problems later in the execution, such checks are the user’s responsibility.

Caution: Infinite Loops and Circular References: There are few restrictions on the interrelationships that can be expressed using registers. Registers can be defined on the basis of processor variables and other registers. Processor variables themselves might be defined on the basis of registers and other processor variables. It is quite possible to set up a circular reference that either cannot

be resolved by the program in an iterative fashion, or that causes variables to progress towards positive or negative infinity. The program attempts to trap such instances, but the user should be careful to avoid circular references.

For example, if the user defined a register to be “fred = mymod.C3” and then defined the capacitance of node 3 in submodel “mymod” to be “fred,” a circular reference has been created. Since the *preprocessor* will assign a random number to “mymod.C3,” this number (between 1.0 and 2.0) will be passed to the processor and will become the capacitance of node 3 perpetually, the program unable to diagnose a possible problem. If on the other hand the user had defined the capacitance of node 3 to be “2*fred,” then both fred and mymod.C3 would have spiralled to positive infinity, hopefully causing a convergence failure or graceful overflow detection by the program. However, not all underflows and overflows or divides-by-zero can be trapped by the program without introducing excessive overhead in normal problems.

4.9.6.2 Disconnecting and Reconnecting Parameters

As part of the user’s control over the update process, some or all of the parameters may be disconnected from the update sequence, and hence are no longer “parameters” in that their value will not change when future updates are performed. These disconnected parameters can later be reconnected and updated automatically if desired.

Section 7.12 also describes routines that can be used to selectively disconnect part of the model, including:

DEREG. . . . . disconnects all automatic updates
DEREG1. . . . . disconnects only a single specified parameter
DEREGC . . . . disconnects a custom selected set of one or more parameters

A similar family of routines (REREG, REREGC, etc.) is available to reconnect parameters to the update process. A special routine, ISREG1, is available to check to see if any one parameter is currently connected.

4.9.6.3 Discrepancies and Forcing Updates

A call to one of the UPREG family of routines, including those called automatically by the program, 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.* This discrepancy can arise if the user has overwritten the registers or parameters since the last update (perhaps due to a call to RESPAR or RESTAR, as explained in Section 4.9.7).

For example, if the temperature of boundary node #99 were defined using the register *sink*, then the temperature of that node and *sink* will initially have the same value, say 300 degrees. If the user then resets the temperature of the node in logic via an assignment statement (e.g., “T99 = 200.0”), ignoring the use of the register to define the temperature of that node, then a subsequent automatic updates (or user calls to one of the UPREG family of routines) will *not* change the temperature of

the node because the value of *sink* itself has not changed. This discrepancy could also have resulted if the user called RESPAR or RESTAR and the retrieved data included nodal temperatures but did not include registers.

To force the update process, the user can use a more expensive variation of UPREG called the FORCER:

```
CALL FORCER
```

This updates all formulas containing either registers or processor variables whether or not changes have been made *and whether or not the parameters have been disconnected* in order to assure consistency. ***It also reconnects all parameters*** via an internal call to REREG, such that any disconnecting done with the DEREG family of routines will need to be repeated. FORCER can also have adverse effects on plenum states since the order in which expressions for PL (or PL!), TL, and XL are updated is pseudo-random. If FORCER is invoked, check plenum states afterwards.

4.9.7 RESPAR, RESTAR, and Registers

Unlike named user data (USER DATA, GLOBAL), register values *are* stored in save and restart data sets when 'ALL' is chosen in the SAVE, RESAVE, SVPART, and SAVEDB routines, and anytime the SAVPAR routine is used. Furthermore, any expressions defining those registers are stored as well. Use 'R' in the argument list of the SAVE, RESAVE, or SVPART routines to designate registers separately (just as 'T' designates temperatures, 'L' designates lump values, etc.). See Section 7.4 for descriptions of these output routines.

When registers are restarted from a previous run (RESTAR, Section 7.5.4) or retrieved from a previous point in the current run (RESPAR, Section 7.5.3), then the new values will overwrite the old ones (and the old expressions) if there is has been a change. These replacements themselves do not force an update of the registers nor the parameters, any more than the assignment statements or subroutine calls described previously in Section 4.9.6. Restoring data values (e.g., nodal Ts) without restoring registers does not therefore overwrite the data values even if the registers defining them are now out of sync per the rules described above (Section 4.9.6.3). Restoring registers but not the corresponding data values similarly does not trigger an update: a discrepancy is allowed since it is presumed to be purposeful. *If such a discrepancy between data and the defining registers is not intentional, save both the data and the registers (using, for example, SAVPAR or the 'ALL' option).*

Caution: Values of parameters (variables defined by formulas containing registers or processor variables) may be overwritten when calling RESTAR or RESPAR or RESTDB, or at least an inconsistency can arise (and is assumed purposeful) if the registers were not also saved and restored. Restoring data but not registers may overwrite previous updates caused by changes to registers or other processor variables. Furthermore, subsequent internal updates (or user calls to the UPREG family of routines, Section 7.12.1) will have no effect unless the registers *themselves* have changed. The user might therefore consider a call to FORCER (Section 4.9.6.3) to assure a consistent update. Usually, *the exclusion of registers and expressions from the saved set should be purposeful, and should be considered an exception to normal usage.*[*]

* In prior versions, the exclusion of registers was recommended in some circumstances that no longer exist.

Guidance: Values of registers may be overwritten when calling RESTAR or RESPAR or RESTDB. For this reason, when RESTAR or RESPAR is invoked from within a Solver, parametric sweep, design space scanning, or Reliability Engineering run (e.g., DSAMPLE), then the code automatically updates any expressions containing key registers after restoring the data.

4.9.8 Common Misconceptions: Fortran vs. Expressions

Some confusion exists because of the misunderstanding between data blocks (which are only taken into account in the once-through preprocessor), and logic blocks (which are executed one or more times in the processor). Part of the confusion is that many data blocks have Fortran-like input structures, including REGISTER DATA. With the advent of processor variables in expressions (as of Version 4.1), data blocks can appear even more to resemble logic blocks when then include references to items such as “loopct” and “subm.g(303).” Other confusion exists because of the presence of two types of calculations: those in Fortran in logic blocks, and those contained in expressions.

For example, defining “fred = 2.0*wilma” **in REGISTER DATA** sets the defining expression for *fred* and initializes its value in the preprocessor. The *relationship* “fred = 2.0*wilma” is remembered by the program: this is not just a one-time Fortran *assignment* of one value to another.

Restating “fred = 2.0*wilma” **within a logic block**, however, is a Fortran assignment and is *not* an expression definition. Rather, the *current value* of “2.0*wilma” (perhaps “2.0” if *wilma* is currently equal to 1.0) will overwrite the *current value* of fred *and* its expression--as if fred had been defined as “fred = 2.0.” To change the underlying expression for a register (and not just its current value), using the CHGREG routine.

Other areas of confusion exist between expressions and Fortran logic blocks. For example, built-in functions (min, max, ln, floor, etc.) and constants (pi, sbcon etc.) exist only within register expressions. They cannot be referred to in Fortran logic, or in the case of functions, are not exactly the same as the Fortran intrinsic function of the same name. For example, “^” means exponentiation only in an expression, but not in Fortran. Also, “min” and “max” can have only two arguments in expressions, whereas in Fortran the analogous functions can have many arguments.

For example, the following line placed in a logic block:

```
AF(10)  = 0.25*pi*DH(10)**2
```

would either result in a DEBUG option complaint that “pi” is undefined, or if the DEBUG option is off (which is much more dangerous), then AF10 (equivalent to “AF(10)”) would simply be set to zero ***since “pi” only exists in expressions, not in logic***. To access such convenient variables in logic, set (for example) a new register such as “pie = pi” and then in logic:

```
AF(10)  = 0.25*pie*DH(10)**2
```

Still, since the above statement is in Fortran and executed in logic blocks, no permanent relationship has been established between the value of DH10 and AF10. If instead a formula had been used to define the AF of tube or STUBE #10 within FLOW DATA:

```
PA CONN, 10, 11, 12, DEV= STUBE
        AF      = 0.25*pi*DH(10)**2
```

Then the value of AF10 would be linked to changes in the hydraulic diameter of that connector because the program will remember the underlying formula, and will update the value of AF10 periodically during the solution. Alternatively, the above input could have also been specified as:

```
PA CONN, 10, 11, 12, DEV= STUBE
        AF      = 0.25*pi*DH(#this)**2
```

The use of pound sign ("#") operators such as "#this" shown above is restricted to input blocks and cannot be used in logic blocks, since the preprocessor is unable to translate "#this" into a real Fortran array location not knowing to which network element "#this" refers.