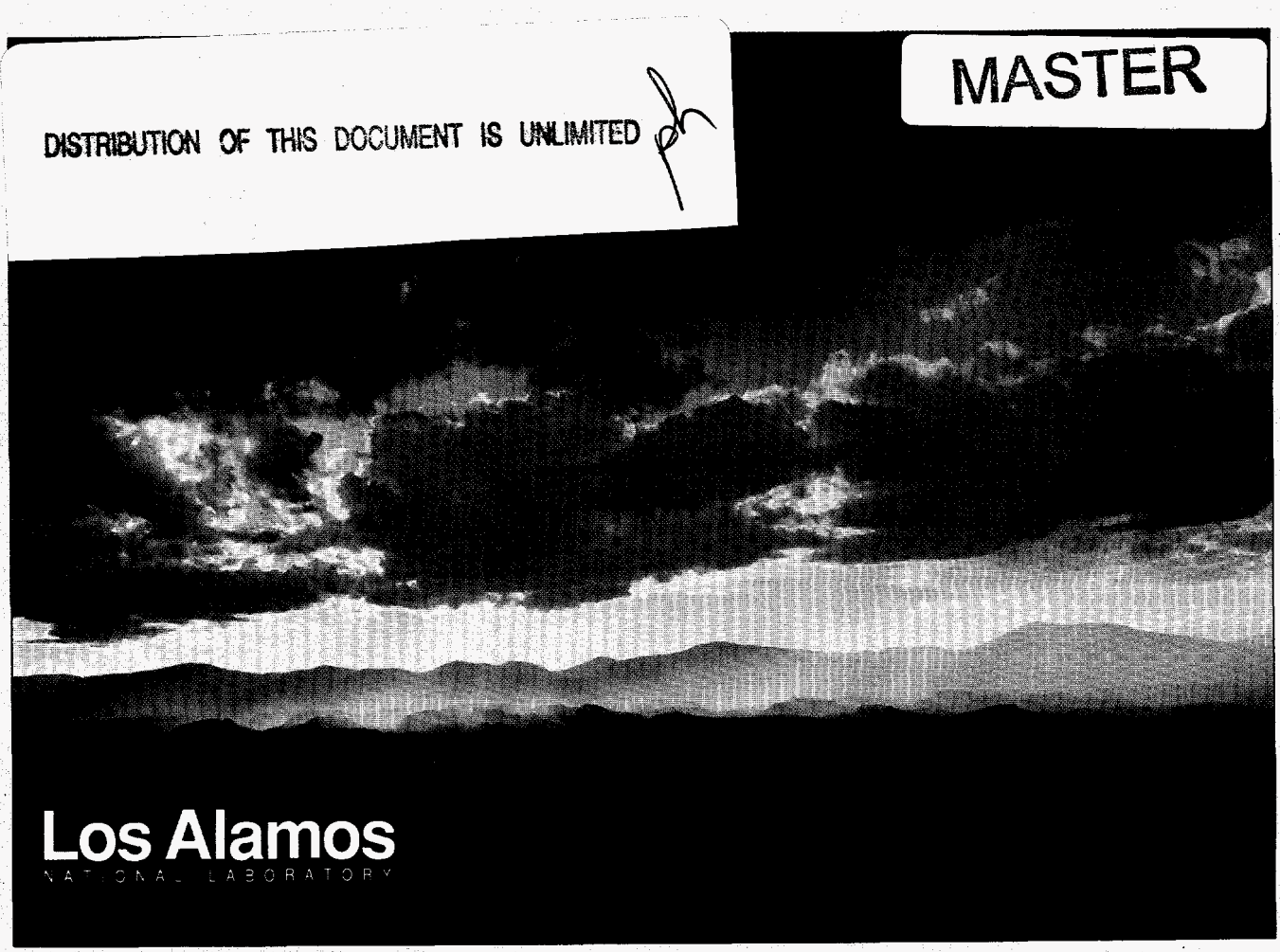
LA-UR-98-2717
May 1998

APPLICATION PROGRAMMING INTERFACE DOCUMENT FOR THE MODERNIZED TRANSIENT REACTOR ANALYSIS CODE (TRAC-M)

by

J. Mahaffy
Pennsylvania State University

B. E. Boyack and R. G. Steinke
Los Alamos National Laboratory



Photograph: by Chris J. Lindberg

DISCLAIMER

APPLICATION PROGRAMMING INTERFACE DOCUMENT FOR THE MODERNIZED TRANSIENT REACTOR ANALYSIS CODE (TRAC-M)

by

J. Mahaffy
Pennsylvania State University

and

B. E. Boyack and R. G. Steinke
Nuclear Systems Design and Analysis
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

The objective of this document is to ease the task of adding new system components to the Transient Reactor Analysis Code (TRAC) or altering old ones. Sufficient information is provided to permit replacement or modification of physical models and correlations. Within TRAC, information is passed at two levels. At the upper level, information is passed by system-wide and component-specific data modules at and above the level of "component" subroutines. At the lower level, information is passed through a combination of module-based data structures and argument lists. This document describes the basic mechanics involved in the flow of information within the code. The discussion of interfaces in the body of this document has been kept to a general level to highlight key considerations. The appendices cover instructions for obtaining a detailed list of variables used to communicate in each subprogram, definitions and locations of key variables, and proposed improvements to intercomponent interfaces that are not available in the first level of code modernization.

I. INTRODUCTION

The objective of this document is to ease the task of adding new system components to the Transient Reactor Analysis Code (TRAC) or altering old ones. In addition, sufficient information is provided to permit replacement or modification of physical models and correlations. The description of interfaces associated with system components requires some repetition of information provided in the Software Design Information Document (SDID) for TRAC-M data structures[1] and in the Code Architecture and Computational Flow document to be prepared at the completion of the database restructuring effort. However, only the general features of the data structure and code architecture will be

described here. These two documents also should be read carefully before attempting to create or modify a TRAC system component.

Within TRAC, information is passed via system-wide and component-specific data modules at and above the level of "component" subroutines, such as Rpipe, Repipe, Ipipe, Pipe1, Pipe2, Pipe3, Dpipe, Xtvpipe and Wpipe. Examples of system level data modules are GlobalDatM, GlobalPntM, and GlobalDimM. Examples of component-specific data modules are PipePtrM, PipeVltM, PlenPtrM, and PlenVltM. Below these subroutines, information is passed through a combination of module-based data structures and argument lists. One goal of the initial TRAC modernization was to minimize changes to subprogram argument lists. As a result, the argument lists and calling trees below the level of the component routines are very similar to those in TRAC-PF1/MOD2. The argument lists of the component subroutines themselves have been eliminated to avoid conflicts with the revised data structure.

The basic logic behind information passing in TRAC has been to pass state variable arrays for the fluid or structure material and indices bounding array sections through argument lists. Scalar-state variables passing boundary information (particularly at the Tee internal junction) were passed through special modules. Variables that operate as constants in state equations, correlations, etc., also have been passed through common blocks (now modules). Unfortunately, this logic was not followed consistently over 20 years of development, and exceptions exist. No consistent pattern has developed for scalar flags and important array indices. These can be found passing through both argument lists and modules.

The most complex and frequently modified interfaces exist in the component-related subroutines. Component subroutines are provided for each of nine key stages of TRAC execution:

1. Input of initial component data (e.g., Rpipe);
2. Input of restart information for a component (e.g., Repipe);
3. Initialization of component-dependent variables (e.g., Ipipe);
4. Stabilizer momentum equation solution, evaluation of various old-time quantities, and other bookkeeping at the beginning of each timestep (e.g., Pipe1);
5. Iterative solution of basic flow equations for each timestep (e.g., Pipe2);
6. Solution of stabilizer mass and energy equations, solution of the conduction equations, and other computations to complete each timestep (e.g., Pipe3);
7. Output of data to the restart dump file (e.g., Dpipe);
8. Output of data to the XTV graphics files (e.g., Xtvpipe);
9. Output of data to the ASCII detailed edit file (e.g., Wpipe).

Similar component subroutines for each of the nine key stages of TRAC execution also exist for the other TRAC components, e.g., Tee, Fill, Break, Pump, Prizer, Valve, and Plenum.

A schematic illustrating the TRAC's top-level program flow, with emphasis on the computational solution of the flow equations, is presented in Fig. 1. The program construct is shown for advancing the solution one timestep, beginning with subroutine Trans, which begins with the stabilizer step for the equation of motion (subroutine Prep), basic equations solution for all equations (subroutine Outer), and stabilizer step for mass and energy equations (subroutine Post).

Within a given timestep, subroutine Prep calls all component subroutines twice. Subroutine Outer calls all component subroutines twice, and subroutine Post calls all component subroutines three times. In each case (Prep, Outer, and Post), two passes provide the setup and solution of a set of equations. However, the two passes in Outer are contained within a Newton iteration loop. Post adds a third pass to calculate some final end of timestep values for mass flows and mean cell densities. The outer loops in Prep, Outer, and Post, which take more than one pass through all components, are indexed by the variable "ibks." This variable takes on values of one and two in Prep, values of zero and one in Outer, and one, two, and three in Post. Component subroutines in these three stages must use the module OneDDat to obtain the value of ibks to follow the flow of the calculation properly.

Each of the subroutines shown in Fig. 1, Init, Steady, Trans, Prep, Outer, and Post, access lower-level subroutines. More detailed calling trees are presented in App. A, beginning with each of these subroutines, as well as the calling tree for the driver routine, TRAC (Fig. A.1).

The next five sections cover some of the basic mechanics involved in the flow of information within the code. A discussion is provided for one-dimensional (1D) component subroutines in Sec. II, for the three-dimensional (3D) Vessels in Sec. III, for the Plenum component in Sec. IV, and for the heat structures in Sec. V. The basic pattern for information flow seen in Pipe is repeated in all basic components and should be understood before trying to work with the more complicated components. Section VI covers noncomponent models, such as the control-system, constrained steady-state, hydraulic-path steady-state initialization, and radiation models. The discussion of interfaces in the body of this document has been kept at a general level to highlight key considerations. For instructions on obtaining a detailed list of variables used to communicate in each subprogram, consult App. B. For definitions and locations of key variables, consult App. C. Appendix D covers proposed improvements to intercomponent interfaces that are not available in the first level of code modernization.

The general code architecture provides the ability to connect a wide range of component and physical models to the existing code. However, many such connections are not desirable or should be approached with extreme caution. Section VII presents known limitations on the range of useful connections.

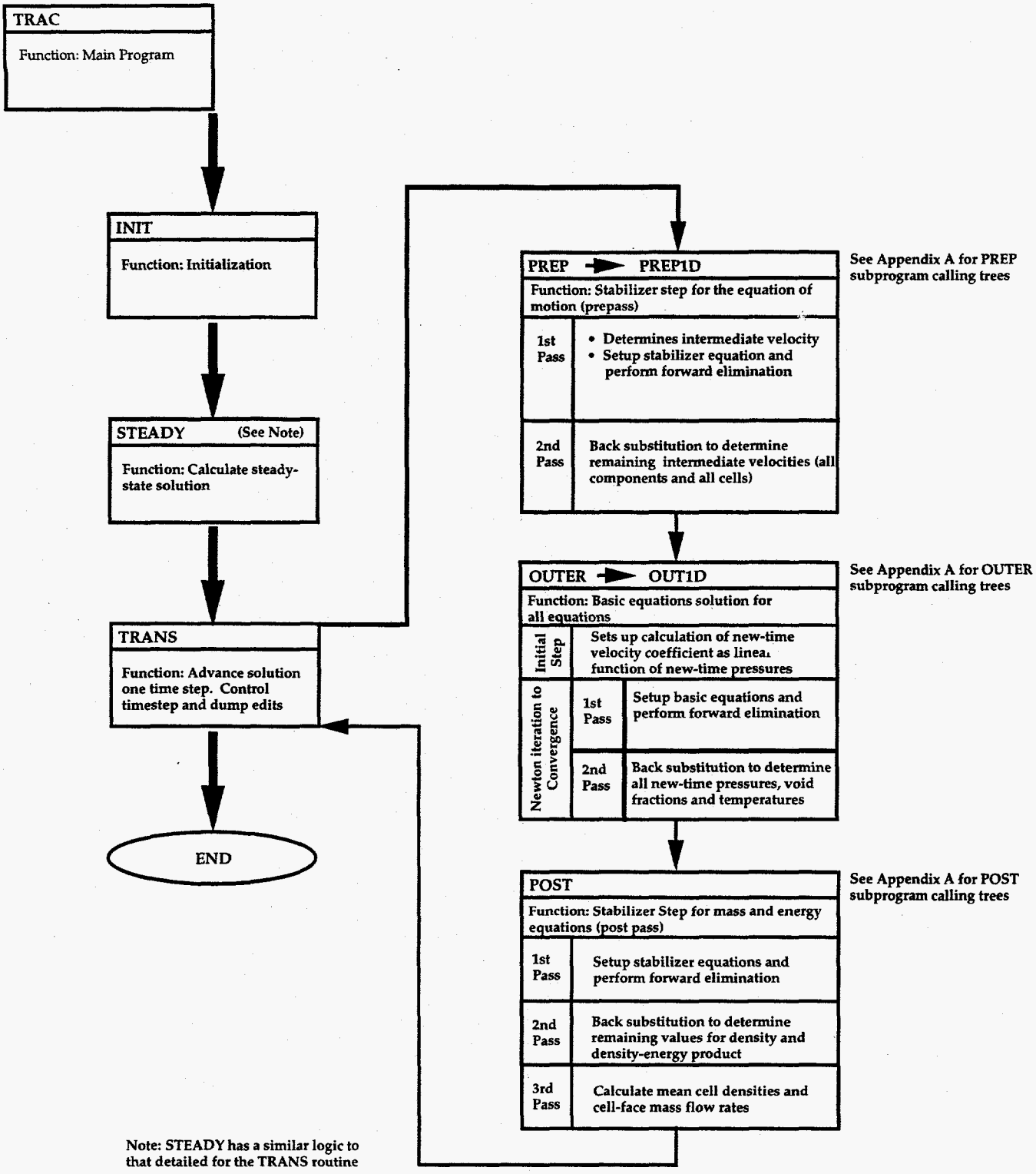


Fig. 1. TRAC computational flow.

II. ONE-DIMENSIONAL COMPONENTS

The interface to a 1D component follows one basic pattern that is best seen in Pipe, with minor variations for boundary conditions and Tee-type components. Tees involve duplication of Pipe coding and special internal generation of boundary conditions at the internal Tee junction. Boundary conditions (Fill and Break) generate junction boundary information on the same cycles as Pipe, but perform relatively few other operations.

II.1. Pipe

The data interfaces for the Pipe component are described here for each of the nine computational stages outlined in Sec. I. The pattern described here for the information passing is followed very closely in the other 1D components and, as will be seen in later sections, is followed to a large degree by the Plenum and Vessel.

II.1.1. Basic Input for Pipe

The basic calling tree associated with input is summarized in Fig. A.2.a of App. A. Throughout this document, the Pipe component is used as the primary vehicle for discussing the lower-level subroutines in the calling tree. Input of initial Pipe data is driven by Rpipe, which is called by the subroutine Rdcomp. Creation of a new component similar to Pipe would require the addition of a Call in Rdcomp to process that component's input. The component's type (Pipe), component number, ID number, and descriptive title are obtained in subroutine Input (Fig. A.1) before it calls Rdcomp. This information is passed to Rpipe via the module for the fixed length table (FltM) as the variables type, num, id, and title. Rpipe obtains values of other scalar variables for the component from the ASCII input file, using the subroutines Readr and Readi (Fig. A.2.b). These two subroutines also echo this input to the detailed output file (trcout). Readr and Readi should be used for input of any scalar data for a component to maintain a consistent interface with the input file and its reflection to the output.

Rpipe calls the subroutine Rcomp to obtain array information on the geometry and initial state of the fluid for all cells (dx, vol, fa, fric, fric, gravp, elev, hd, hdht, nff, lccfl, alpn, vln, vvn, tln, tvn, pn, and pan, and where appropriate, wfmfl, wfmfv, qppp, matid, twn, concn, and sn). Rcomp in turn uses the subroutine Loadn to bring array data from the input and the subroutines Warray and Wiarn, respectively, to echo real and integer array values to the output. Loadn, Warray, and Wiarn also are used directly by Rpipe to obtain additional array information. These subroutines (Rcomp, Loadn, Warray, and Wiarn) are the standard interfaces for reading and echoing array values from the input file and should be used for this purpose in any new component.

When input errors are detected in subroutines at or below subroutine Input, the input line is echoed to the standard detailed ASCII output file and a warning message is printed to that file, the message file, and the terminal. The error message is produced by a call to the subroutine Error, with the first argument set to "2" to indicate a warning rather than a fatal message. By convention, the message (passed as the second argument) begins with the name of the subroutine processing the input line, bounded by

single asterisks (e.g., "*rpipe* inconsistent init & table power"). When additional diagnostic information is necessary, including values of variables, direct Write statements are necessary. Pairing of this information to the messages from Error requires three writes: one to the terminal (unit number in variable "itty" from module Io); one to the standard detailed ASCII output (unit number in variable "iout" from module Io); and one to the message file (unit number in variable "imout" from module Io).

Termination of input processing is flagged at two levels of severity. The lowest-level input routines (Loadn, Readi, Readr, and Nxtcmp) set the value of variable "ioerr" (located in the module Io) to one when an input error is detected. Subroutine Input checks the value of "ioerr" after completion of basic component input (call to Rdcomp) and terminates if it is not zero. The presumption is that input errors are severe enough that it is not worth any processing of the restart file or checking of flow network connectivity. Higher-level routines (Input, Rpipe, Rtee, etc.) flag problems for later termination by setting the variable "jflag" (contained in module Badinput) to one. One exception is subroutine Rcomp, which uses a variable "jflagc" (contained in the common block concck) for the same purpose. The class of errors detected at these levels is presumed to be localized enough to make checking of flow network connectivity profitable. Input will terminate execution before returning if jflag or jflagc is not equal to zero.

Rpipe has one other important but subtle interface that must be replicated in new components. By supplying values to the "jun" array (and incrementing jun's current index "jptr"), Rpipe supplies information to the system necessary to establish the order of the calculation and the structure of the network solution matrix. The "jun" array is scanned by Srtlp (Fig. A.1) after all input is processed, and the order of component processing is placed in the array "iorder". Srtlp also returns other significant information necessary for the details of the network solution procedure (see App. D for details). The lower-level calling trees for other TRAC components are shown in Figs. A.2.b through A.2.d.

II.1.2. Restart Input for Pipe

The basic calling tree associated with restart input is summarized in Fig. A.3 of App. A. Input of restart information for a Pipe component is driven by the subroutine Repipe, which is called by Rdrest. Creation of a new component similar to Pipe would require the addition of Call in Rdrest to process that component's input. Restart input begins with communication of the lists of all system components (iorder) and all components in the ASCII input deck (nbr) to Rdrest via the system-wide data structure (Fig. A.1). The Pipe's fixed length table is read from the restart file by a call from Rdrest to rstFLT. If the Pipe's component number is included in the current list of system components but not in the list of component's read from the ASCII input file, then Repipe is called to complete restart input.

Repipe uses the subroutine Rstvlt to read the Pipe variable-length table from the restart file. It then echoes values of the variable-length table to the standard detailed output file using subroutine Reecho. Standard arrays required for restart of 1D flow (dx, vol, fa, fric,

grav, hd, nff, lccfl, wa, qppp, matid, alpo, alpn, vln, tln, pn, pan, wfmfl, wfmfv, aran, twn, tvn, alvn, chtin, vvn, arvn, arln, arevn, areln, rmvm, rvmf, vmn, bitn, hiv, hil, hig, higo, cifn, rhs, vvt, vlt, gamn, elev, chtan, alven, twan, twen, tcen, and, when appropriate, sn, concn, and qppc) are acquired by a call to Recomp. Those arrays that would normally appear in an echo of input are printed by a call to Wrcomp, which in turn uses the standard low-level routines Warray and Wiarn to write array values to the standard detailed output file. Actual input of values or arrays of values in Repipe or Recomp is accomplished with the subroutine Bfin rather than a direct FORTRAN Read statement because TRAC contains its own buffered I/O routines (Bfaloc, Bfin, and Bfout) for output to binary files. These buffered I/O subroutines should be used with any new component, as should standard routines to echo values to the standard detailed output.

As with Rpipe, values for the "jun" array must be loaded from Repipe to provide information on the pipe's location in the flow network.

The restart and dump need not be the direct responsibility of the component's programmer. App. D contains a proposal for "system services" that would provide for these functions with minimal information from the component's programmer.

II.1.3. Initialization of Pipe-Dependent Variables

The basic calling tree associated with initialization is summarized in Fig. A.4 of App. A. Initialization of Pipe-dependent data is driven by the subroutine Ipipe, which is called by Icomp. Any new component would require the creation of an initialization routine and the addition of an appropriate call to Icomp.

Icomp sets the values of two arrays that provide information to component subroutines. The first, "jseq", is simply a list of all junction numbers used in the current system model. It is used only during the initialization phase, providing a convention to components for storage and use of junction-specific information such as the boundary (bd) array. The second array filled by Icomp (vsi) contains sign convention information for component junctions, aligned one for one with the junction numbers in "jseq". If an element of "vsi" is +1.0, then the velocities in the two adjacent components have the same sign convention (flow at +2.0 m/s at that junction in one component is also +2.0 m/s at that junction in the other component). If an element of vsi is -1.0, then the velocities in the two adjacent components have the opposite sign conventions (flow at +2.0 m/s at that junction in one component is interpreted as -2.0 m/s at that junction in the other component). This array is used throughout the calculation to translate velocity and mass-flow boundary information between components.

Calls to component-specific initialization subroutines are contained in a DO loop that will cycle either two times for normal execution or three times if steady-state (stdyst) options 3 or 4 are selected to perform an initial estimate of steady-state temperature and velocity distributions. The status of the loop counter is stored in the variable iinl (named common block elvkf) and can take on the values of 0, 1, or 2. However, component-specific subroutines (e.g., Ipipe) are called only when iinl has values of 1 or 2. The basic work of component initialization takes place when iinl=1. When iinl=2, Ipipe and

similar routines check the consistency of cell edge quantities at the junction, compute the elevation changes across components, and convert the loss coefficients to TRAC's specific form of friction factors. All components must follow this pattern of processing during the initialization step for the boundary checking process to function.

Ipipe begins by obtaining values for two communications variables contained in the variable-length table (module PipeVlt). Variables "js1" and "js2" contain the index in junction-related arrays (bd, vsi) for the left and right junctions of the pipe. They are used throughout the calculation to index the appropriate elements of the boundary (bd) and velocity sign (vsi) arrays for the pipe end junctions. Values are determined for "js1" and "js2" with the function Jfind.

Ipipe next calls to subroutine Junsol twice (one for each junction) to set isollb and isolrb (in module PipeVlt), indicating the nature of the velocity calculation at the junction. A value of 0 from one of these variables indicates that the velocity is fixed by a Fill boundary condition. A value of 2 indicates that a Break is across the junction and that no other active component contributes to the momentum equation. A value of 1 indicates that another active component (Pipe, Tee, Vessel, etc.) is on the other side of the junction but that the current component (this Pipe) performs the evaluation of the momentum equation. A value of -1 indicates that another component evaluates the momentum equation, and that component appears before the current one in the order of computation.

The same calls to Junsol initialize the network index array iou. If the junction being processed is an active participant in the network solution (isollb or isolrb is +1 or -1), then the input value for that junction number is placed in the appropriate location in iou. A later call to Setnet from Icomp converts these junction numbers to unique indices for the network junction variables associated with the component junction.

Ipipe, like all other existing 1D component initialization routines, uses a call to the subroutine Iprop to initialize dependent fluid-state variables (density, internal energy, etc.), physical properties such as viscosity, and mixture properties such as the mean density. The actual computation of these properties is done or driven by the subroutines Thermo, Fprop, and Mixprp, respectively. Information is communicated between Iprop and these subroutines via their argument lists. Iprop communicates the information directly to the component-derived-type data structure. Iprop should be used whenever possible for new components. If a replacement is constructed for a special component, care should be taken to understand and mimic the use of the variable "irest" (from module Flt) in Iprop. Many properties (particularly macroscopic densities and energies) must be generated from more basic variables when a component is first input. However, when irest=1, the component data are coming from a restart file, bringing values for many of these variables from the restart file that must not be overwritten during initialization.

Initialization is the first stage at which boundary information is generated and passed. Ipipe uses the standard 1D boundary subroutine Setbd, which accepts arguments

necessary for setting the boundary conditions at the two end cells of the 1D section. Setbd calls J1D for each of the ends to actually set the elements of the "bd"-derived-type array.

The boundary (bd) array is a derived type with one element for each component junction in the system. When calculations begin for a given component, the information in the boundary array for each of its junctions represents conditions in the adjacent component. When a component completes its calculations, its end conditions are loaded over this boundary information so that calculations for the adjacent components will have correct boundary information available. The components of the derived type contain all of the geometry and fluid state information necessary for one component to model flow across the junction from another using a first-order-difference method. For the ith junction, the derived type is as follows:

bd(i)%alp - adjacent cell, old void fraction
bd(i)%alpn - adjacent cell, new void fraction
bd(i)%alpo - adjacent cell ,void fraction from step before old time
bd(i)%ara - adjacent cell, old noncondensable macroscopic gas density
bd(i)%aran - adjacent cell, new noncondensable macroscopic gas density
bd(i)%aratio - ratio of total flow area in this direction to this junction area
bd(i)%arel - adjacent cell, old macroscopic liquid internal energy per volume
bd(i)%areln - adjacent cell, new macroscopic liquid internal energy per volume
bd(i)%arev - adjacent cell, old macroscopic gas internal energy per volume
bd(i)%arevn - adjacent cell, new macroscopic gas internal energy per volume
bd(i)%arl - adjacent cell, old macroscopic liquid density
bd(i)%arln - adjacent cell, new macroscopic liquid density
bd(i)%arv - adjacent cell, old macroscopic gas density
bd(i)%arvn - adjacent cell, new macroscopic gas density
bd(i)%bit - adjacent cell, old bit flags
bd(i)%bitn - adjacent cell, new bit flags
bd(i)%cifn2 - new time interfacial drag coefficient, one face past the junction
bd(i)%concn - adjacent cell, new solute concentration
bd(i)%dfldp - junction derivative of liquid velocity with pressure
bd(i)%dfvdp - junction derivative of gas velocity with pressure
bd(i)%dx - adjacent cell length
bd(i)%eln - adjacent cell liquid, specific internal energy
bd(i)%evn - adjacent cell gas, specific internal energy
bd(i)%fa - flow area one face past the junction
bd(i)%fa - junction flow area
bd(i)%gamn - adjacent cell, new mass-transfer term
bd(i)%grav2 - gravity vector, one face past the junction
bd(i)%hd - junction hydraulic diameter
bd(i)%num - adjacent component number
bd(i)%p - adjacent cell, old pressure

bd(i)%pa - adjacent cell, old noncondensable partial pressure
bd(i)%pn - adjacent cell, new pressure
bd(i)%roa - adjacent cell, old noncondensable gas density
bd(i)%rol - adjacent cell, old liquid density
bd(i)%roln - adjacent cell, new liquid density
bd(i)%rom - adjacent cell, old mean density
bd(i)%rov - adjacent cell, old gas density
bd(i)%rovn - adjacent cell, new gas density
bd(i)%sig - adjacent cell, old surface tension
bd(i)%sxl - coefficient in liquid momentum source from adjacent Tee connection
bd(i)%sxv - coefficient in gas momentum source from adjacent Tee connection
bd(i)%tln - adjacent cell, new liquid temperature
bd(i)%tmsl - constant in liquid momentum source from adjacent Tee connection
bd(i)%tmsv - constant in gas momentum source from adjacent Tee connection
bd(i)%tvn - adjacent cell, new gas temperature
bd(i)%type - adjacent component type
bd(i)%visl - adjacent cell, old liquid viscosity
bd(i)%visv - adjacent cell, old gas viscosity
bd(i)%vl - junction, old liquid velocity
bd(i)%vln - junction, new liquid velocity
bd(i)%vln2 - new time liquid velocity, one face past the junction
bd(i)%vlt - junction, new liquid stabilizer velocity
bd(i)%vlt2 - new stabilizer liquid velocity, one face past the junction
bd(i)%vlto - junction, old liquid stabilizer velocity
bd(i)%vlto2 - old stabilizer liquid velocity, one face past the junction
bd(i)%vlvol - adjacent cell, center liquid velocity
bd(i)%vol - adjacent cell volume
bd(i)%vsi - junction velocity, sign convention translation
bd(i)%vv - junction, old gas velocity
bd(i)%vvn - junction, new gas velocity
bd(i)%vvn2 - new time gas velocity, one face past the junction
bd(i)%vvt - junction, new gas stabilizer velocity
bd(i)%vvt2 - new stabilizer gas velocity, one face past the junction
bd(i)%vvto - junction, old gas stabilizer velocity
bd(i)%vvto2 - old stabilizer gas velocity, one face past the junction
bd(i)%vvvol - adjacent cell, center gas velocity
bd(i)%wfmfl - junction, liquid wall friction input scale factor
bd(i)%wfmfv - junction, gas wall friction input scale factor
bd(i)%xsm - adjacent cell, center x position
bd(i)%ysm - adjacent cell, center y position
bd(i)%zsm - adjacent cell, center z position

Loading this array provides the single most important information interface between components. Unfortunately, the cyclic dual use of each element in the array places strict requirements on the order of processing of components, thus restricting options available for parallel processing. Appendix D presents a future option to generate boundary information in a way that is consistent with the needs for parallel processing and that is more transparent when creating or modifying components.

II.1.4. Solution Prepass

This stage of the calculation includes the stabilizer momentum equation solution and evaluation of various old-time quantities and other bookkeeping necessary at the beginning of each timestep. The basic calling tree associated with the prepass is summarized in Fig. A.5.a of App. A. Again, the Pipe component is used to illustrate the lower-level calling tree. The Pipe prepass is driven by the subroutine Pipe1, which is called by Prep1D. Creating a new component similar to Pipe would require the addition of a Call in Prep1D to handle the prepass for that component.

Subroutine Prep loops over most component subroutines (via Prep1D and Prep3D) twice, communicating the pass number through the variable "ibks" in module OneDDat. Each pass over components begins with a double loop over all 1D system flow loops and all 1D components (including plenums) within each flow loop. The loop index "il" used in Prep1D is needed in lower-level routines to select network-variable indices and network coefficient arrays and is communicated through module OneDDat. In this stage, the network index array is used to reference two variables per network junction: the junction stabilizer liquid and vapor velocities. Variable coefficients for the network equations are passed through "aol" and "aov" and constants through the arrays "drl" and "drv", all contained in the module named "Network". Network coefficients linking 1D velocities to Vessel stabilizer velocities are stored in the "rclvss" and "rcvvss".

The first pass through 1D components results in a call to Preper from Pipe1 (Fig. A.5.b). Preper communicates directly with the component-derived-type data structure and moves most information to lower levels via argument lists. Unfortunately, it currently mixes four types of tasks in its calls to other subprograms. The first of these is general bookkeeping, including calculations of some Vessel junction matrix coefficients and a call to Volv to compute cell-centered velocities. The second task is setup and partial solution for the stabilizer momentum equations, which are delegated to Femom. The third task is calculation of basic physical properties, such as wall friction coefficients (call to Fwall), interfacial drag coefficients (Femom), wall metal properties (call to Mprop), and wall heat-transfer coefficients (HTCs) (call to Htpipe). Modular considerations suggest that at least the task of computing basic physical properties should be isolated in a separate subroutine, which is also called at the component level (e.g., from Pipe1). The fourth task is the evaluation of a component-specific model. If the component type is a pump (Fig. A.5.c), then the subroutine Pumpsr is called to provide pump-momentum source terms. Preper is a transition routine from the standpoint of data communication. It uses the system-wide and component-specific data structures, but passes on information on the state of the fluid through the argument lists of lower-level subroutines.

Boundary information is communicated via Setbd (called from Pipe1, Fig. A.5.b in App. A) during the first pass through all components (ibks=1). However, the second pass through all components (ibks=2) performs only a back substitution on the stabilizer momentum equations. As a result, a standard update to the boundary array is not performed. Only elements associated with the stabilizer velocities are updated directly in subroutine Bkmom. This includes the elements associated with the Tee momentum source terms (sxl, sxv, tmsl, and tmsv), which are just set to zero in Pipe.

In addition to the use of argument lists and the component-derived-type data structure, two other important interfaces are found in subroutine Preper (called by Pipe1). The simplest is the use of the variables im100 and im100x (module GlobalDat) to signal if a timestep backup has occurred (im100 or im100x = -100). In the event of a backup, calculations of cell-center velocities are not repeated. Another important communication path exists to Femom and other flow equation subroutines. These are in principle isolated from the component and system data structures. Indices for the network arrays are communicated to these low-level subroutines through the variables "i01", "i02", and "i03", and the velocity sign conventions at the junctions are communicated via "s01"and "s02". All are contained in the module OneDDat. The sign conventions associated with "s01" and "s02" are related, but not identical to those in "vsi". Within a component responsible for the evaluation of a network junction equation, the value of this sign convention at that junction is always +1. If the component does not evaluate the junction equation at a given junction, then the sign convention contained in "s01" or "s02" (as appropriate) matches that in "vsi" for that junction. A variable "s03" is not needed because the sign convention at the junction between the primary and secondary side of a Tee is known to be +1. The lower-level calling trees for other TRAC components are shown in Figs. A.5.d through A.5.l.

II.1.5. Solution Outer Iteration

This stage is devoted to the iterative solution of basic flow equations for each timestep. The basic calling tree associated with this iteration is summarized in Fig. A.6.a of App. A. The subroutine Hout has primary control of the iterations, but most of the control for a given iteration is contained in subroutine Outer (called by Hout). The Pipe contribution to the outer iteration is driven by the subroutine Pipe2, which is called by Out1D. Creating a new component similar to Pipe would require the addition of a Call in Out1D to handle the outer iteration for that component.

Subroutine Outer loops over most component subroutines (no action is taken on heat structures) twice, communicating the pass number through the variable "ibks" in module OneDDat. Out1D contains DO loops over all 1D system flow loops and all 1D components (including plenums) within each flow loop. The loop index "il" used in Out1D is needed in lower-level routines to select network-variable indices and network coefficient arrays and is communicated through module OneDDat. In this stage, the network index array is used to reference one variable per network junction, which is the cell-centered difference across the junction of the iterative change in pressure. Variable coefficients for the network equations are passed through the array "aou" and constants

through the array "dvb", both contained in the module named "Network". Network coefficients linking 1D junction variables to Vessel new-time pressures are stored the system-wide array in the array "rcvss". Communication of these coefficients is particularly important because Outer completes the computation of the network matrix diagonal, in addition to solving for network variables as functions of Vessel new-time pressures.

Communicating boundary information follows a different path in the outer iteration than in the prepass. The boundary array is set with direct calls to J1D from subroutine Inner, as called by Pipe2 (Fig. A.6.b). Boundary array information also is used in Inner to set values of the new junction velocities (and derivatives with respect to pressure) in the component if they have already been calculated in another component during this stage.

Inner does little besides boundary array operations, calling Tf1D to manage most of the work of setting up and solving the basic equations. Tf1D follows the pattern of Preper in setting network array indices and sign conventions (i01, i02, i03, s01, and s02). It also contains the unfortunate mixture of calls to physical correlations (e.g., the call to Htif, which calculates the interfacial HTCs), with calls to numerical solution subroutines (Tf1Ds1 and Tf1Ds, to set up and partially solve the basic flow equations). Tf1D is a transition routine from the standpoint of data communication. It uses the system-wide and component-specific data structures, but passes on information on the state of the fluid through the argument lists of lower-level subroutines.

One other communicating variable worth noting is "oitno" (named common istat). Its base use is to hold the iteration count, generated in Hout and used to trigger special first-iteration operations in low-level routines such as Tf1D. The count is also used by Newdlt as a contribution to the calculation of timestep size. In the event of an iteration failure, it takes on a second function as a flag to the postpass containing a value of 100.

One flaw in the current code structure and information flow in the outer iteration stage is that equation setup, equation solution, and evaluation of physical models are driven at a relatively low level (Tf1D). A similar situation exists in the prepass with the mixing of equation setup and evaluation of interfacial drag coefficients in subroutine Femom. One key future set of tasks will be to separate these types of calculations at a much higher level (see App. D). The calling tree from Outer to Out3D and lower levels are provided in Fig. A.6.c. Additional lower-level calling trees are provided in Figs. A.6.d through A.6.i.

II.1.6. Solution Postpass

The basic calling tree associated with the postpass is summarized in Fig. A.7.a of App. A. The postpass is driven by subroutine Post and loops through all components three times. As with Prep and Outer, the index for this loop is the variable "ibks" in module OneDDat. This process performs (1) the solution of stabilizer mass and energy equations when ibks = 1 and 2; (2) the solution of the conduction equations and evaluation of fluid properties (viscosity, specific heat, conductivity, surface tension, and heat of vaporization) when ibks = 2 ; and (3) other minor computations necessary to complete

each timestep (mass flows and mean velocity) when ibks = 3. The Pipe contribution to the outer iteration is driven by the subroutine Pipe3, which is called by Post. In addition to driving the most component routines directly, Post calls Post3D to drive postpass activities associated with Vessel components. It also plays a significant role in restructuring the junction mass and energy equations in the single-phase limit and sets the diagonal values of the network matrices. Network matrix information and associated right-hand-side arrays are passed from most lower-level routines via the module named "Network" directly to linear system Solvers (Sgefat and Sgeslt) via their argument lists.

At the component level (Pipe3), boundary information is passed with the same mechanism as in the prepass. Subroutine Setbd is called to set both left- and right-junction boundary conditions, which in turn uses J1D to set individual values at a given junction (Fig. A.7.b).

Solving the stabilizer momentum equations is driven by a call to Constb. Only junction information and bounding indices for arrays are passed to Constb from Pipe1 via the argument list. Constb accesses most necessary information via the system-wide and component data structures and passes this to lower-level subroutines via their argument lists. As with Preper and Inner, Constb must generate network junction indices and sign coefficients (i01, s01, etc.) for the lower-level subroutines (passed through OneDDat). Constb calls Stbme to set up and partially solve the stabilizer equations within the component, passing all information on the state of the fluid through the argument list.

Pipe1 calls Poster (Fig. A.7.b) to handle the back-substitution portion of the stabilizer equation solution, along with several other tasks. This approach is in many respects analogous to the approach in Preper, sharing the problem of combining the evaluation of physical properties (call to Fprop) with bookkeeping and the solution of flow equations (call to Bksstb). As with that routine, Pipe1 should be broken eventually into more flexible modules. Pipe1 uses the module OneDDat as a communications path to pass network matrix indices and sign conventions (i01, s01, etc.) to Bksstb and, as with Preper and Inner, is a transition subroutine from the standpoint of communications. Pipe1 has access to the system-wide and component-specific data structures and passes most of this information to lower-level subroutines through their argument lists.

Post, Pipe3, and Poster receive information on the success of the iterative solution through the variable "oitno" in the named common "istat". On an iteration failure, Post sets the variable "ibks" to two, skipping the equation solution steps. Pipe3 skips a table evaluation for heat sources. On a backup, Poster drives the restoration of all new-time variables to their original old-time values, which are needed to restart the iteration; however, most other tasks in Poster are suppressed. Additional lower-level calling trees are provided in Figs. A.7.c–g.

II.1.7. Restart Dump for Pipe

The basic calling tree associated with the restart dump is summarized in Fig. A.8 of App. A. The restart process is driven by Dmpit, which uses Bfout to write general

variables, calls Rdflt (driver for Bfin calls) to obtain component fixed-length tables, and writes component-specific data with subroutines such as Dpipe. The vast majority of communication throughout the chain of calls is via modules associated with system-wide and component-specific data structures. Switching from modules to argument lists as the means of communication occurs only at the calls to Bfout.

Subroutine Dpipe is very brief, using Bfout directly to write Pipe-specific arrays and calling Dcomp to dump information generic to 1D components. Dcomp obtains the variable-length table using the subroutine Rstvlt and drives the output of both the fixed- and variable-length tables with subroutines Dmpflt and Dmpvlt, respectively (both use Bfout for actual output). It then issues a series of calls to Bfout to write the array data general to all 1D components.

From the standpoint of the restart data interface, it is important to remember that all data are routed to the restart dump file via the subroutine Bfout. The structure of the file itself can be deduced fairly quickly by following the string of Bfout calls under Dmpit. However, because of the potential importance of this interface to later separation of ASCII input processing into a separate program, a full description of the dump file is provided in App. E.

II.1.8. Output of Graphics Data

Names of subroutines initializing and writing graphics information begin with the letters "xtv". However, this does not mean that the output is useful only to the graphics postprocessor named "XTV". This key program interface is well indexed and contains all information necessary to extract data for other data postprocessing, including translation for use by other graphics packages. Although the raw data are written in binary format, the necessary index information is written to a separate ASCII file that is relatively easy to interpret. This section contains the information necessary to use this index information for reading the binary graphics data.

The graphics output is initialized with a call from Init to Xtvinit (Fig. A.1), which sets the descriptive names of all array variables to be written on the binary graphics file (xtvgr.b). Initialization continues with a call to the Xtvdr, using an argument of zero [Call xtvdr(0)]. This value of the dummy argument "xmode" is propagated to lower-level subroutines through argument lists and triggers a mode that writes time-independent component information and index information about time-dependent variables written to "xtvgr.t". Unfortunately, this mix of time-independent and index information requires extra work when writing a program to extract data from the graphics file. Time could be saved by creating a special data extraction program by either adapting source code from XTV or modifying the existing output subroutines (Xtv1D, Xtvplen, ...), which would replace write statements with reads.

The actual binary graphics output is driven by Xtvdr with a value of one passed to the dummy argument "xmode". This call appears in Trans, Steady, and Pstepq. The basic calling tree associated with writing of graphics data is summarized in Fig. A.9 of App. A. Xtvdr begins writing for a timestep by calling Xtvbufs to output the edit time. It then

loops over components calling Rdflt to read the component fixed-length table and a component-specific output subroutine (e.g., Xtvpipe, Xtvtee, Xtvvalv, Xtvvsl, and Xtvplen). It ends with calls to subroutines to output heat structures (Xtvht), signal variables (Xtvsig), and control blocks (Xtvcb). Output to the binary file (xtvgr.b) is buffered and at the lowest level written with the C function "fwrite".

The exact contents of the file "xtvgr.t" vary with the components in the problem and order of execution selected by TRAC for those components. The file begins with the first line from the TRAC input title information. After that, blocks of information follow for all components. Currently, the only supported 1D components are the Pipe, Tee, and Valve, which output the same set of information to "xtvgr.t" by using Xtv1D. Planned upgrades to the graphics output will include the other 1D components, as well as component-specific quantities such as pump speed. Special information blocks exist for Plenum and Vessel components. Information for the "flow" components is followed first by blocks for all heat-structure components, then those blocks for signal variables, and finally those for control blocks.

1D Component Information Block

First Record Group of the Block (single record): c8type, num, nc, ctitle

- c8type - Component type [formatting precedes this with an asterisk (*)]
- num - Component number from the input deck
- nc - Total number of cells in the component
- ctitle - Component title information from the input deck

Second Record Group of the Block (single record): jun1, jun2, jun3, ncell1, jcell

- jun1 - Input junction number for the left (low-numbered) end
- jun2 - Input junction number for the right (high-numbered) end
- jun3 - Input junction number for the Tee side leg, right junction (if present)
- ncell1 - Number of cells in the primary (if Tee)
- jcell - Cell in the primary to which the Tee side leg attaches (if present)

Third Record Group of the Block (eight numbers per record): dx(1 : nc)

- dx - Cell-length array

Fourth Record Group of the Block (eight numbers per record): grav(1 : nc+1)

- grav - Cell-face gravity cosines

Fifth Record Group of the Block (eight numbers per record): fa(1 : nc+1)

- fa - Cell-face flow areas

Sixth Record Group of the Block (single record): nvname

- nvname - Number of variable "arrays" output to xtvgr.b

Seventh Record Group of the Block (nvname records): vname1, fflag1

vname1 - Decriptive name for the variable being output [formatting follows this name with an asterisk (*)]

fflag1 - Flag indicating size of the variable array
= 1, Cell-face variable (nc+1 values)
= 0, Cell-center variable (nc values)
= -1, Scalar variable (1 value)

Current values for vame1, associated values of fflag1 and variable names are:

vname1(1)='volume fraction'	fflag1(1)=0	alp
vname1(2)='pressure'	fflag1(2)=0	p
vname1(3)='saturation temperature'	fflag1(3)=0	tsat
vname1(4)='liquid temperature'	fflag1(4)=0	tl
vname1(5)='vapor temperature'	fflag1(5)=0	tv
vname1(6)='liquid velocity'	fflag1(6)=1	vl
vname1(7)='vapor velocity'	fflag1(7)=1	vv
vname1(8)='flow area'	fflag1(8)=1	fa
vname1(9)='liquid density'	fflag1(9)=0	rol
vname1(10)='vapor density'	fflag1(10)=0	rov
vname1(11)='solute mass fraction'	fflag1(11)=0	concn
vname1(12)='non-condensible pressure'	fflag1(12)=0	pan
vname1(13)='total mass-flow'	fflag1(13)=1	mft
vname1(14)='vapor mass-flow'	fflag1(14)=1	mfv
vname1(15)='mass phase change rate'	fflag1(15)=0	gamn

Plenum Component Information Block

First Record Group of the Block (single record): c8type, num, 1, ctitle

c8type - Component type [formatting precedes this with an asterisk (*)]

num - Component number from the input deck

1 - Total number of cells in the component (always 1)

ctitle - Component title information from the input deck

Second Record Group of the Block (single record): npljn

npljn - Number of junctions connected to the plenum

Third Record Group of the Block (eight numbers per record): junj(1 : npljn)

junj - Component junction numbers associated with connections to the plenum

Fourth Record Group of the Block (single record): nvnamep

nvnamep - Number of variable "arrays" output to xtvgr.b

Fifth Record Group of the Block (nvnamep records): vnamep, fflag

vnamep - Descriptive name for the variable being output [formatting follows this name with an asterisk (*)]

fflag - Flag indicating size of the variable array
= 1, Cell-face variable (npljn values)
= 0, Cell-center variable (1 value)
= -1, Scalar variable (1 value)

Current values for vamep and associated values of fflag are

vnamep(1)='volume fraction'	fflag=0	alpn
vnamep(2)='pressure'	fflag=0	pn
vnamep(3)='saturation temperature'	fflag=0	tsat
vnamep(4)='liquid temperature'	fflag=0	tln
vnamep(5)='vapor temperature'	fflag=0	tvn
vnamep(6)='liquid density'	fflag=0	roln
vnamep(7)='vapor density'	fflag=0	rovn
vnamep(8)='solute mass fraction'	fflag=0	concn
vnamep(9)='non-condensible pressure'	fflag=0	pan
vname1(10)='mass phase change rate'	fflag=0	gamn

Vessel Component Information Block

First Record Group of the Block (single record): c8type, num, nrsx, ctitle

c8type - Component type [formatting precedes this with an asterisk (*)]

num - Component number from the input deck

nrsx - Number of radial cells in the 3D fluid mesh

ctitle - Component title information from the input deck

Second Record Group of the Block (single record): ntsx, nasx, ncsr

ntsx - Number of Theta cells in the 3D fluid mesh

nasx - Number of axial cells in the 3D fluid mesh

ncsr - Number of source connections to the Vessel

Third Record Group of the Block (ncsr records): ir, it, iz, juns

ir - Radial index for the junction cell

it - Theta index for the junction cell

iz - Axial index for the junction cell

juns - Input deck junction number associated with this junction between the Vessel and adjacent 1D component.

Fourth Record Group of the Block (single record):

igeom - Set to 0 if mesh geometry is cylindrical and to 1 if it is Cartesian

Fifth Record Group of the Block (eight numbers per record): rad(1 : nrsx)

rad - Radii of the outer radial-cell faces (igeom=0), or location of upper face in the x direction (igeom=1). The first face is assumed to be positioned at rad=0 or x=0.

Sixth Record Group of the Block (eight numbers per record): th(1 : ntsx)

th - The first face is assumed to be positioned at th=0 or y=0. This provides the Angle (in radians) of the remaining azimuthal cell faces (igeom=0) or location of the remaining cell faces in the y direction (igeom=1).

Seventh Record Group of the Block (eight numbers per record): z(1 : nasx)

z - The first face is assumed to be positioned at z=0. This provides the location of the remaining cell faces in the z direction (igeom=0 or 1).

Eighth Record Group of the Block (single record): nvname3

nvname3 - Number of variable "arrays" output to xtvgr.b

Nineth Record Group of the Block (nvname3 records): vname3, fflag3

vname3 - Descriptive name for the variable being output [formatting follows this name with an asterisk (*)]

fflag3 - Flag indicating size of the variable array
= 1, Cell face variable (nasx*nrsx*ntsx values)
= 0, Cell center variable (nasx*nrsx*ntsx values)
= -1, Scalar variable (1 value)

Current values for vame3, associated values of fflag3, and variable names are given below. Depending on the value of igeom, "xr" designates either the "x" or "r" direction, and "yt" designates either the y or theta (azimuthal) direction. Variables with flag values of 0 (cell center) or 1 (cell edge) have a total of nasx*nrsx*ntsx values written to xtvgr.b:

vname3(1)='volume fraction'	fflag3(1)=0	alpn
vname3(2)='pressure'	fflag3(2)=0	pn
vname3(3)='saturation temperature'	fflag3(3)=0	tsn
vname3(4)='liquid temperature'	fflag3(4)=0	tln
vname3(5)='vapor temperature'	fflag3(5)=0	tvn
vname3(6)='effective wall temperature'	fflag3(6)=0	wallt
vname3(7)='xr liquid velocity'	fflag3(7)=1	vlnxr
vname3(8)='xr vapor velocity'	fflag3(8)=1	vvnxr
vname3(9)='yt liquid velocity'	fflag3(9)=1	vlnyt
vname3(10)='yt vapor velocity'	fflag3(10)=1	vvnyt
vname3(11)='z liquid velocity'	fflag3(11)=1	vlnz
vname3(12)='z vapor velocity'	fflag3(12)=1	vvnz
vname3(13)='solute mass fraction'	fflag3(13)=0	conc

vname3(14)='non-condensible pressure'	fflag3(14)=0	pan
vname3(15)='mass phase-change rate'	fflag3(15)=0	gamn
vname3(16)='xr liquid mass-flow'	fflag3(16)=1	c5p4
vname3(17)='xr vapor mass-flow'	fflag3(17)=1	c5p6
vname3(18)='yt liquid mass-flow'	fflag3(18)=1	c5p3
vname3(19)='yt vapor mass-flow'	fflag3(19)=1	c5p5
vname3(20)='z liquid mass-flow'	fflag3(20)=1	vmfrl
vname3(21)='z vapor mass-flow'	fflag3(21)=1	vmfrv
vname3(22)='vessel water-mass'	fflag3(22)= -1	vlqmss
vname3(23)='core water mass'	fflag3(23)= -1	vcore
vname3(24)='core liquid frac'	fflag3(24)= -1	corelq

Heat-Structure Component Information Blocks

First Record Group of the Block (single record): c8type, num, 1, ctitle

- c8type - Component type (ROD or SLAB) (formatting precedes this with an asterisk)
- num - Component number from the input deck
- 1 - integer 1 (forced value for total number of cells)
- ctitle - Component title information from the input deck

Second Record Group of the Block (single record): nodes, nzmax, nrods, idbci, idbco

- nodes - Number of radial conduction nodes
- nzmax - Maximum permitted number of axial conduction nodes
- nrods - Number of copies of this component
- idbci - Inner-surface boundary option
  = 0, Perfect insulator or, for a ROD, no inner surface
  = 1, Constant HTCs and external temperatures
  = 2, Coupled to fluid cells in one or more components
- idbco - Outer surface boundary option

Third Record Group of the Block (single record) : nvname

- nvname - fixed at 4: Number of variable "arrays" output to xtvgr.b

Fourth Record Group of the Block (four records): Index records for heat-structure scalars

These four records each contain a decriptive name for the variable being output to the binary file, followed by an asterisk and a flag value fixed at zero. Here, the flag value of zero represents a scalar (or array of length 1).

'max avg rod temp'	fflag = 0	tramax
'max hot rod temp'	fflag = 0	trhmax
'inner surf total power'	fflag = 0	tpowi
'outer surf total power'	fflag = 0	tpowo

This general heat-structure information is followed by "nrods" blocks of detailed rod information, one block for each copy of the heat structure.

First Record Group of the Block (single record): c8type, num, nodes, c8type, ncr, ctitle

- c8type - Component type (ROD or SLAB) (formatting precedes this with an asterisk and follows it with a "c")
- num - Component number from the input deck
- nodes - Number of radial conduction nodes
- c8type - Component type (ROD or SLAB)
- ncr - Index number of the ROD copy (a number between 1 and nrods)
- ctitle - Component title information from the input deck (formatting precedes this with "-")

Second Record Group of the Block (single record): nodes, nzmax, ncr, idbci, idbco

- nzmax - Maximum permitted number of axial conduction nodes
- ncr - Index number of the ROD copy (a number between 1 and nrods)
- idbci - Inner surface boundary option
  = 0, Perfect insulator, or (for a ROD) no inner surface
  = 1, Constant HTCs and external temperatures
  = 2, Couple to specified fluid cells in one or more components
- idbco - Outer-surface boundary option

Third Record Group of the Block (eight numbers per record): radrd(1:nodes)

- radrd - Radii of the temperature nodes in the conduction mesh

Fourth Record Group of the Block (single record): nv

- nv - Number of variable "arrays" output to xtvgr.b

Fifth Record Group of the Block (nvname records): vnameh, fflag

- vname - Decriptive name for the variable being output (formatting follows this string with and asterisk)
- fflagh - Flag indicating size of the variable array
  = 1, axial surface node value (nzmax values)
  = 2, two-dimensional (2D) temperature node value (nzmax*nodes values)
  = -1, scalar variable (1 value)

Current values for vame1 and associated values of fflag1 are

vnameh(1)='axial pos'	fflagh(1)=1	zht
vnameh(2)='struct temp'	fflagh(2)=2	rftn
vnameh(3)='inner htc regime'	fflagh(3)=1	ihtf
vnameh(4)='inner Stanton Number'	fflagh(4)=1	stnu
vnameh(5)='inner liq temp'	fflagh(5)=1	tld

vnameh(6)='inner liq htc'	fflagh(6)=1	hrfl
vnameh(7)='inner vap htc'	fflagh(7)=1	hrfv
vnameh(8)='outer htc regime'	fflagh(8)=1	ihtf
vnameh(9)='outer Stanton Number'	fflagh(9)=1	stnu
vnameh(10)='outer liq temp'	fflagh(10)=1	tld
vnameh(11)='outer liq htc'	fflagh(11)=1	hrfl
vnameh(12)='outer vap htc'	fflagh(12)=1	hrfv

Signal Variable-Component Information Blocks

First Record Group of the Block (single record header) '*signal', 1, 1, 'signal variables'

'*signal' - Component type is fixed to '*signal'
1 - Component number is forced to the value 1
1 - integer 1 (forced value for total number of cells)
'signal variables' - Title fixed as 'signal variables'

Second Record Group of the Block (single record): ntsv

ntsv - Number of signal variables. This specifies the number of values written to xtvgr.b

This general signal variable information is followed by "ntsv" blocks of detailed information.

First Record Group of the Block (single record): "sv", id, ':' labsv, '*', 0

'sv' - record begins with the literal string "sv" to mark it as a signal variable
id - Input signal variable ID number (same as input variable idsv)
':' - colon used as a separator
labsv - Label describing the function of this signal variable (generated internally by TRAC). Output formatting follows this label immediately with an asterisk (*) and then the integer zero (0)

Second Record Group of the Block (single record): idsv, isvn, ilcn, icn1, icn2

idsv - Input signal variable ID number
isvn - Signal variable parameter number, which indicates the source of the signal variable (see Table VI-1 in the TRAC/PF1-MOD2 User's Guide, NUREG/CR-5673)
ilcn - Depending on the value of "isvn", this is either a coolant loop number, component number, trip ID number, or not used.
icn1 - Cell number of the first location where the signal variable obtains information
icn2 - Cell number of the last location where the signal variable obtains information

For each of these Signal variable blocks in xtvgr.t, one current time value of the signal variable is written to the file xtvgr.b per time edit.

Control-Block-Component Information Blocks

First Record Group of the Block (single record header) '*control', 1, 1, 'control block output'

'*control'	-	Component type is fixed to '*control'
1	-	Component number is forced to the value 1
1	-	integer 1 (forced value for total number of cells)
'control block output'	-	Title is fixed to this string

Second Record Group of the Block (single record): ntcb

ntcb	-	Number of control blocks. This specifies the number of values written to xtvgr.b on each time edit.

Third Record Group of the Block (ntcb records): "cb", id,'*', 0

'cb'	-	record begins with the literal string "cb" to mark it as a control block
id	-	Input control block ID number (same as input variable idcb)
'*'	-	asterisk (*)
0	-	the integer zero (0)

For each control block listed in xtvgr.t, one current time value of the control block is written to the file xtvgr.b per time edit.

II.1.9. Detailed ASCII Output for Pipe

This stage writes information to the detailed ASCII output file (trcout). The basic calling tree associated with this output task is summarized in Fig. A.10 of App. A. The stage is driven by Edit, which calls Sedit to write summary information for the time and Wcomp to output information on signal variables, control blocks, and component information. Actual component edits are produced one level down by subroutines such as Wpipe, which in turn uses Ecomp to convert data to the requested output units (calls to Uncnvt) and to write this data to the output file (trcout). The dominant communications channels are modules (system-wide and component-specific data structures) down to the calls to Uncnvt, which relies on its argument list.

Unlike the restart dump file, the form of the output is very difficult to trace from the programming. However, the resulting output file is meant to aid the code user and is not intended as an interface to another program. As a result, no description of this file is provided here. This information can be obtained from App. E of the TRAC-PF1/MOD2 User's Guide.[2]

II.2. Special Considerations for a Tee Component

The Tee functions as two Pipe components, one of which has a set of source terms in one cell (indexed by "jcell") to account for flow from the other pipe. Using the source terms currently requires that the section representing the secondary side of the Tee be evaluated first to provide velocity information at the Tee junction. The source terms

involve passing and storing many pieces of information through an indirect communications route. Association of the information with a given Tee requires that the information be stored in the component-specific data structure—in this case, the module TeeVlt. However, the source terms are used at the lowest-level flow equation subroutines (Femom, Tf1Ds1, Tf1Ds, etc.) and must be communicated via a different route. The path chosen was to use variables with similar names in the module OneDDat. Transfer of information to and from this module can be seen in subroutines such as Tee1 and is described in detail later this section.

The functional split of the Tee into two pipes introduces a component junction that is not included in the system-wide storage for junction information (BD and VSI arrays), but is included in the system-wide arrays associated with the network equation solution. This gives rise to a need for several important communicating variables. Boundary information for the low-numbered end of the Tee secondary is stored in the Tee-specific array, derived-data-structure teeAr as the component "teeAr(cco)%bd4". This data structure is contained in the module TeeArray. The index variable "cco" (current component ordered) for the derived-type array teeAr is communicated to the Tee (and all other components) through the module "Global". Every driver routine for components (Prep1D, Out1D, Post, etc.) sets the value of cco = compIndices(i), where "i" is the position of the component in the calculational sequence. The array compIndices also reside in the module named "Global" and is set during input processing after Srtlp has determined the order for processing components.

Because the side leg behaves exactly like a pipe in the network solution, the index of the junction index array (iou) element associated with the side leg must be communicated to the Tee and then on to lower-level solution subroutines. The Tee component routines (Tee1, Tee2, etc.) obtain this value through the variable "ntee", and when driving lower-level routines (Preper, Inner, Poster, etc.) for the side leg, pass this value on through the variable "icme". Both "ntee" and "icme" are contained in the module "OneDDat".

The other significant considerations for the Tee beyond basic boundary communication are related to special source terms. This is first apparent during initialization with the call to Etee from Itee and later in a similar call from Tee3 in the postpass. Etee sets coefficients used by Femom for the implicit evaluation of momentum transport. The following equations express the contribution from the Tee side leg to momentum flux in the liquid motion equation at the right face of the joining cell.

$$\text{flux}_{j+1/2} = \text{r2l}\left[\tilde{V}_{tee}^{n+1}(2\tilde{V}_{tee}^{n+1}\cos(\theta) + \tilde{V}_{j-1/2}^{n})) + \tilde{V}_{tee}^{n}\tilde{V}_{j-1/2}^{n+1} - \tilde{V}_{tee}^{n}(\tilde{V}_{j-1/2}^{n} + \tilde{V}_{tee}^{n}\cos(\theta)))\right] ,$$

where

$$\text{r2l} = \frac{A_{tee}\cdot(1-\alpha_{tee})\cdot\rho_{l,\,tee}}{A_{j+1/2}\cdot(1-\alpha_{j})\cdot\rho_{l,\,j}} .$$

In this case, subscript "j" indicates the junction cell and subscript "Tee" indicates the junction face or first cell in the side leg as appropriate. Subscripts denoting liquid have been dropped from the velocities for brevity. Analogous equations define the values of r2v associated with gas momentum equation at the j+1/2 face, r1l associated with the liquid momentum at the j-1/2 (left) face, and r1v associated with the gas momentum at the j-1/2 face. The values of r1l, r1v, r2l, and r2v are returned from Etee to Itee or Tee3 via the module OneDDat and copied into variables rt1l, rt1v, rt2l, and rt2v, respectively, which are all contained the module TeeVlt. These four TeeVlt variables are copied back to the corresponding OneDDat variables in subroutines Tee1 and Tee2 for use by the low-level subroutines evaluating the equations of motion (Femom and Tf1Ds1).

The problem of Tee momentum source terms is complicated by situations where the side leg connects to either the first or last cell in the primary leg. In this case, it is possible that a momentum source is produced at a component junction but that the motion equation at that junction is evaluated by the adjacent component. As a result, the contributions of the above flux term must be communicated to the adjacent component. The nature of the stabilizer motion equations requires that the contributions to the source flux be communicated in three parts. In the above example equation for $flux_{j+1/2}$, the combination that acts as the coefficient of the new time stabilizer (tilde) velocity at the Tee junction represents a coupling coefficient between two network junction velocities. This coefficient is obtained from the function Teemf1 and is placed in the appropriate element of the network junction matrix for each motion equation [aov(i01,i03) or aov(i02,i03) for gas motion and aol(i01,i03) or aol(i01,i03) for liquid motion] by Femom. This becomes a special case in Femom for which the full equation is not evaluated but the relevant junction array elements are modified to include the effects of side-leg flow. The factor for the new time stabilizer velocity one face away from the component junction is computed by the function Teemf2 and must be passed through the boundary array [bd(i)%sxl for liquid or bd(i)%sxv for gas]. This must also occur for the remainder of $flux_{j+1/2}$. This flux consists of purely old-time quantities that are computed by the function Teemet and passed through bd(i)%tmsl for liquid or bd(i)%tmsv for gas. Because of the special nature of these boundary array terms, they are generated from Etee rather than J1D.

Transfer of these Tee momentum source terms for evaluation of the basic motion equations (done in Tf1Ds1) is less complex because all stabilizer velocities have been determined by this stage of the calculation. When a Tee side leg connects to a primary end cell, the necessary source terms are computed with a call from Tee1 to TBC1, which in turn uses the low-level function Teemom for the detailed source calculation. This occurs just after the final solution for the stabilizer velocities (Bkmom), placing the values in boundary array elements bd(i)%tmsl for liquid and bd(i)%tmsv for gas.

One other special momentum source adjustment must be made for the Tee. Although the Tee side leg is computed just like any Pipe, the velocities feeding into momentum flux terms from the face beyond its low-end junction need special evaluation. This is implemented as special boundary velocities computed as follows:

```
bd4%vl2=c1a*vl(jcell)+c2a*vl(jcell+1)
bd4%vv2=c1av*vv(jcell)+c2av*vv(jcell+1)
bd4%vlt2=c1a*vlt(jcell)+c2a*vlt(jcell+1)
bd4%vvt2=c1av*vvt(jcell)+c2av*vvt(jcell+1)
bd4%vlto2=c1a*vlto(jcell)+c2a*vlto(jcell+1)
bd4%vvto2=c1av*vvto(jcell)+c2av*vvto(jcell+1)
```

Computation of the coefficients such as c1a yields a value of zero if the associated velocity is directed away from the side leg (including the case where the side is angled away from the primary face associated with the velocity). Otherwise, the coefficient is equal to the absolute value of the cosine of the side-leg angle [abs(cost)].

The above elements of bd4 are calculated with a call to JBD4 from Tee1, before calls to Preper (evaluation of the stabilizer motion equations), and the components vvt2 and vlt2 of bd4 are updated directly by Tee1 after the calls to Bkmom (final solution for new-time stabilizer velocities.)

Solution for stabilizer velocities also requires direct use of the coefficients c1a, c2a, c1av, and c2av. As with other such information in TeeVlt, they are copied to variables in OneDDat (ca=c1a, ca1=c2a, cav=c1av, ca1=c2av) for communication to low-level subroutines such as Femom.

Communication of information for mass and energy source terms to the primary leg is currently very direct. Low-level subroutines such as Tf1Ds have direct knowledge of the Tee array structure and access the needed side-leg elements directly. This is not a particularly flexible approach and should be replaced in later modernization tasks.

II.3. Other 1D Components

The remaining 1D components follow the pattern of Pipe, with minor variations. Of these components, Valve and Prizer are the closest to Pipe because neither use special communicating variables in OneDDat. The valve carries additional variables in its variable-length table (module ValveVlt) to model the valve face. Four of these important variables are ivps (the face index at which the flow area is altered), ivty (the valve type), ivsv (the variable containing the signal variable or control block identifier acting as the independent variable in the valve tables), and avlve (area when the valve is fully opened). The full area is needed because the valve area is tabulated or calculated as a fraction of the fully open valve. The valve also carries extra arrays in the module ValveArray to model the valve tables. This additional information is input in RVLVE and Revlve; output by Wvlve, Dvlve, or Xtvvalv; and processed for each timestep by Vlvex (called from Vlve1 when ibks=1). Vlvex transfers the information on the actual area and hydraulic diameter at the valve face to lower-level subroutines by directly altering the elements in the 1D geometry arrays "g1DAr(cco)%fa(ivps)" and "g1DAr(cco)%hd(ivps)" contained in the module Gen1dArray.

The pressurizer follows a similar pattern, but with less additional information. Input information such as the total heater power (qheat) is stored in the module PrizeVlt; qheat is used in the subroutine PRZRX1 (called by PRIZR1 when ibks=1) to compute a heat addition (or subtraction) for each cell and is communicated to low-level subroutines through modification of the array "g1DAr(cco)%hlatw". This array normally is used as part of the communication of heat flow from all heat structures to each fluid cell. The logic in PRZRX1 assumes that contributions from heat structures already exist and adds the contribution from the pressurizer heaters (or sprays).

PUMP differs from the Valve and Prizer components in that it communicates key information to the calculation via OneDDat. General information on scalars such as pump speed are stored in the module PumpVlt, and special arrays defining pump head and torque curves and speed tables are located in the module PumpArray. Calculation of pump head and torque is performed by calls from Pumpsr to Pumpx. Pumpsr has the job of generating a pump momentum source term for use in the 1D hydro subroutines (Femom and Tf1Ds1). This source is the specific work of the pump (in Joules per kilograms) and is stored in the variable smom in PumpVlt and transferred to the OneDDat variable ssmom by Pumpsr and Pump2 for use in evaluating the motion equations in Femom and Tf1Ds1, respectively. Only one source term is generated because the model assumes homogeneous flow at a cell face with a pump source term. This modeling simplification currently requires that Femom and Tf1Ds have knowledge of the component type through use of the module Flt. The nature of the SETS numerical method makes it only practical to consider implicit behavior of the pump source with respect to flow velocity. The derivative of smom with repect to velocity is calculated by Pumpsr and stored as dsmom in PumpVlt. It is copied to "dvjp" in OneDDat by Pumpsr and Pump2. The face at which the source ssmom is applied is communicated by the variable "msc"(in OneDDat), which is set as msc= 2 in Pump1 and Pump2.

As noted in Sec. II.1.4, the subroutine Pumpsr is called by Preper and exists at a lower level than similar subroutines for other components.

Break and Fill are considered to be 1D components to the extent that they use the same 1D array structures to store fluid-state information. They communicate with other components through the boundary (bd) array but do not have the problem of interfacing with low-level hydrodynamic routines because they are fixed or tabular boundary conditions. Tables are handled in a manner similar to Valves and Pumps. Break evaluates necessary tabular information with a call from Break1 to Break and to Fill with a call from Fill1 To Fillx. Either of these components may be configured so that results must be obtained from multiple tables (pressure, temperatures, void fraction, etc.) using the same independent variable. Both use the following method. The first table is processed with Evltab, which takes as one of its input arguments the identifier for the signal variable or control block where the current value is the input to the table (independent variable, or abscissa). Evltab obtains that value by direct access to the control data structure (see Sec. IV.1), uses it in table evaluation, and returns the value through the argument list. Work is saved by evaluating any additional tables with a direct call to the subroutine Linint0, providing this value as an argument.

III. THE PLENUM

The plenum is special in several respects but is still called by driver routines in the same loops as normal 1D components. The Plenum always has just one fluid cell and permits any number of connections to that cell from other active 1D components (no direct Break or Fill connections). To simplify treatment of the connections, all velocity calculations at such connections are forced to occur in the adjacent component. This is accomplished in subroutine Srtlp (called by Input), which places all plenums after all active 1D components in the computational order of a given loop.

Interfaces between the plenum and other fluid components follow the pattern set by Pipe. Fluid conditions are passed through the boundary (bd) arrays, and velocity sign convention information is passed through "vsi". Both of these arrays are contained in the module named "Boundary". The major deviation in this regard is that boundary information is created by a plenum for use by another component through a call to Bdplen. Neither the subroutine Setbd nor J1D is used for this purpose. In the computational flow, calls to Bdplen occur in positions analogous to those of Setbd and J1D in Pipe driver routines.

Information needed for the network matrix solution is communicated via the network arrays contained in the module Network, as with 1D components. However, indices needed to select the correct elements of these arrays are not communicated via the global "iou" array. That array predates the plenum and contains only enough space for components with three junctions. The module PlenArray contains a derived-type array plenAr. One component of plenAr is the array "i0j", which contains the network indices for all connections in that component. For the ith junction of the current plenum, the network index would be"plenAr(cco)%i0j(i)". The elements of these plenum index arrays are generated during initialization in the subroutine IPLEN. Unlike 1D components, the values contained in "i0j" are not communicated to the low-level hydro subroutines via intermediate variables. The plenum uses its own hydro routines and passes the index array directly through the argument list.

IV. VESSEL

Vessel is heavily isolated from the other components in both computational (see App. A) and data flow. Information is received and returned by Vessel through Boundary and Network module arrays. This is important from the standpoint of adding new 1D components. When standard conventions are followed in setting the boundary and network array contributions from the 1D component, the interface to a Vessel is automatic. One important part of following these conventions is that the new components must be processed at the same stages of the computational cycle as other 1D components. As currently configured, Vessels must be processed after all other fluid components. This restriction will be lifted after the modularization of the network solution (see App. D).

The Vessel component is scheduled for modernization after completion of the initial modernization tasks. As a result, a detailed description of data interfaces internal to the Vessel is not being provided in this document at this time. Vessel follows the same stages of computation documented for the pipe, with analogous calls to lower-level subroutines. Boundary information is returned from Vessel by calls to J3D, which is analogous to J1D. Within the calling tree for Vessel, incoming boundary information is translated to Vessel source arrays at the component routine level (Vssl1, Vssl2, and Vssl3) and passed to hydrodynamic routines (Tf3Ds, etc.) so that the "bd" array is not seen at the same depth as in the 1D routines. Unlike the 1D component, fluid-state information is passed to the hydro routines via the data structure (module VessEquiv) and passed via the argument list only to low-level state and property subprograms (Thermo, Htif, etc.).

Establishing an interface between Vessel and a 3D kinetics package is discussed in the following section on heat structures. The mechanism used to extract fluid-state information from Vessel for the calculation of HTCs should be mimicked.

V. HEAT STRUCTURES

Before dealing with programming related to the heat structure, it is useful to understand the history of development for that component. The low-level subprograms for I/O, initialization, material properties, and solution of the conduction equation originated with the rod conduction capabilities that were integral to the Vessel component in early versions of TRAC. Thus, it is common to see subprogram and variable names containing the string "rod" at lower levels. Because these rods reside in the core region of Vessel, driver subroutines for the prepass and postpass had the names Core1 and Core3. Creating the heat-structure component in the mid-1980s expanded on the key conduction subroutines of both Vessel and a later 1D Core component, making both obsolete. However, naming conventions were never altered, and only the highest-level component routines (Cihtst, Htstr1, etc.) carry names directly associated with the component. Low-level subprograms and arrays carry names suggesting a rod (cylindrical) geometry, when in fact they also contain capabilities for modeling Cartesian geometry.

The heat structure represents a special component class that is calculated separately from the fluid components during a given timestep and is explicitly coupled only to the fluid calculation. The structure participates in all stages of computation (Sec. I) except the fifth, which is devoted solely to the iterative solution of the basic flow equations. Discussions of multiple passes through components during stages 3-6 (see Secs. I and II) are not applicable to the heat structures. The component-level drivers for initialization (Cihtst), prepass (Htstr1), and postpass (Htstr3) are called only once during each of these stages.

The most significant data interface issues involve the details of the coupling between heat structures and fluid components. The code currently operates under two restrictions in this respect. Heat structures can never be connected to Plenum components or to Vessel cells that contain source terms. This simply reflects a development history in which time was not available to adapt to the Plenum data structure or include Vessel source velocities in the evaluation of heat-transfer correlations.

Construction of the interface to fluid components begins with input from either Rhtstr or Rerod1 of information on fluid cell connections. Variables containing the string "hcom" are integer arrays containing the component numbers for cells beginning one cell below the heat structure and ending one cell above the structure. The two extra cells are necessary when liquid levels must be detected to refine the sub-grid approximation of HTCs. Variables containing the string "hcel" are integer arrays containing the cell number corresponding to the same elements in the "hcom" arrays. When one of these variable names (or pointers) ends with "i", it is storing information for the "inner" surface of the structure, and when it ends with "o", it is storing information for the "outer" surface connections.

More communication information is stored in the arrays lchci and lchco during initialization. This is generated by a combination of actions in subroutines Irodl, Lchpip, and Lchvss. For the ith connecting cell in the hcel-hcom information, the corresponding "lchci" array contains the component type in element lchci(2,i) (similar situation for lchco). The element lchci(1,i) contains necessary offset information to locate fluid state and property information in the component arrays. This is a requirement of the old container array structure, is obsolete for the 1D components, and will become obsolete for 3D components when that data structure is altered.

The first active communication occurs in the prepass. Here, heat-structure computations are driven by the subroutine Htstr1. It begins the timestep by moving all new time heat-structure information into the old-time arrays. Next, it picks up fluid information with a call to Fltom, signaling the direction and scope of data movement by passing a value of one through the last argument. A call from Htstr1 to Core1 drives the bulk of the prepass calculations, and, as a final step, the information on HTCs and wall temperatures is passed back to the appropriate fluid cells with another call to Fltom from Htstr1. In this case, the reversed data flow and scope of information are signaled with a final argument value of -1.

Communication of information between Core1 and lower-level subroutines is dominantly through argument lists. Activity in the subroutine Core1 is centered on the generation of HTCs. This is complicated by the need to maintain coefficient and wall temperature information on both the fine axial conduction mesh and the coarse mesh associated with the fluid cells. Core1 begins with a call to TRIP, which checks to see if the trip has been set to activate the fine mesh. The next major action is a call to Mfrod, which sets the material properties for the structure. Core1 proceeds to make any necessary adjustments in the fine mesh with calls to Shrink, Expand, and Fnmesh. HTCs and critical heat-flux temperatures are generated with calls to Zcore, Htvssl, and Htcor as needed. As these calls are made, Core1 computes coarse node integrals for six quantities: (1) the product of liquid HTC and wall area (hlar); (2) the product of vapor HTC and wall area (hvar); (3) the product of liquid HTC, wall temperature, and wall area (hlatr); (4) the product of vapor HTC, wall temperature, and wall area (hvatr); (5) the wall area (watr); and (6) the product of subcooled boiling heat flux and wall area (hgamr). At the same time, coarse node averages are computed for the critical heat flux (CHF) and the temperature of the CHF. Core1 concludes with any necessary calculations for heat sources, including possible calls to obtain reactivity feedback information (Rfdbk) and to compute the solution of the point kinetics equations (Rkin, also used for tabulated power sources).

An understanding of Fltom is important to any near-term creation of interfaces to the fluid or heat-structure data. However, it should be noted that this method of communication will be replaced in the late modernization stages of system service activities described in App. D.

Fltom contains nested DO loops that cycle over all elements of the heat structure (indexed by ncr=1,nrods) and over all active coarse axial levels within each element or

all fluid cells communicating with the element (indexed by nz=nzl,nzu). This sweeps over all cell interfaces between conduction and fluid cells. The loop structure has separate blocks for processing inner and outer surfaces. Any necessary shifts in the fluid cell location are computed based on the input array "idrod", which is used to shift the location of structure copies relative to the base location given in the "hcel" array (shift stored in the variable nzz). The actual transfer of information is accomplished in Piprod for 1D components or Vssrod for Vessels. One call is made to one of these subroutines for each communicating fluid cell. The final dummy argument for FLTOM is passed on as the final argument to Piprod and Vssrod to give them information on the scope and direction of data transfer.

Piprod begins by determining the position of the connecting fluid component in the ordered component data arrays (stored in "i"). If the final argument (imfl) is negative, data are moved to the fluid component arrays, which is required to compute heat sources (hgam, hla, hva, hlatw, hvatw, and wat) and inverted annular flow (finan). This is accomplished with calls to the subroutine "IncrementGen1d", which adds the current heat-structure contributions to whatever has already been contributed to the fluid cell from other heat structures. If "imfl" is 2, Piprod will bring new-time fluid temperatures (tln and tvn) into the heat-structure data (tlnr and tlnr) with two references to the function "GetGen1d" (used only from the postpass). If "imfl" is one, Piprod loads all necessary fluid-cell geometry, fluid-state, and fluid-property information into the corresponding heat-structure arrays. In this process, a negative value of "ihcel" (fluid cell number) is used as a flag if the heat structure is oriented upside down relative to the fluid cell. This produces a different offset (ioff) for fetching the appropriate cell face velocity.

Vssrod follows a pattern similar to Piprod, using the sign of the axial position index "iz" as a flag on reversed relative orientation. For negative "imfl", additional information is transferred to the Vessel database when the new reflood model is active. This is used in calculating interfacial drag and HTCs, which are consistent with conditions created by the heat transfer. All data transfers in Vssrod currently occur via direct FORTRAN assignment statements involving locations in the Vessel and heat-structure arrays. This will be changed to the use of subroutines similar to "GetGen1d" and "IncrementGen1d" when modernization of the Vessel and heat-structure data structures is completed.

Heat conduction computations are driven in the postpass by subroutine Htstr3. It contains necessary operations for a timestep backup (oitno=-100), but normally just updates fluid temperature information calling Fltom with the last argument (imfl) set to 2 and drives the conduction calculation with a call to Core3. It completes the timestep with a call to Htstrp to compute the instantaneous power and energy content for all heat structures for use in energy balance calculations.

The subroutine Core3 basically acts as a transition between data-structure and argument-based communication of information. It loops over all copies of the heat structure, calling Frod with all necessary physical information passed through the Frod argument list. Frod uses Hgap to obtain gap HTCs (if needed) and calls Rodht to perform the actual 2D transient conduction calculation.

VI. NONCOMPONENT MODELS

There are six major noncomponent models in TRAC-M: signal variables, control blocks, trips, constrained steady state, hydraulic-path steady-state initialization, and radiation. The radiation model is not currently functional, but will be reimplemented in the 3D modernization project. Documentation on the interface to the radiation model will be provided at that time.

VI.1. Signal Variables, Control Blocks, and Trips

Signal variables, control blocks, and trips are tightly linked both in the computational and data structures of the code. They are input to the code by a call to Rcntl from subroutine Input or on restart by a call to Recntl from Rdrest. Evaluation of these variables takes place once each timestep through a call to Trips at the very beginning of subroutine Prep. Trips loops over the evaluation of all signal variables and control blocks and trips a total of "ntcp" times to permit iterative evaluation of controls. The user sets the variable "ntcp" in the input data file. A suibset of all signal variables, control blocks, and tribs may be evaluated during each of the "ntcp" loops. Actual evaluation of signal variables is controlled by Svset, which in turn calls Svset1 to evaluate signal variables associated with 1D components, Svset3 to evaluate signal variables associated with 3D (Vessel) components, and Svseth to evaluate signal variables associated with heat structures. Once signal variables are evaluated, Trips calls Cbset to evaluate control-block functions 1 to 61 in Conblk and control-block functions 100 (time delay), 101 ([fcn(x)], and 102 ([fcn(x_1, x_2, x_3)] during all three stages of a timestep. Trips numbered 200 [proportional-integral (PI) controller] and 201 [proportional-integral differential (PID) controller] are evaluated with a call from Trips to Trpset.

Values from signal variables, control blocks, and trips are accessed by most components, usually during the prepass. The trip status is easily accessed through use of the subroutine Trips. The ID of the trip to be queried is passed as the first argument to Trips. The second through fourth arguments return the set status (iset), the time since last reset (tlaps), and the difference between the input signal to the trip and its setpoint value (delsv). Unfortunately, no such subroutines are available to obtain values of signal variables and control blocks. Components needing this information access the control data structure directly.

The control data structure currently is still based on integer pointers into container arrays. Integer values are stored in the array "ict", and reals are stored in "act". The pointer to the beginning of data in each of these arrays is "lcntl", which is set as "lcntl=1" in subroutine Input. Pointers to the arrays are still in the form used by old TRAC versions, which assume that integers and reals have the same length and physically occupy the same array space. This results in wasted space in "ict" and "act" but eases the transition to a modernized code. The layout of the control data arrays is provided at the beginning of Rcntl and reproduced on the following page. Coding in component routines to access values of signal variables or control blocks is dependent on a knowledge of this structure. Extraction of these values will become a much more transparent process after full modernization of the data structure.

The following tabulation provides a definition of control parameters stored in the act or ict array [Note: act(1) = act(lcntl)].

Contents of act(lcntl+pointer) to act(pointer+length-1)	**Pointer**	**Length**
Dimension Data	1	11
Total number of control-parameter storage locations required	1	
ntsv, total number of signal variables	2	
ntcb, total number of control blocks	3	
ntcf, total number of control-block tabular data entries	4	
ntrp, total number of trips	5	
ntcp, number of control-parameter-evaluation passes	6	
ntse, total number of different signal-expression-trip signals	7	
ntct, total number of different trip-controlled-trip signals	8	
ntsf, total number of different set-point-factor tables	9	
ntdp, total number of trips that when set on generate a restart dump	10	
ntsd, total number of trips that when set on initiate use of special time-step data	11	
Control-parameter-evaluation pass data	12	6*ntcp
- The following six parameters define each of the ntcp evaluation passes		
isv1, smallest signal-variable id number evaluated during the evaluation pass		
isv2, largest signal-variable id number evaluated during the evaluation pass		
icb1, smallest (magnitude) control-block id number evaluated during the evaluation pass		
icb2, largest (magnitude) control-block id number evaluated during the evaluation pass		
itp1, smallest (magnitude) trip id number evaluated during the evaluation pass		
itp2, largest (magnitude) trip id number evaluated during the evaluation pass		

- After all the control parameters are read from input and the restart file, subroutine order rearranges each of the three control-parameter types in order of increasing id number magnitude; the above six parameters for each evaluation pass are then redefined with their corresponding control-parameter type list (do loop) index number

Signal Variable Data — 12+6*ntcp — 7*ntsv

- The following seven parameters define each of the ntsv signal variables

idsv, the signal-variable id number
isvn, the signal-variable parameter number
ilcn, the coolant loop number, component number, or trip id number where the signal variable is defined
icn1, the first-location cell number
icn2, the second-location cell number
6th parameter, previous value of the signal variable
7th parameter, present value of the signal variable

Control-Block Data — 12+6*ntcp +7*ntsv — 17*ntcb

- The following seventeen parameters define each of the ntcb control blocks

idcb, the control-block id number
icbn, the control-block mathematical operation number
icb1, the first input-parameter id number
icb2, the second input-parameter id number
icb3, the third input-parameter id number
cbgain, the control-block gain factor
cbxmin, the minimum value of the control-block output parameter
cbxmax, the maximum value of the control-block output parameter
cbcon1, control-block first constant

cbcon2, control-block second constant
flag1, the control-block integration limit flag (a2)
flags, the control-block output status flag (a2)
cbin1, the control-block first input-parameter value
cbin2, the control-block second input-parameter value
cbin3, the control-block third input-parameter value
16th parameter, previous value of the control-block output parameter
cbout, present value of the control-block output parameter

Control-Block Tabular Data - Storage space is reserved here for the tabular data from all icbn=100 101,and 102 control blocks	12+6*ntcp +7*ntsv +17*ntcb	ntcf
Control-Block parameter units label data and trip parameter units label data and trip signal-expression constant units label data	12+6*ntcp +7*ntsv +17*ntcb +ntcf	6*ntcb +ntrp +5*ntse
Trip Data - The following eighty parameters define each of the ntrp trips idtp, the trip id number isrt, the signal-range-type number iset, the trip-set-status number itst, the trip-signal-type number idsg, the id number defining the trip signal setp(i) for i=1,4; the ith set-point value dtsp(i) for i=1,4; the ith set point's delay time ifsp(i) for i=1,4; the ith set point's set-point-factor table id number 18th parameter, delsv, the difference between the trip-signal value and the set-point value that turns the trip off 19th parameter, the previous value of this trip's signal 20th parameter, the previous subrange	12+6*ntcp +7*ntsv +17*ntcb +ntcf +6*ntcb +ntrp +5*ntse	80*ntrp

integer number where the trip-signal value resided (1=left, 2=center or right, 3=right)

Up to five pending set-status changes are defined by the following twelve parameters; the first pending set-status-change data is stored in the 21th to the 32th parameters, the second is stored in the 33th to the 44th parameters, etc.

parameter 1, the trip-signal sub-range number corresponding to the pending set-status change

parameter 2, the problem time at which the trip is to be changed to this set status

Five data pairs are stored here to save the problem time and the delsv (18th parameter) values over a past time interval corresponding to the delay time of the set point that turns the trip off; these data are stored so that a delay time shift can be incorporated into the evaluation of delsv for the 18th parameter; in this way, the value of delsv goes to zero at the same time that the trip is turned off

Trip-Signal-Expression signal data	12+6*ntcp +7*ntsv +17*ntcb +ntcf +6*ntcb +ntrp +5*ntse +80*ntrp	38*ntse

- The following thirty-eight parameters define each of the trip-signal-expression signals that are different

idse, the trip-signal-expression id number

inse, the number of subexpressions defining the signal expression

incn, the number of different constants referenced by the subexpressions

The following set of three parameters for each of ten subexpressions are stored here

parameter 1, the id number of the

```
          arithmetic operator used by the
          subexpression
       parameter 2, the id number of the
          subexpression's first argument
       parameter 3, the id number of the
          subexpression's second argument
   Storage space is reserved here for
          five constants used as arguments
          in the subexpressions

Trip-Controlled-Trip signal data              12+6*ntcp  12*ntct
   - The following twelve parameters           +7*ntsv
     define each of the trip-controlled-       +17*ntcb
     trip signals that are different             +ntcf
   idtn, the trip-controlled-trip id           +6*ntcb
          number                                 +ntrp
   intn, the number of trips whose set-        +5*ntse
          status values define this trip's      +80*ntrp
          signal                                +38*ntse
   Storage space is reserved here for ten
          trip id numbers whose set status
          define the trip signal

Trip Set-Point-Factor table data             +12+6*ntcp 23*ntsf
   - The following twenty-three para-          +7*ntsv
     meters define each of the set-point       +17*ntcb
     factor tables that are different            +ntcf
   idft, the set-point-factor-table            +6*ntcb
          id number                              +ntrp
   idsg, the id number for the set-            +5*ntse
          point-factor table's independent     +80*ntrp
          variable                             +38*ntse
   inft, the number of set-point-factor        +12*ntct
          table entry pairs of data
   Storage space is reserved here for ten
          pairs of independent variable
          and set-point factor values of
          tabular data that define the
          set-point-factor table

Trip-initiated restart dump and problem
   termination data                          12+6*ntcp  min(ntdp,1)
   - The following 1+ntdp parameters          +7*ntsv     +ntdp
     define the trips that when set on       +17*ntcb
     generate a restart data dump and          +ntcf
     possible problem termination            +6*ntcb
   ndmp, the number of trip id numbers         +ntrp
          controlling dumps and exit           +5*ntse
```

```
   Storage space is reserved here for        +80*ntrp
         ntdp trip id numbers                +38*ntse
                                             +12*ntct
                                             +23*ntsf

Trip-initiated time-step data                12+6*ntcp  15*ntsd
   - The following fifteen parameters         +7*ntsv
     define each of the ntsd special         +17*ntcb
     time-step data sets                        +ntcf
   ndid, the id number for this time-         +6*ntcb
         step data set                         +ntrp
   ntid, the number of trip id numbers        +5*ntse
         that initiate use of this           +80*ntrp
         time-step data                      +38*ntse
   Storage space is reserved here for        +12*ntct
         five id numbers of trips that       +23*ntsf
         initiate use of this time-step      +min(ntdp,1)
         data                                   +ntdp
   dtmin, the minimum time-step size
   dtmax, the maximum time-step size
   dtend, the problem time interval
         during which these time-step
         data are to be applied
   dtsof, the new timestep or the factor
         to be applied to the existing
         timestep for defining the time
         step to be used
   edint, the print edit time interval
   gfint, the graphics edit time interval
   dmpit, the restart dump edit time
         interval
   sedint, the short print edit time
         interval

etime, the problem time corresponding to
   this control parameter data               ict(1)       1
```

The total number of control-parameter storage locations required is

ict(1) =12+6*ntcp+7*ntsv+17*ntcb+ntcf+6*ntcb
+ntrp+5*ntse+80*ntrp+38*ntse+12*ntct
+23*ntsf+min(ntdp,1)+ntdp+15*ntsd

PI- and PID-controller blocks have slightly different definitions for some of the 17 stored values. If kcb is the index pointing to the element just before the beginning of PI- or PID-controller block data, then the following definitions are in effect:

act(kcb+5) = first constant wt of the weighted summer (was icb3)
act(kcb+11) = desired delta (Δt) error removal time constant (was flag3)
act(kcb+12) = first-order lag tau (τ) time constant (was flags)
act(kcb+13) = previous delta (ΔF) error (was cbin1)
act(kcb+14) = previous (gain * time integral of delta F) (was cbin2)
act(kcb+15) = previous (gain * delta F time derivative + delta f) (was cbin3)

VI.2. Constrained Steady State

There are three types of steady-state calculations: generalized steady state (GSS) for Stdyst=1 or 3, constrained steady state (CSS) for Stdyst = 2 or 4, and static-check steady state (SSS) for Stdyst = 5 (Word 1 on Main Data Card 4). A GSS calculation evaluates a pseudo-transient timestep solution that asymptotically converges to the steady-state solution. Convergence is based on the maximum fractional change per second of seven hydraulic-solution parameters, all being less than the input FORTRAN variable EPSS. A CSS calculation is a GSS calculation where additional user-defined component-action adjustments are made by a PI controller to achieve a known and/or desired hydraulic steady-state condition. The nature of the available CSS controllers and their evaluation and database are described in this section. An SSS calculation checks for erroneous momentum and heat sources in a plant model by having TRAC not evaluate the pump momentum source and heat transfer so that fluid flow is expected to go to zero asymptotically.

A CSS controller adjusts an uncertain component-action state to achieve a better-known hydraulic condition in the steady-state solution. There are four types of CSS controllers from which the TRAC user can select. Each type can be applied to one or more components in a plant model. A type-1 CSS controller adjusts a pump impeller's rotational speed to achieve a desired fluid mass flow through the Pump component. A type-2 CSS controller adjusts a valve's flow-area fraction to achieve a desired adjacent-cell upstream fluid pressure or fluid mass flow through the Valve component's adjustable interface. A type-3 CSS controller performs one of three different adjustments (pump-impeller rotational speed of a Pump component, flow-area fraction of a Valve component, or mass flow in or out of a Fill component) to achieve a desired fluid mass flow through its component that equals the fluid mass flow at a designated location in the plant model. A type-5 CSS controller performs one of four different adjustments to a Htstr component or its hydraulically coupled Break components (hydraulic-channel fluid pressure at the inner or outer surface; heat-transfer area at the inner, outer, or both surfaces; thermal conductivity of the inner, outer, or both surface nodes or of all nodes; or heat-transfer area of both surfaces and thermal conductivity of all nodes) to achieve a desired single-phase fluid temperature or two-phase gas volume fraction at a designated location in the plant model. The type-4 CSS controller was eliminated when the Stgen component was removed from TRAC. It adjusted the secondary-side fluid pressure or the tube inner and outer heat-transfer areas of a steam generator to achieve a desired primary-side downstream-location liquid temperature. By remodeling a Stgen component with Pipe, Tee, and Htstr components, the functionality of the type-4 CSS controller is provided by a subset of the functionality of the type-5 CSS controller.

Each of the Ncontr user-defined CSS controllers requires one input-data record with four or five values read by Input (adjusted-component ID number, minimum and maximum range of parameter adjustment, the type or location of the monitored parameter that is to have a desired value, and the type of adjusted parameter). Each CSS controller's desired hydraulic-parameter value is input at its monitored-parameter location in the component data. CSS-controller data are not written to the dump/restart file and so need to be reinput by the Tracin file if the CSS calculation is continued with a restart. The number of CSS controllers and their input parameters can be changed during a restart. Components defining the desired hydraulic-parameter value for each CSS controller also need to be reinput by the Tracin file. This later requirement makes restarting a CSS calculation inconvenient. Generally, TRAC users evaluate a CSS calculation to steady-state convergence without doing a CSS-calculation restart.

After reading a component's data from the Tracin file, subroutines Rpump, Rvalve, Rfill, and Rhtstr determine if their component is being adjusted by a CSS controller when Stdyst = 2 or 4. If a CSS controller is being applied to the component, the desired hydraulic-parameter value is obtained from its specified location for types 1, 2, and 5 controllers. Type-3 CSS controllers get their desired fluid mass flow each timestep from its specified location in the plant model. For the Ith CSS controller (where I = 1, 2, . . . , Ncontr), a signal variable with ID number 9900+I is created to monitor the desired hydraulic-parameter value at its specified location, and a PI-controller control block with ID number –(9900+I) is created to evaluate the adjustment of the component-action parameter. Signal variable ID numbers >9900 and _9999 and control block ID numbers <–9900 and –9999 are reserved for CSS-controller parameters defined internally by TRAC.

The Kth type-3 CSS controller, which adjusts a Pump or Valve (where K = 1, 2, . . . , Nconts), requires a second signal variable with ID number 9900+Ncontr+K to monitor the pump-impeller interface or adjustable-valve interface fluid mass flow. The difference between the 9900+Ncontr+K signal-variable fluid mass flow and the 9900+I signal-variable fluid mass flow drives the PI-controller control block adjustment of the pump-impeller rotational speed or the valve adjustable-interface flow-area fraction. A PI-controller control block is not defined for a type-3 CSS controller, which adjusts the in-or-out fluid mass flow of a Fill component. That is because the 9900+I signal-variable fluid mass flow determines the Fill-component fluid mass flow directly for the next timestep. An absolute-value function control block with ID –(9900+I) of the 9900+I signal variable's fluid mass flow is defined instead. It is this absolute-value fluid mass flow with a positive sign for outflow from the Fill and a negative sign for inflow to the Fill that is defined as the adjusted fluid mass flow of the Fill component.

There are Ncontp type-5 CSS controllers that adjust the hydraulic-channel fluid pressure at the inner or outer surface of an Htstr component. They each have 50 elements of the A array reserved to save the ID numbers of all Break components that are hydraulically coupled to the adjusted Htstr. The fluid pressure is adjusted by the Htstr's PI-controller in all hydraulically coupled Break components. An ID list of Valve components that are

closed and not adjusted by a CSS controller is saved by Input in array Numvc(N) for N = 1, 2, . . . , Nvc _ 50. This ID list is used by Fbrcss (called by Input) to determine all Break components that are hydraulically coupled to the Htstr. Breaks separated from the Htstr by these Valve components that are closed and not adjusted by a CSS controller are not considered to be hydraulically coupled to the Htstr component.

Data storage for CSS controllers consists of the 5*Ncontr input-data values in the A array with pointer Lcontr, values of Ncontr and Nconts in common block Contrl, 7 values for each of the Ncontr+Nconts created signal variables, 17 values for each of the Ncontr created control blocks, 50*Ncontp elements of the A arrays with pointer Lcontp for the Break components hydraulically coupled to type-5 CSS controllers, and 50 elements of the Numvc array in subroutine Input.

The CSS-controller signal variables and control blocks are evaluated for each timestep by subroutine Prep, which calls Trips, which calls Svset for signal variables and CBSET for control blocks (see VI.1). Then subroutine Prep:

1. for types-1 and -3 CSS controllers applied to a Pump component, calls Prep1D, which calls Pump1, which calls Preper, which calls Pumpsr to apply the PI-controller control-block-determined pump-impeller rotational speed to the Pump component;

2. for types-2 and -3 CSS controllers applied to a Valve component, calls Prep1D, which calls Vlve1, which calls Vlvex to apply the PI-controller control-block-determined adjustable-valve interface flow-area fraction to the Valve component;

3. for type-3 CSS controllers applied to a Fill component, calls Prep1D, which calls Fill1, which calls Fillx to apply the absolute-value control-block-determined in-or-out fluid mass flow to the Fill component; and

4. for type-5 CSS controllers applied to a Htstr component, calls Htstr1, which calls Core1 to apply the PI-controller control-block-determined heat-transfer area and/or thermal conductivity to the Htstr component. For type-5 CSS controllers that adjust the hydraulic-channel fluid pressure at the inner or outer surface of the Htstr, Prep calls Prep1D, which calls Break1 to apply the PI-controller control-block-determined adjustment of the pressure boundary condition for Break components hydraulically coupled to the Htstr component.

Interactive feedback between CSS controllers needs to be considered by TRAC users when defining them. Their derived form assumes no interactive feedback. When the adjustments of two or more CSS controllers are strongly coupled by the thermal-hydraulic solution, their predicted controller adjustments may be bad, causing the solution to wander and not converge to the desired thermal-hydraulic parameter values. One such interaction has been programmed for in TRAC. When a type-5 CSS controller adjusts the fluid pressure where a type-2 CSS controller defines the desired value for an upstream fluid pressure, the pressure adjustment of the type-5 CSS controller also is

applied to the desired value for the type-2 CSS controller's upstream fluid pressure. The desired value of the upstream fluid pressure becomes a moving target for the type-2 CSS controller, just as the desired fluid mass flow at a specified location in the plant model for a type-3 CSS controller becomes a moving target when it varies each timestep.

The CSS-controller-adjusted component-action parameter is part of the component data that are output to the restart-data Trcdmp file. Signal variables with ID numbers >9900 and _9999 and control blocks with ID numbers <–9900 and _–9999 are not output to the restart-data Trcdmp file. Restarting a transient calculation from the last (or any one) of these data edits will maintain the CSS-adjusted value of the component-action parameter at the start of the transient. That value remains constant until a user-defined component-action adjustment defined by the component data is applied during the transient calculation.

These four CSS-controller types are programmed for user convenience. An equivalent controller (except for the heat-transfer area and thermal conductivity adjustments of a type-5 CSS controller) could be defined directly through input with signal variables, control blocks, and component actions of the TRAC control system. For controller types that are not programmed, the TRAC user can define them through input as long as the controller's adjustment is an existing component action (see Table 2-4 in the TRAC Theory manual[3]). Additional component-action and CSS-controller types could be programmed if their availability is required by the user community.

VI.3. Hydraulic-Path Steady-State Initialization

The initial thermal-hydraulic steady-state solution estimate, user specified by the hydraulic-component input data, generally can be improved by the hydraulic-path steady-state initialization procedure in TRAC before the steady-state calculation is evaluated. Doing this generally reduces the computational effort of the steady-state calculation. The user selects this option by adding 2 to the value of Stdyst for a GSS or CSS calculation; i.e., Stdyst=1 or 2 for a GSS or CSS calculation may be defined as Stdyst=3 or 4 for a GSS or CSS calculation with its initial thermal-hydraulic steady-state solution estimate internally initialized by TRAC during the initialization phase of a GSS or CSS calculation.

Choosing the hydraulic-path steady-state initialization-procedure option requires the TRAC user to input hydraulic-path steady-state initialization data in the TRACIN file. These input data are defined by the input-data format description in Sec. 6.3.4 of the TRAC User's Guide.[4] In specifying these data, the 1D hydraulic-component network of the plant model is partitioned into Npaths connecting and nonoverlapping 1D flow paths. All possible 1D flow paths in the network are considered unless the input 1D hydraulic-component data already define such a flow condition (and is not connected to a Plenum component) or the ID path's steady-state flow is not expected to be significant. Even 1D paths without flow may be considered in order to define an appropriate thermal condition (not defined by the 1D hydraulic-component data). The input hydraulic-component data need to be defined only as isothermal and no flow when

selecting the hydraulic-path steady-state initialization option. During the initialization phase, TRAC replaces the hydraulic-component gas volume fraction, phasic temperatures, and phasic velocities input data with the thermal-hydraulic parameter values specified by the hydraulic-path steady-state initialization data.

Hydraulic-path steady-state initialization data are what the TRAC user knows or estimates the steady-state thermal-hydraulic solution will be along each of the 1D flow paths. Each 1D flow path has its entrance and exit mesh-cell interfaces defined where inflow and outflow occur to the path. A known or estimated steady-state phasic-temperature and phasic-velocities flow condition is defined at a single mesh-cell interface anywhere within the 1D flow path (inclusive of its end interfaces). The total and noncondensable-gas pressures may be defined as constant along each 1D flow path or defined by the 1D hydraulic-component data. A significant power source or sink along a subrange of mesh cells within the path also needs to be defined (such as for heat transfer between the primary and secondary sides of a heat exchanger). 1D flow paths begin or end at any mesh-cell interface as long as they are different interfaces and do not overlap internally with the cells of other 1D flow paths. However, 1D flow paths must begin or end at the internal-junction interface of a Tee or Sepd component, at a junction of a Plenum component, or at a source-connection junction of a Vessel component. The internal-junction interface of a Tee or Sepd component and the junction of a Plenum component must define the phasic-temperature and phasic-velocities flow condition of its 1D flow path. Plenum component junctions are assumed to have no steady-state fluid flow if they do not define the end interface of a 1D flow path. However, the fluid flow condition at Vessel-component source-connection junctions may be input specified by hydraulic-component data or initialized by hydraulic-path steady-state initialization data. This provides sufficient information for TRAC internally to initialize the steady-state thermal-hydraulic condition of all 1D hydraulic components along each 1D flow path, as well as all of the Plenum and Vessel components to which such 1D flow paths may be connected.

The 1D hydraulic-component wall and Htstr component Rod or Slab temperature is defined by the input component data and is not initialized by the hydraulic-path steady-state initialization procedure. This also applies to the total and noncondensable-gas pressures unless they are initialized with a constant value for all cells of a 1D flow path. Structure temperatures and coolant pressures need not be initialized accurately because the steady-state calculation quickly determines their steady-state condition consistent with the gas volume fraction, phasic temperatures, and phasic velocities defined by the hydraulic-path steady-state initialization procedure. On the other hand, the gas volume fraction, phasic temperatures, and phasic velocities are the slowest to converge to their steady-state solution and usually require at least three or four convective-flow passes through each 1D flow path to converge to their steady-state values if a significant change is required in the initial thermal-hydraulic solution estimate. Providing a good initial estimate for the gas volume fraction, phasic temperatures, and phasic velocities can significantly reduce the TRAC evaluation time needed to satisfy the steady-state convergence criteria.

Subroutine Init, at the beginning of the TRAC initialization phase, calls Icomp, which calls Ihpss1 to evaluate the hydraulic-path steady-state initialization procedure. Subroutine Ihpss1 performs two passes through all of the 1D hydraulic components. The first pass initializes the gas volume fraction, phasic temperatures, and phasic velocities of all 1D hydraulic components defined by 1D flow paths. The second pass adjusts the gas volume fraction and phasic velocities donored from two-phase fluid mesh cells. This is done to conserve the mesh-cell interface fluid mass flow and the total input fluid-mass inventory of all hydraulically coupled 1D components to which two-phase mesh cells are a part. Both of these solution-estimate initialization passes are performed by Ihpss1 before the regular two-pass initialization is evaluated by Icomp for all 1D hydraulic components.

Each 1D hydraulic component is considered separately during the first pass of the solution-estimate initialization. For each component, while considering all of its hydraulically coupled neighboring 1D hydraulic components, a search of all the Npaths input-defined 1D flow paths is performed to determine if part or all of the component's cells lie within any of the 1D flow paths. When such a path is found, the path's defined thermal-hydraulic flow condition is determined to be either in the component or in a hydraulically coupled neighboring component. The latter option requires that the thermal-hydraulic flow condition be moved to one of the junction interfaces of the component. Moving the thermal-hydraulic flow condition to a junction interface of the component of interest requires incorporating fluid mass-flow and energy-flow sources and sinks to each Jcell from Tee or Sepd component internal junctions along the way, as well as incorporating power sources and sinks to all mesh cells along the way. Fluid mass flow, energy flow, and power sources and sinks also are incorporated when moving the thermal-hydraulic flow condition from the component junction to interfaces within the component in the process of defining the interface mass flow and cell energy throughout the component. The phasic temperatures then are determined from the cell energy, the phasic densities from the phasic temperatures, and the phasic velocities from the phasic densities and interface mass flow and area.

The gas volume fraction α for two-phase fluid lies between α_m and α_n, where $\alpha = \alpha_m$ and $V_G = V_L \cdot (\rho_L/\rho_G)$, assuming no interfacial drag, and $\alpha = \alpha_n$ and $V_G = V_L$, assuming infinite interfacial drag. During the first pass of the initialization procedure, four fluid masses are summed over each region of hydraulically coupled 1D components: (1) the MASI fluid mass input by the 1D hydraulic-component data; (2) the MAST fluid mass based on $\alpha = 0$, α_n, or 1 and T_G and T_L initialized; (3) the MASM fluid mass based on $\alpha = 0$, α_m, or 1 and T_G and T_L initialized; and (4) the MASN fluid mass based on $\alpha = \alpha_m$ and $T_G = T_{sat} = T_L$ initialized. This fluid-mass inventory information is used during the second pass of the initialization procedure to determine the $\alpha_m \le \alpha \le \alpha_n$ gas volume fraction in two-phase mesh cells. This conserves the input-specified fluid mass MASI by defining

$$\alpha = \alpha_n + (\alpha_m - \alpha_n) \bullet f \ ,$$

where

$$f = (MASI–MAST)/(MASM-MASN).$$

See Sec. 2.4.8.1 of the TRAC Theory Manual[2] for further details.

During the first pass of the initialization procedure, the thermal-hydraulic condition of each Plenum-component cell is initialized in Ihpss1. This is based on summing the connecting 1D flow-path fluid mass and energy inflows and outflows. The coolant enthalpy of the Plenum cell is defined by the ratio of the energy inflow to the mass inflow. This ratio should equal the ratio of the energy outflow to the mass outflow. A warning message is issued if the user-specified fluid mass and energy inflows and outflows differ by more than 1%. The summed fluid mass inflows and outflows are constrained to be equal by multiplying the inflows by f and dividing the outflows by f, where

$$f = \sqrt{(outflow)/(inflow)} .$$

The gas volume fraction and phasic temperatures are evaluated based on the cell enthalpy and pressure. Two-phase fluid mass conservation is applied during the second pass of the initialization procedure, as described above. The phasic velocities at each junction are defined by the phasic mass flows divided by the donor-cell phasic volume fraction times the phasic temperature-dependent density.

After the 1D hydraulic-component gas-volume-fraction, phasic-temperature, and phasic-velocity distributions have been initialized from hydraulic-path steady-state initialization data in Ihpss1, their fluid mass and energy flows at all source connections to each Vessel component are evaluated by subroutine Ihpss3 called by Civssl. This determines the boundary conditions for defining mass- and energy-conservation matrix equations for each Vessel component. The net fluid mass flow is required to be zero for all source connections to a Vessel component. A warning message is printed if the summed fluid mass inflows and outflows differ by more than 1%. The summed fluid mass inflows and outflows are constrained to be equal by multiplying the inflows by f and dividing the outflows by f, where $f = \sqrt{(outflow)/(inflow)}$. The difference between the summed energy outflows and energy inflows is defined to be the power generated in the core region of the Vessel component. When a core region is not defined (when ICRU = 0 and ICRL = 0), this power is assumed to be generated over the entire Vessel component. The power volumetric generation rate is assumed to be constant in the structure material of each cell of the core region or Vessel [in the 1–FRVOL(I,J,K) fraction of the cell volume].

The mass-conservation matrix equation is derived from a simplified form of the phasic motion equations for liquid and gas. This is done to determine a 3D velocity distribution that satisfies the source-connection fluid mass-flow boundary condition, conserves net mass flow in each Vessel cell, and reasonably approximates the fluid flow pattern in the

Vessel component. The temporal, momentum-convection, and gravity terms in the motion equations are assumed to be zero, and the liquid and gas velocity distributions are assumed to be the same. This simplifies the phasic motion equations to a single approximate motion equation for each interface between cells and defines the cell-interface fluid mass flow by

$$\dot{m} = -\Delta P \cdot A/R \quad .$$

The cell-interface flow resistance is approximated by the sum of the form-loss and a flow-length drag resistance. Laminar flow rather than turbulent flow is assumed so that the cell-interface flow resistance is a constant rather than proportional to the magnitude of the interface fluid velocity. The mass-conservation matrix equation requires that the net coolant mass flow of each Vessel cell be zero:

$$\left[\sum_{out}\dot{m}_i - \sum_{in}\dot{m}_i\right]_n = \left[\sum_{insoucon}\dot{m} - \sum_{outsoucon}\dot{m}\right]_n = = \left[\sum_{soucon}\dot{m}\right]_n \quad .$$

Substituting the simplified motion-equation definition of fluid mass flow between Vessel cells gives

$$\sum_{ithface}\frac{A_i}{R_i}\cdot(P_n - P_i) = \left[\sum_{soucon}\dot{m}\right]_n \quad ,$$

where P_n is the coolant pressure in Vessel cell n and Pi is the fluid pressure in neighboring cell i on the other side of interface i. For a Vessel component with a total of N cells, this is the Nth-order mass-conservation matrix equation

$$\bar{M} \cdot \underline{P} = \underline{S} \quad .$$

Because of fluid mass conservation over the entire Vessel component, as well as in each of its N cells, the matrix equation has only N-1 unique equations and N pressure unknowns. The Nth cell equation is replaced with the pressure-normalization requirement $P_N = 10^6$ (in Pa units) to provide N unique equations. This arbitrary normalization of the evaluated pressure distribution is done because the pressure difference between cells, which determines the fluid mass flow across a cell interface, is the result of interest being solved for. TRAC also checks that each Vessel cell has mass-flow coupling to at least one of its six neighboring cells. Without such coupling, the pressure solution in an isolated cell is arbitrary, causing the matrix solution to fail.

Vessel cell-interface fluid mass flows are determined by $\dot{m} = -\Delta P \cdot A/R$ based on the above pressure-distribution solution. Evaluating their fluid velocities requires defining and solving the fluid energy-conservation matrix equation to determine the gas void fraction and phasic temperatures. Then the fluid density can be evaluated and the fluid velocity determined from the fluid mass flow and donor-cell fluid density.

The energy donored by the fluid mass flow is the product of the fluid mass flow and enthalpy

$$h_n \cdot (T_n) = e_n \cdot (T_n) + P_n/\rho_n \cdot (T_n) \ .$$

The fluid energy conservation equation for each Vessel cell n is

$$\left[\sum_{out} \dot{m}_i \cdot h_n - \sum_{in} \dot{m}_i \cdot {}_i\right]_n = \left[\sum_{insoucon} \dot{m} \cdot h - \sum_{outsoucon} \dot{m} \cdot h\right]_n + PowDen \cdot VolStr_n \ ,$$

where

$$PowDen = \left[\sum_{insoucon} \dot{m} \cdot h - \sum_{outsoucon} \dot{m} \cdot h\right]_{vessel} / \left(\max \cdot \left(10^{-10}, \sum\right)\right)$$

if cell n is in the core region or Vessel (when ICRU = 0 and ICRL = 0) and is zero otherwise, and

$$VolStr_n = Vol_n \cdot (1 - FRVOL_n) \ ,$$

where Vol_n is the volume of cell n and $FRVOL_n$ is the input-specified fluid volume fraction of cell n.

A Vessel cell with energy flow out only through source connections defines an energy-conservation equation that cannot be solved for the fluid enthalpy of the cell. TRAC avoids this difficulty by using the cell enthalpy to define the outflow energy through source connections. The above equation becomes

$$\left[\sum_{outsoucon} \dot{m} \cdot h_n + \sum_{out} \dot{m}_i \cdot h_n - \sum_{in} \dot{m}_i \cdot h_i\right]_n = \left[\sum_{insoucon} \dot{m} \cdot h\right]_n + PowDen \cdot VolStr_n \ .$$

For a Vessel component with N cells, this is the Nth-order fluid energy-conservation matrix equation

$$\bar{N} \cdot \underline{h} = \underline{E} \ ,$$

which can be solved for the enthalpy distribution. The enthalpy of an isolated Vessel cell with no mass flow coupling to its neighboring cells and without source connections is arbitrary. TRAC avoids this by redefining its matrix-row equation in matrix $\bar{N}$ to require its enthalpy to equal the evaluated enthalpy in the cell above (below, if the cell is in the top level).

The enthalpy-distribution solution is converted to gas-volume-fraction and phasic-temperature distributions in the same manner as for 1D components. Two-phase gas volume fractions are adjusted to conserve the fluid mass inventory input for each Vessel component by the procedure described above for 1D components. Then the donor-cell fluid density is evaluated based on the gas volume fraction and phasic temperatures to evaluate the interface velocity

$$V_i = \dot{m}_i / (\rho \cdot (T, \alpha)_{donorcell} \cdot A_i)$$

from the interface mass flow and flow area.

All of the above evaluations for a Vessel component are performed in subroutine Ihpss3. Array storage for the mass- and energy-conservation matrix equations is that already reserved in the A array for the multiple-Vessel pressure matrix equation that is evaluated during the outer-interative solution. See Sec. 2.4.8.2 of the TRAC Theory Manual[3] for further details concerning the thermal-hydraulic initialization of a Vessel component.

VI.4. The Radiation Model

This model is not currently installed in TRAC-M. Documentation will be provided after its reinstallation.

REFERENCES

1. S. Jolly-Woodruff, J. Mahaffy, P. Giguere, J. Dearing, and B. Boyack, "Software Design Implementation Document for TRAC-M Data Structures," Los Alamos National Laboratory document LA-UR-97-3026 revised (October 1997).

2. N. M. Schnurr, R. G. Steinke, V. Martinez, and J. W. Spore, "TRAC-PF1/MOD2 Code Manual, User's Guide," US Nuclear Regulatory Commission report NUREG/CR-5673 (July 1992).

3. J. W. Spore, S. J. Jolly-Woodruff, T. K. Knight, J-C. Lin, R. A. Nelson, K. O. Pasamehmetoglu, R. G. Steinke, and C. Unal, "TRAC-PF1/MOD2 Theory Manual," Los Alamos National Laboratory document LA-12031-M (draft) (July 21, 1993).

4. R. G. Steinke, V. Martinez, N. M. Schnurr, J. W. Spore, and J. V. Valdez, "TRAC-P User's Guide," Los Alamos National Laboratory document (draft) (August 1996).

APPENDIX A

SUBPROGRAM CALLING TREES

This appendix contains basic calling trees associated with TRAC-M to aid in the understanding of both computational flow and the flow of data through the program. Figure A.1 provides information on the branching at the highest levels of the program. The remaining figures provide basic information on the trees for the key stages of the program and breakouts of the component level trees for these stages where appropriate. These stages, as outlined in Sec. I, and the associated figures are as follows:

1. Input of initial component data (e.g., Rpipe), Fig. A.2;
2. Input of restart information for a component (e.g., Repipe), Fig. A.3;
3. Initialization of component-dependent variables (e.g., Ipipe), Fig. A.4;
4. Prepass(Prep), including the stabilizer momentum equation solution, evaluation of various old-time quantities and other bookkeeping at the beginning of each timestep (e.g., Pipe1), Fig. A.5;
5. Iterative solution of basic flow equations (Hout) for each timestep (e.g., Pipe2), Fig. A.6;
6. Postpass (Post), including the solution of stabilizer mass and energy equations, solution of the conduction equations, and other computations to complete each timestep (e.g., Pipe3), Fig. A.7;
7. Output of data to the restart dump file (e.g., Dpipe), Fig. A.8;
8. Output of data to the XTV graphics files (e.g., Xtvpipe), Fig. A.9;
9. Output of data to the ASCII detailed edit file (e.g., Wpipe), Fig. A.10.

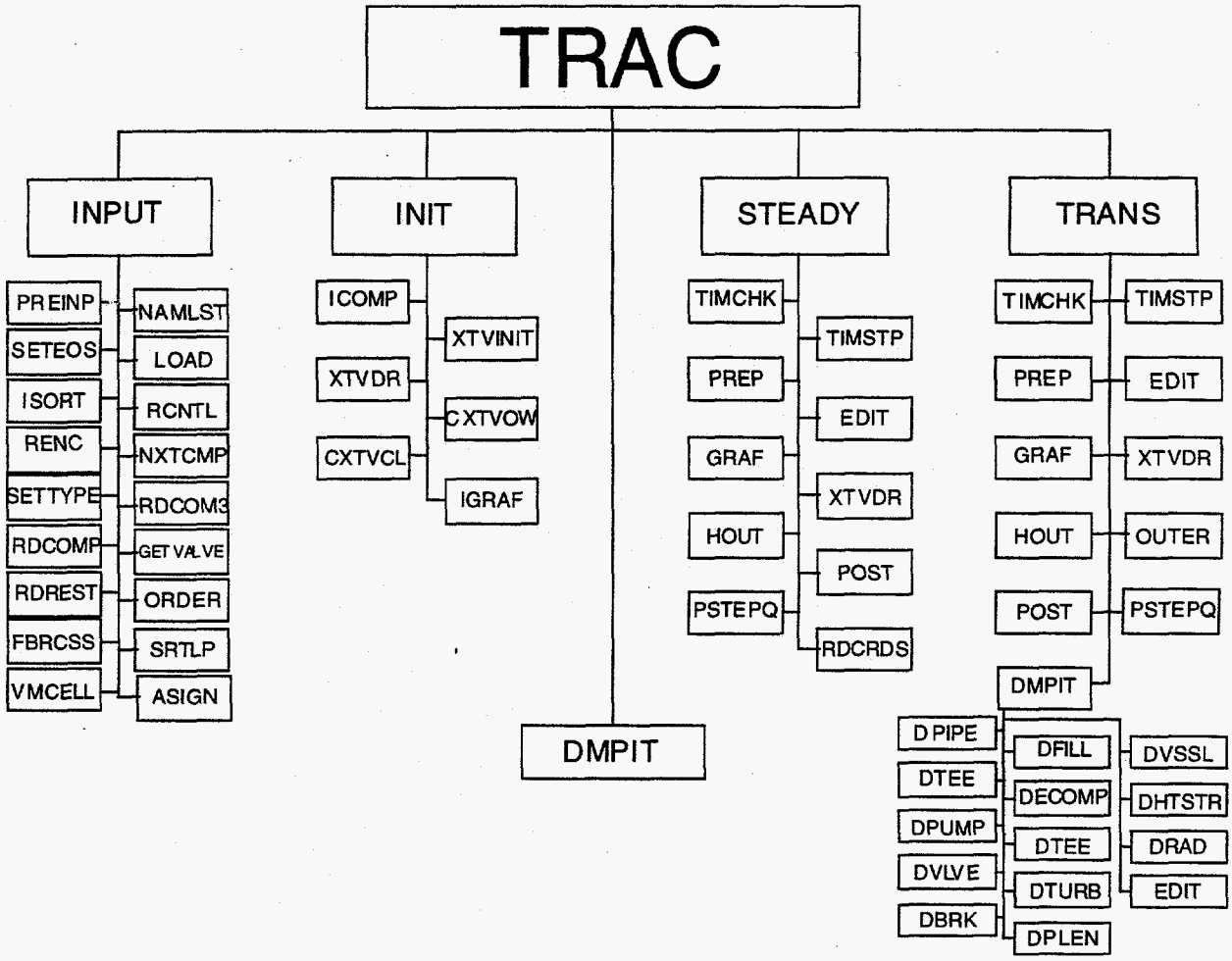


Fig. A.1. Program TRAC calling tree.

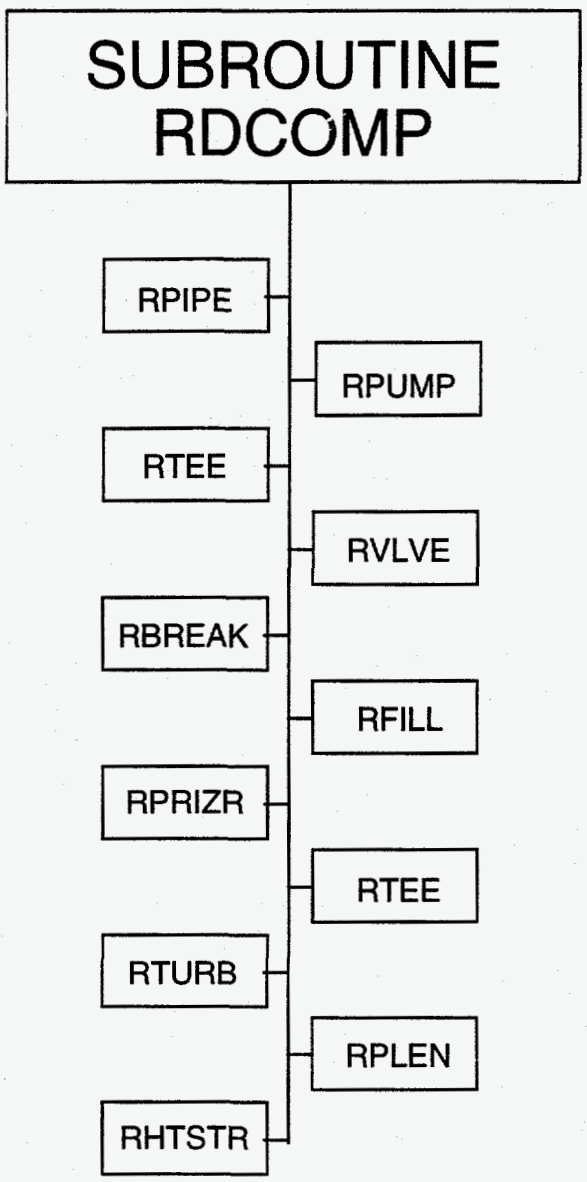


Fig. A.2.a. Top level of input deck processing (Rdcomp).

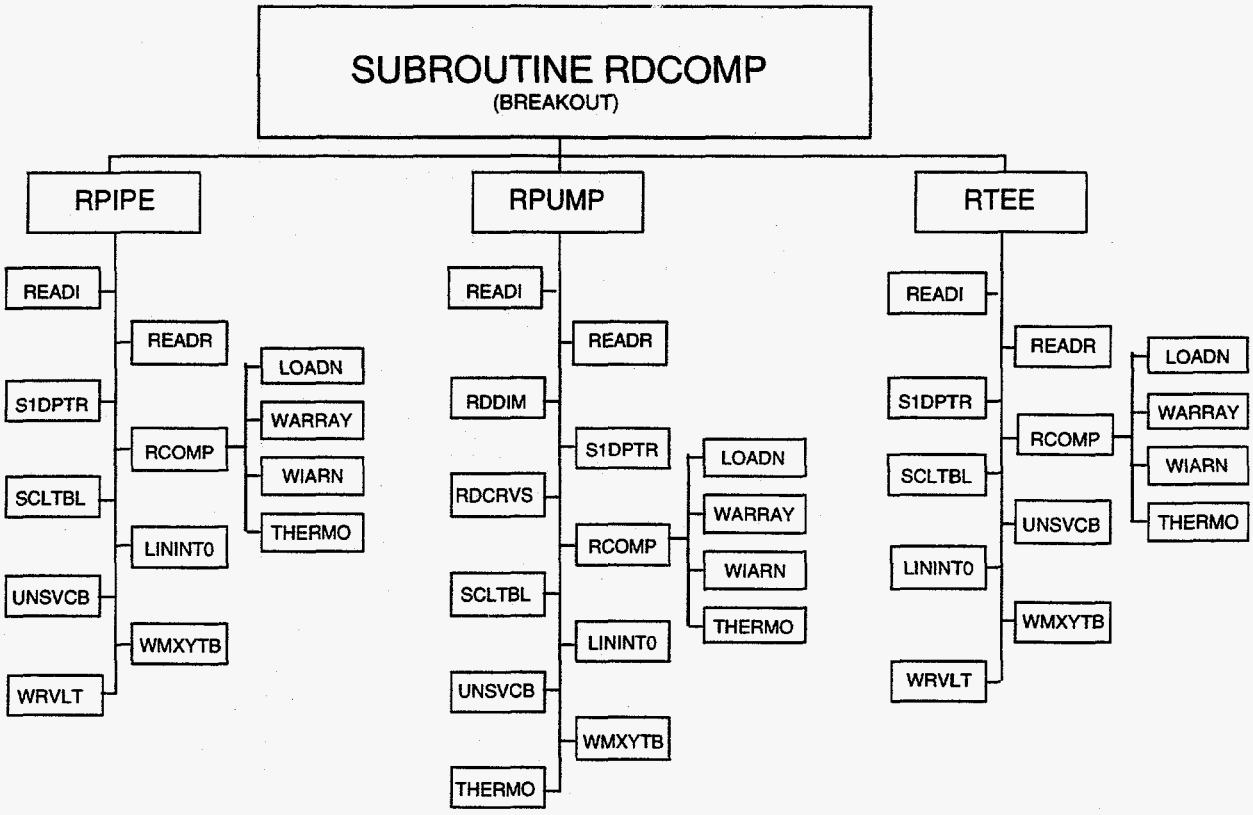


Fig. A.2.b. Breakout of top level of input deck processing (Rdcomp).

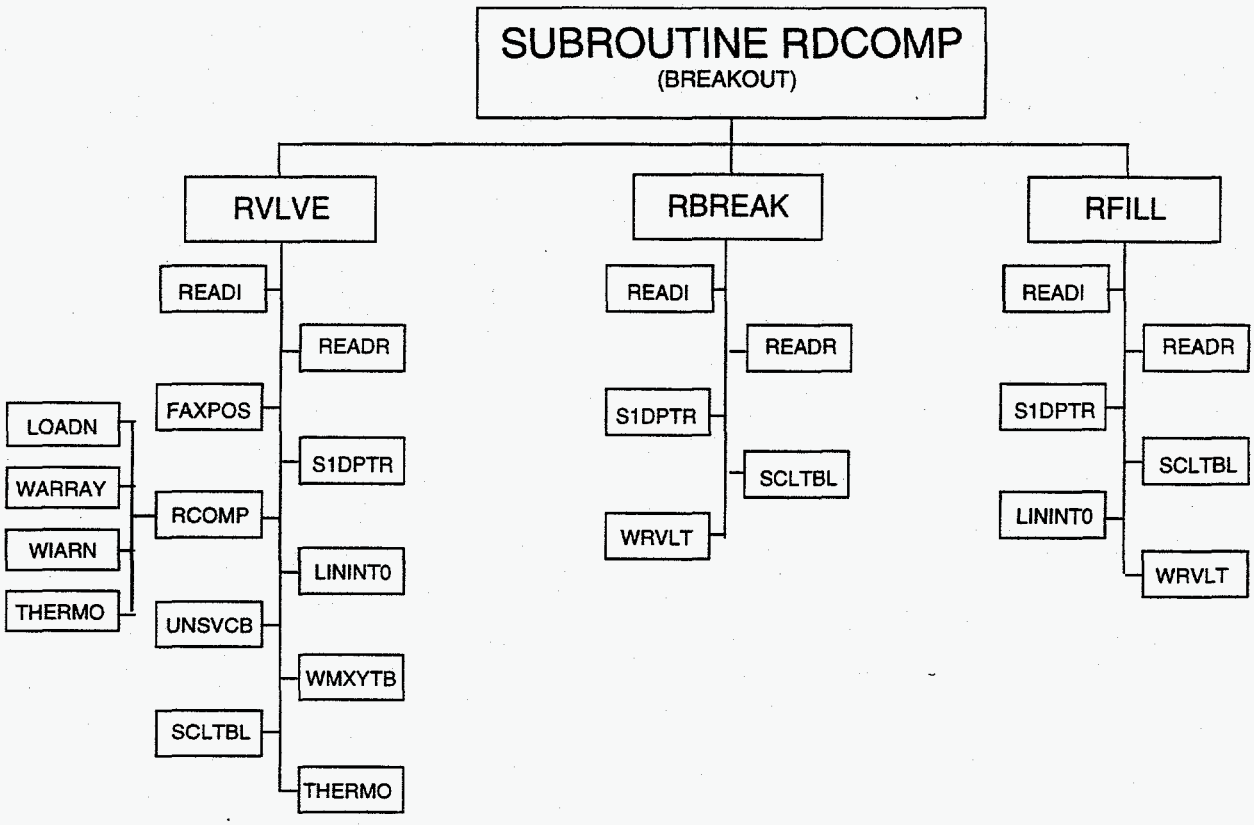


Fig. A.2.b. Breakout of top level of input deck processing (Rdcomp) (cont).

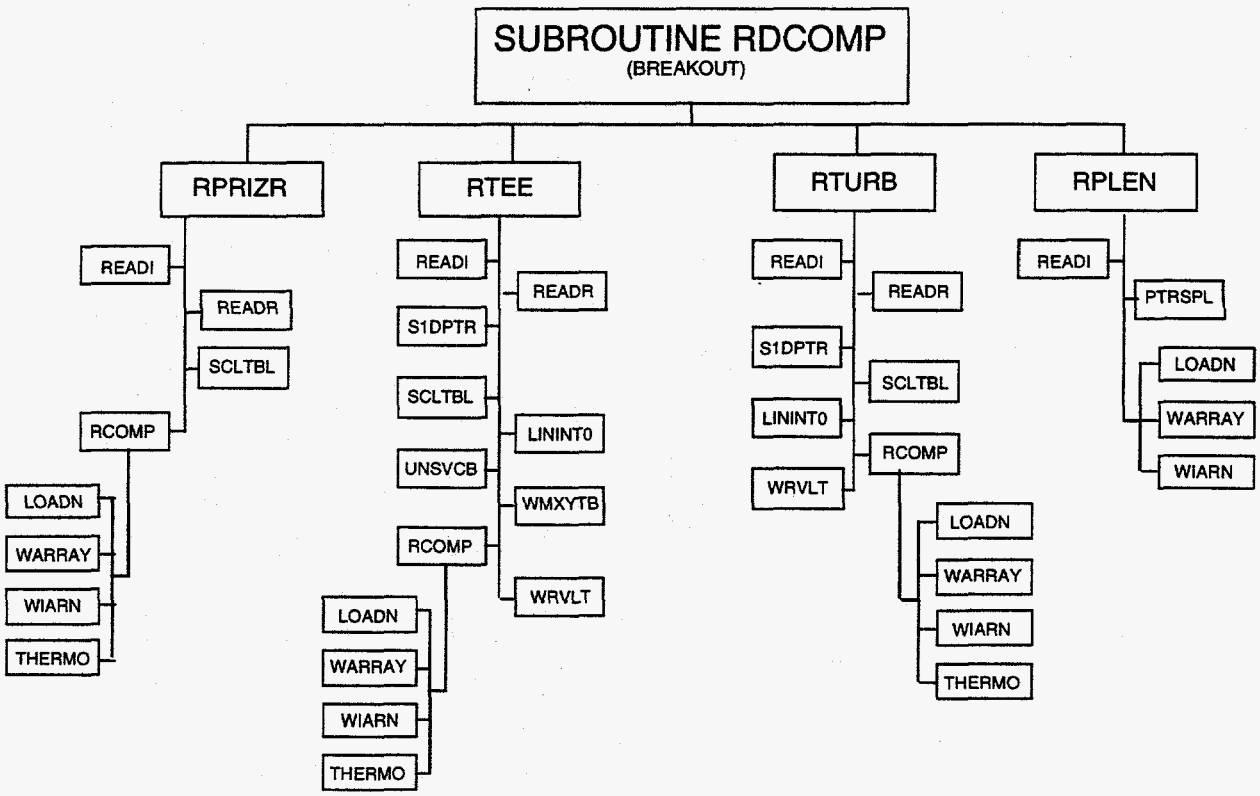


Fig. A.2.b. Breakout of top level of input deck processing (Rdcomp) (cont).

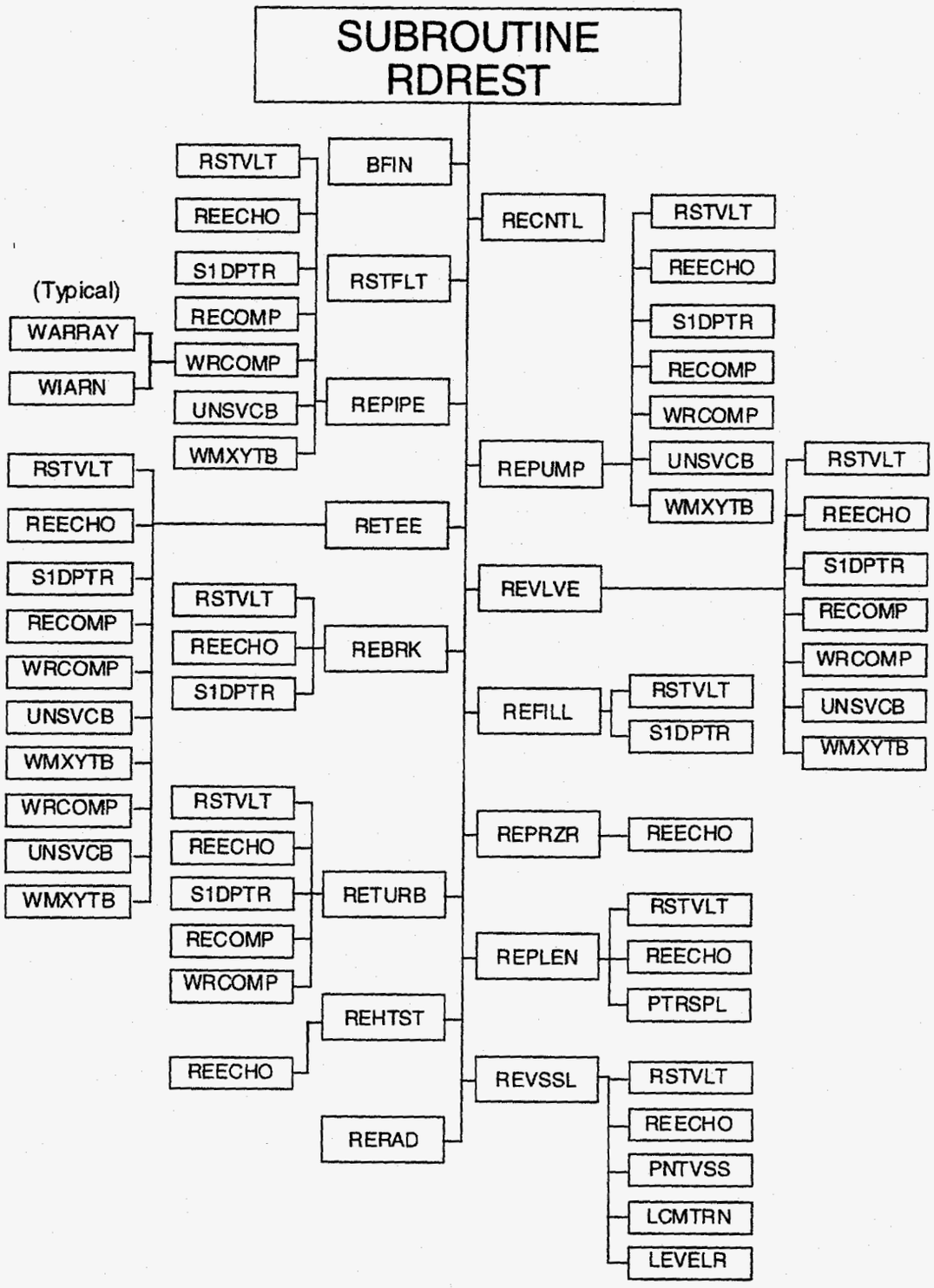


Fig. A.3. Top level of restart input (Rdrest).

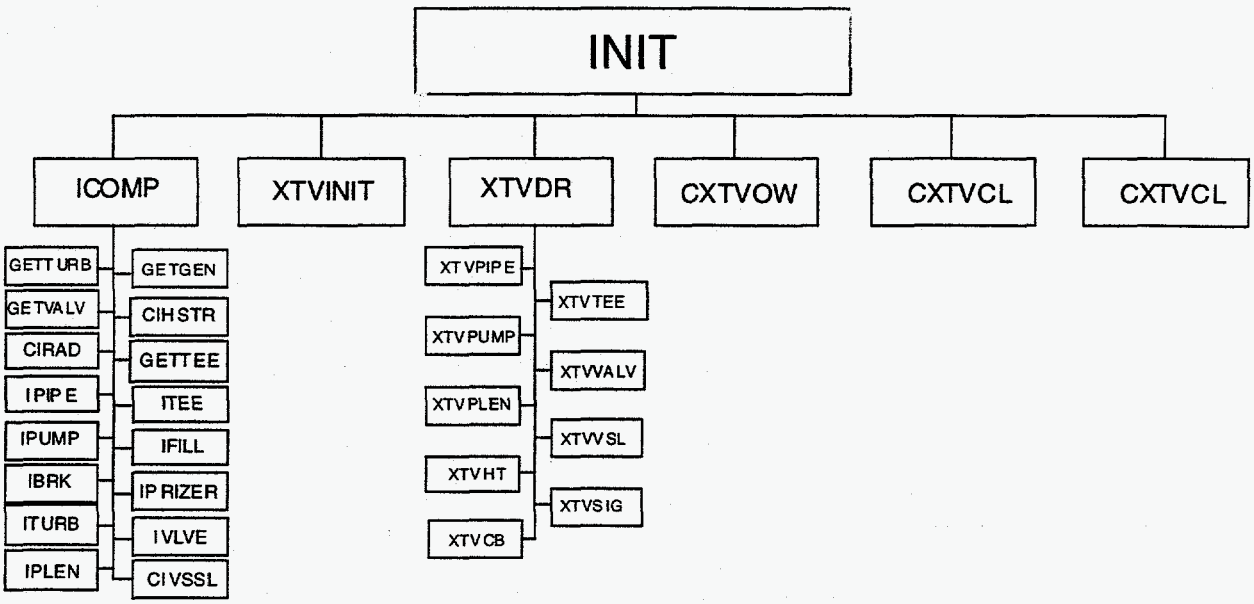


Fig. A.4. Top level of data initialization input (Init).

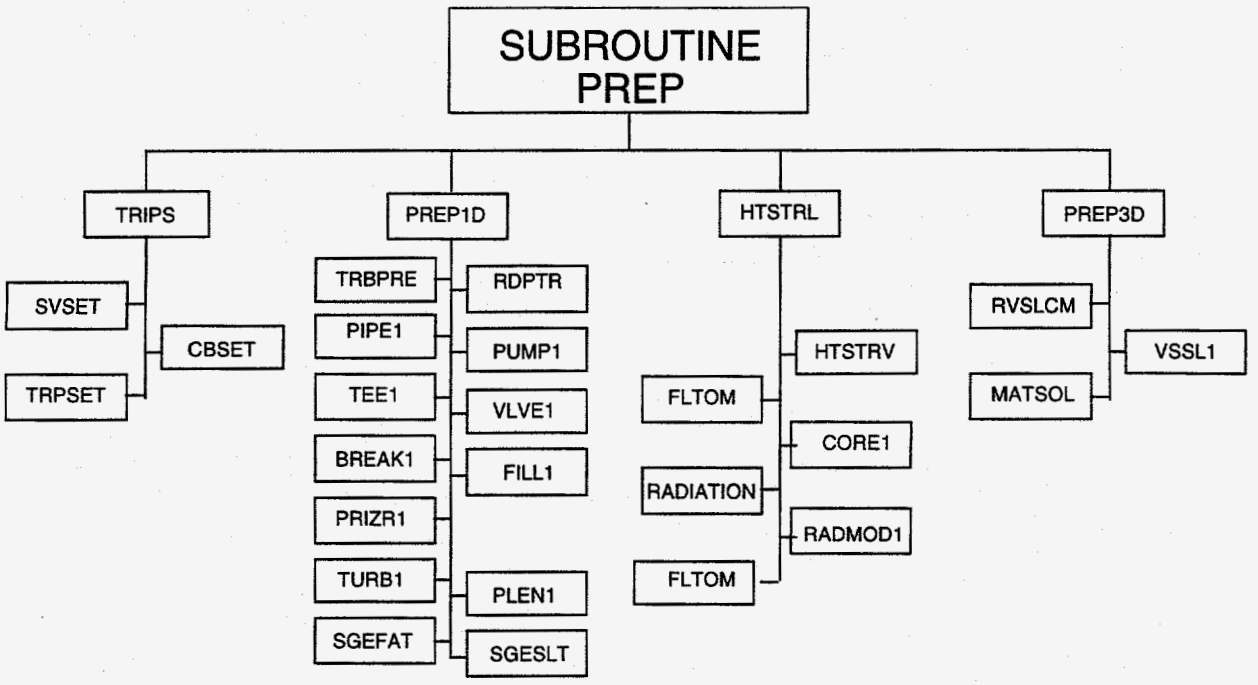


Fig. A.5.a. Top level of solution prepass (Prep).

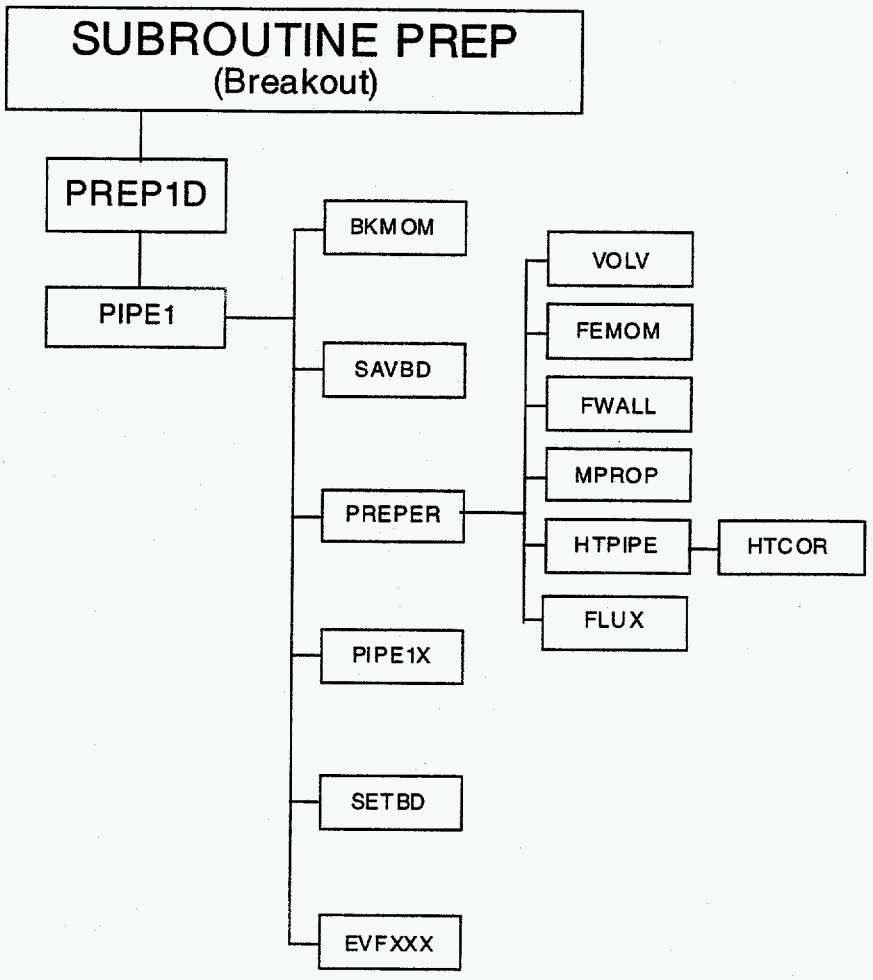


Fig. A.5.b. Breakout of top level of solution prepass (Prep).

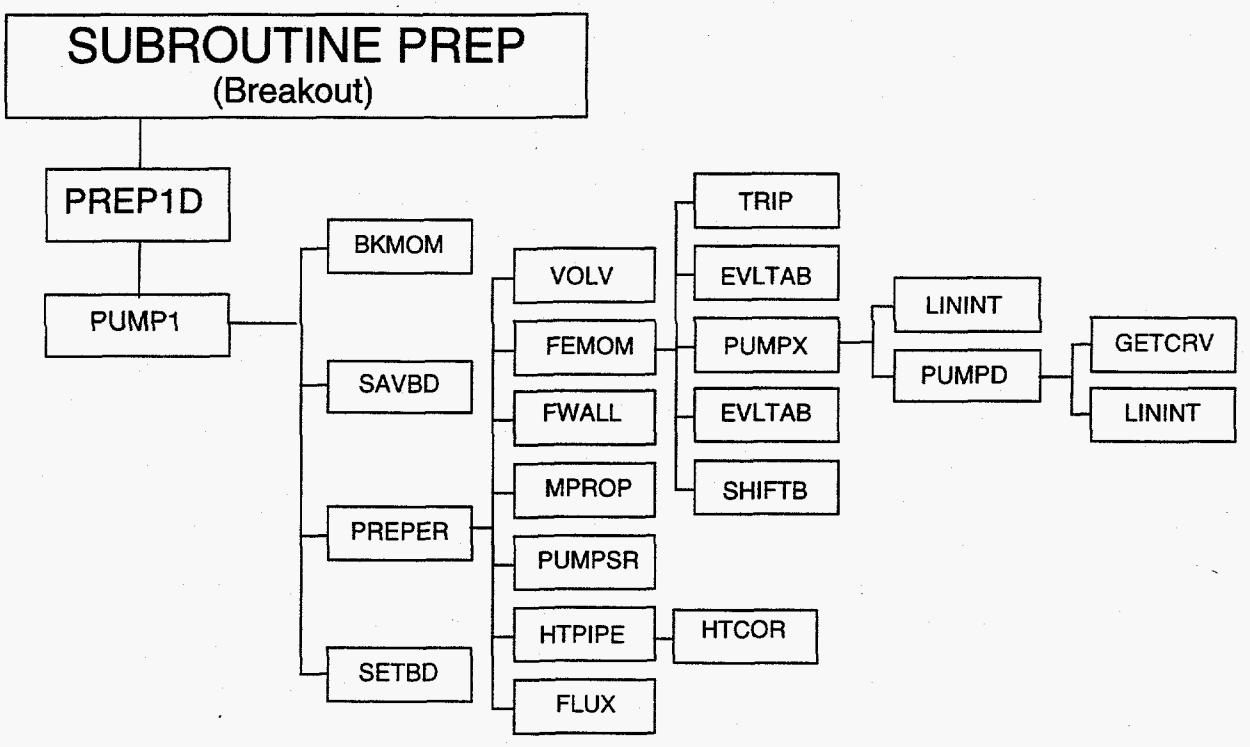


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

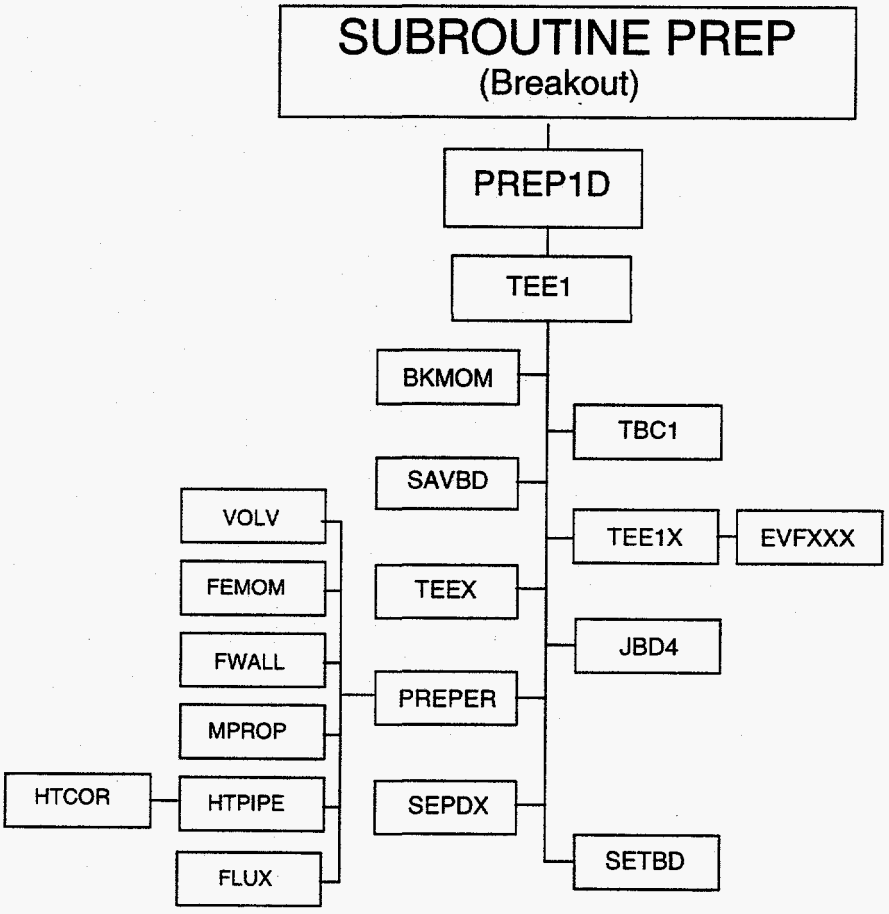


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

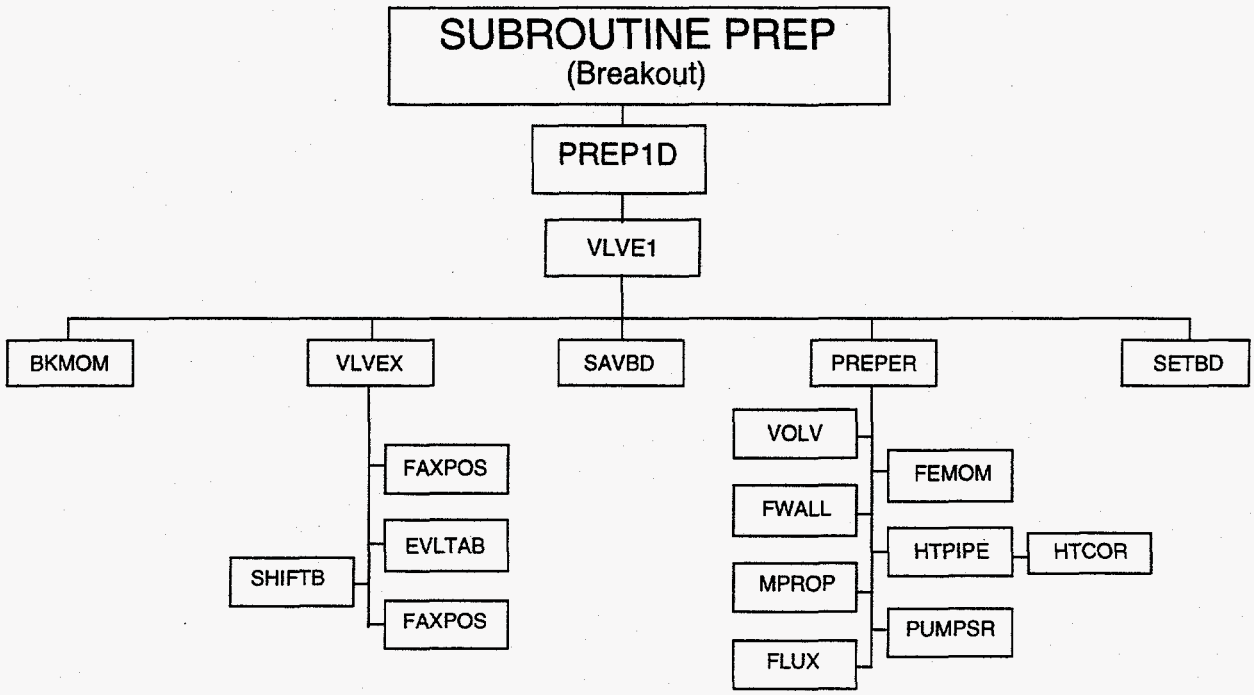


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

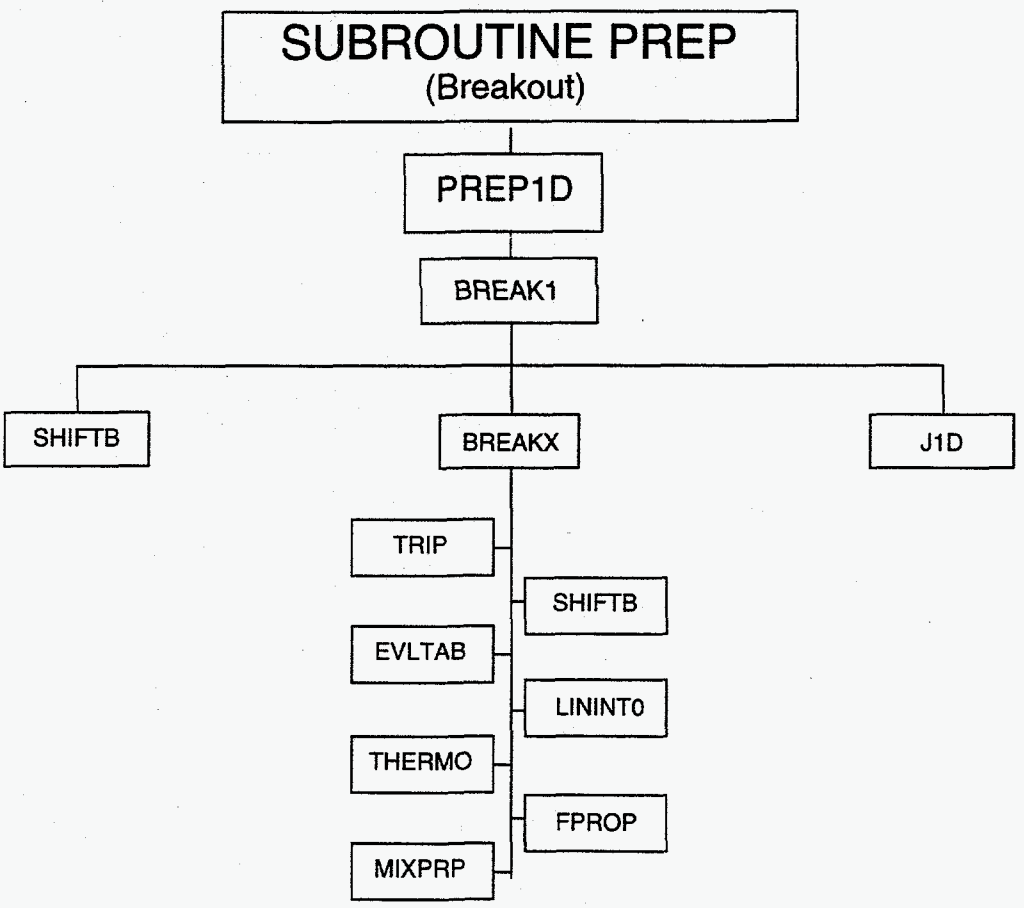


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

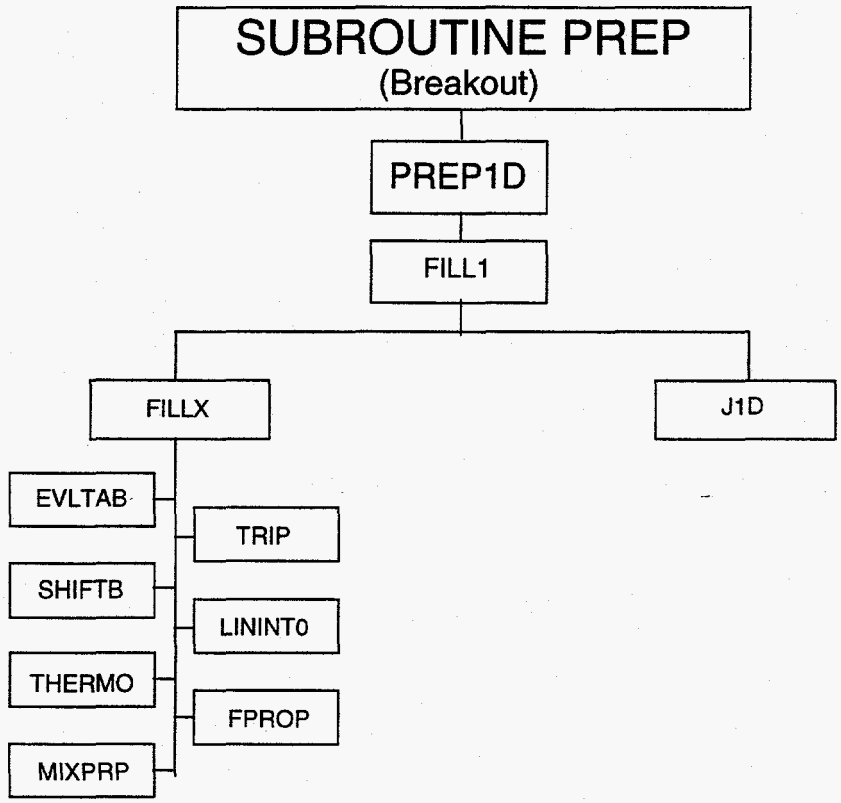


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

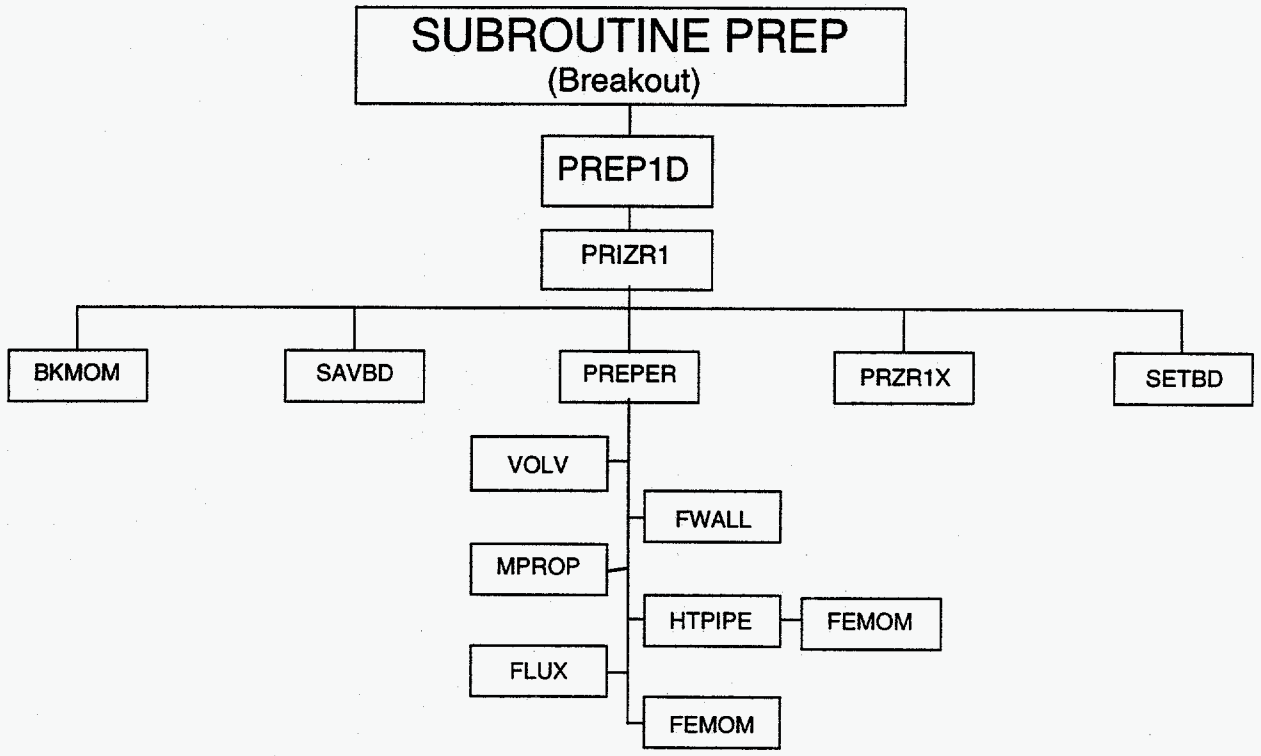


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

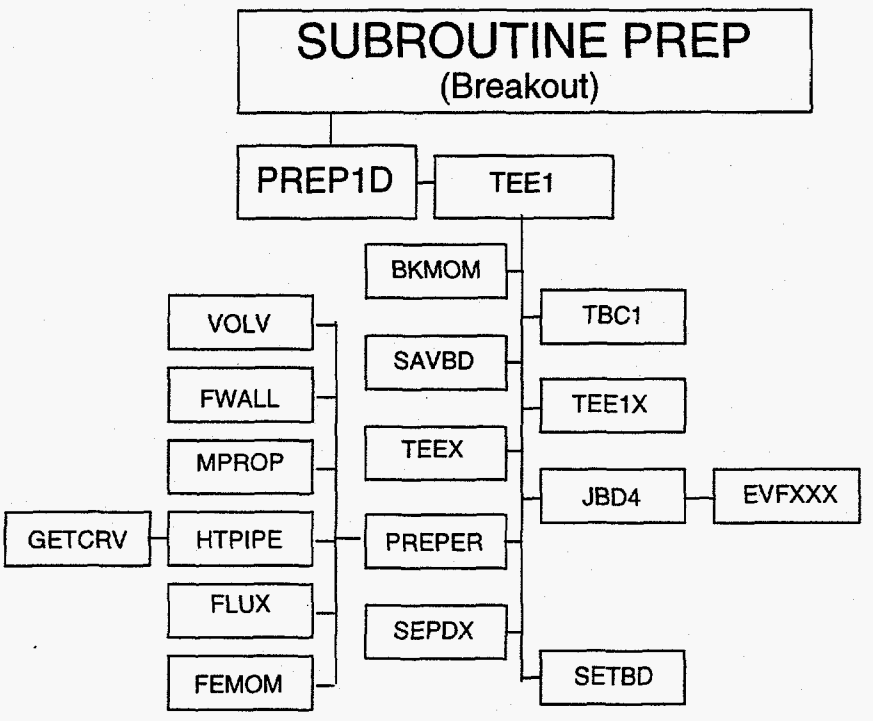


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

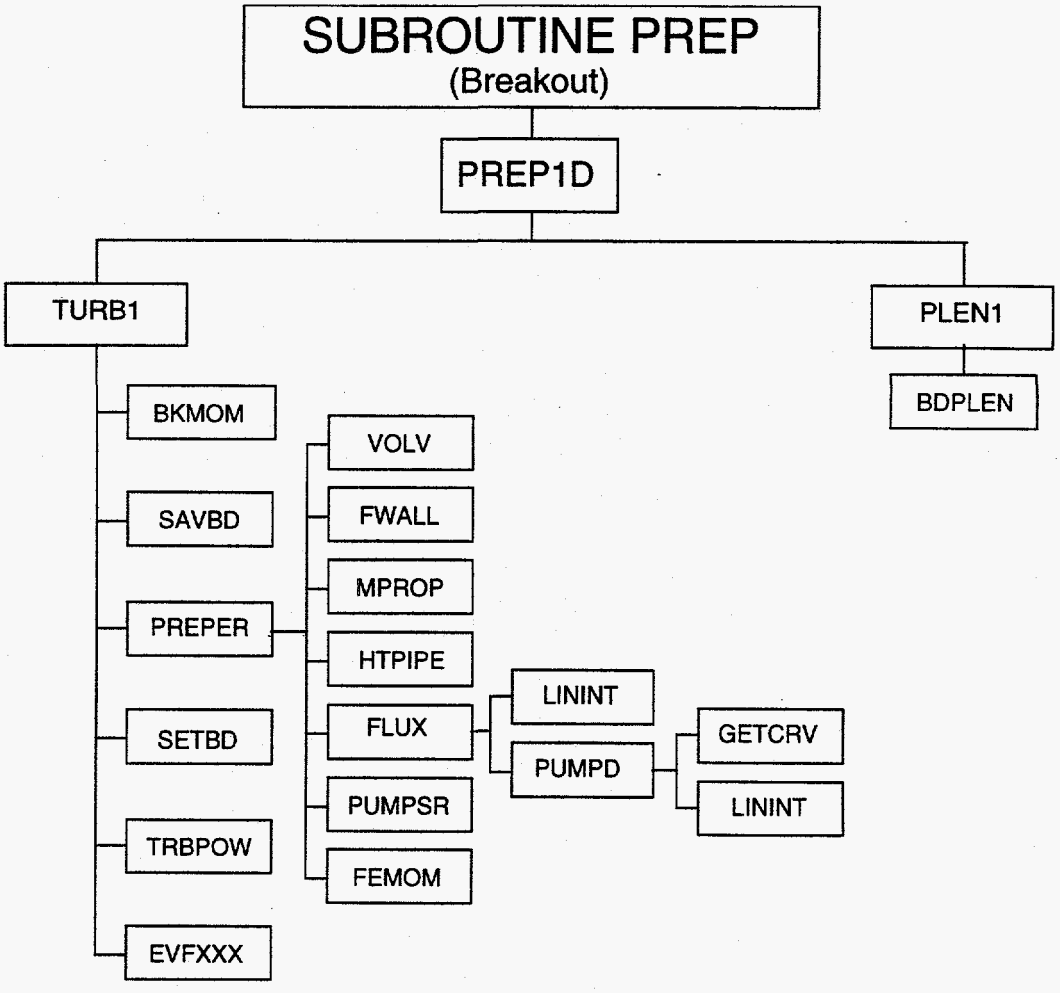


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

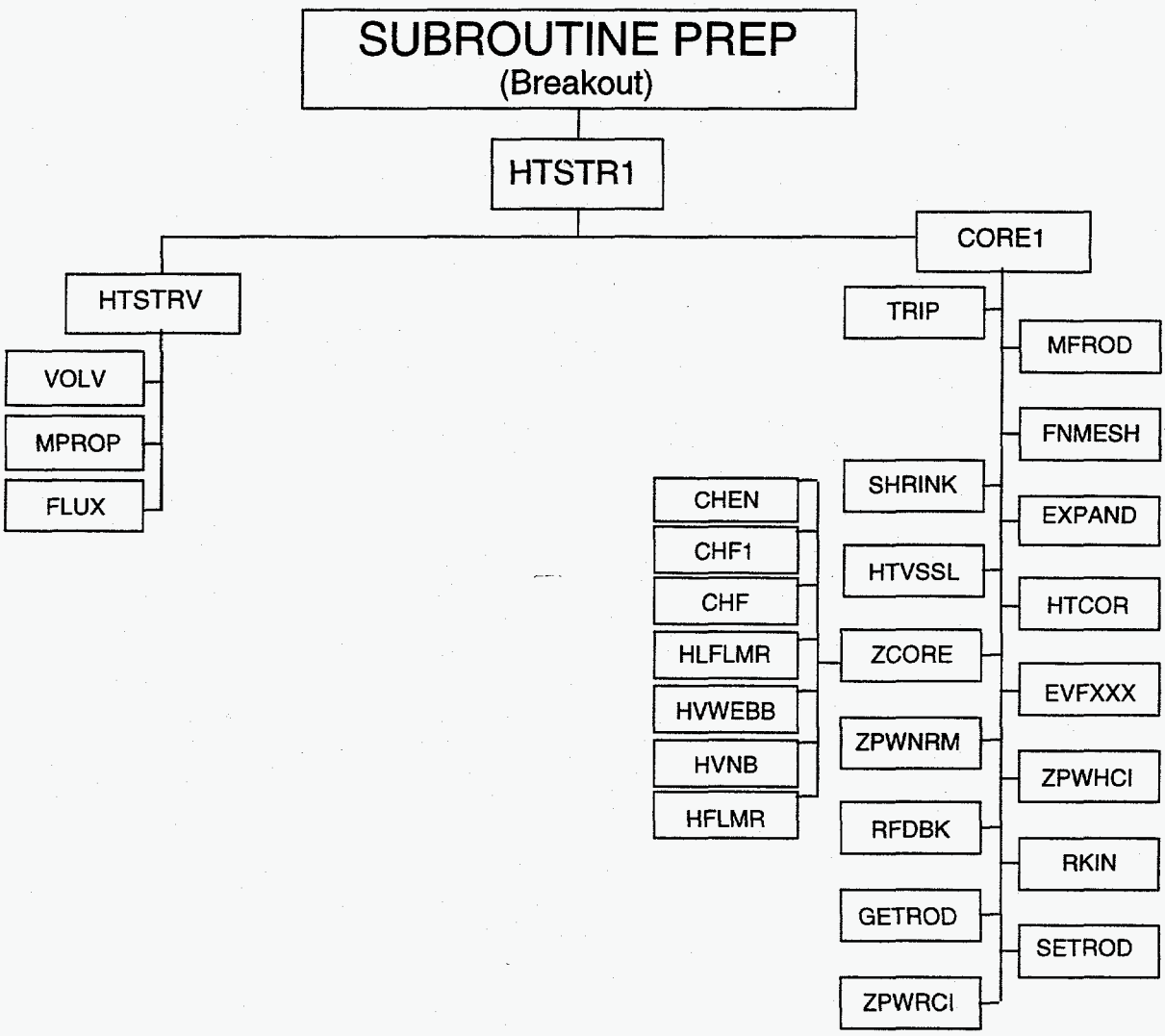


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

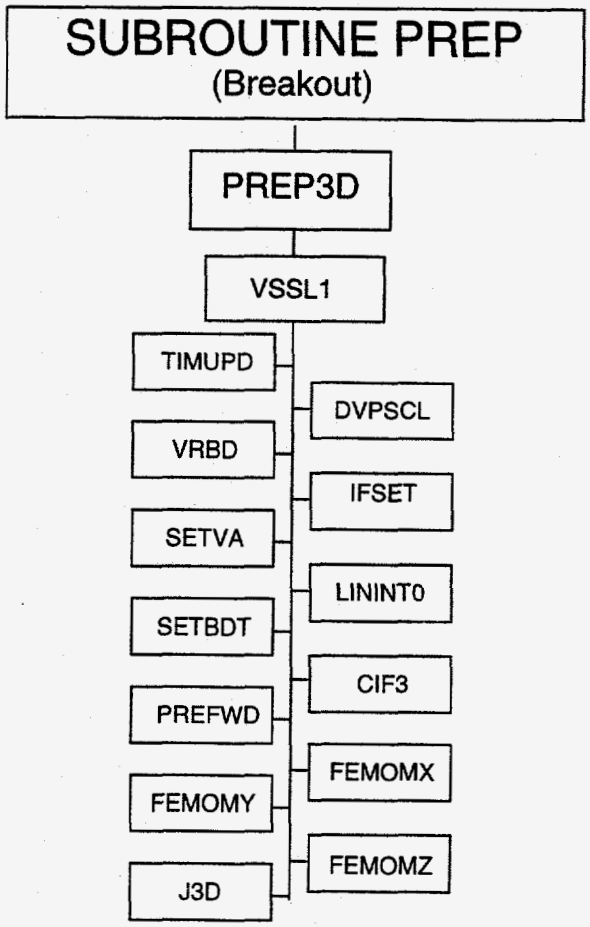


Fig. A.5.b. Breakout of top level of solution prepass (Prep) (cont).

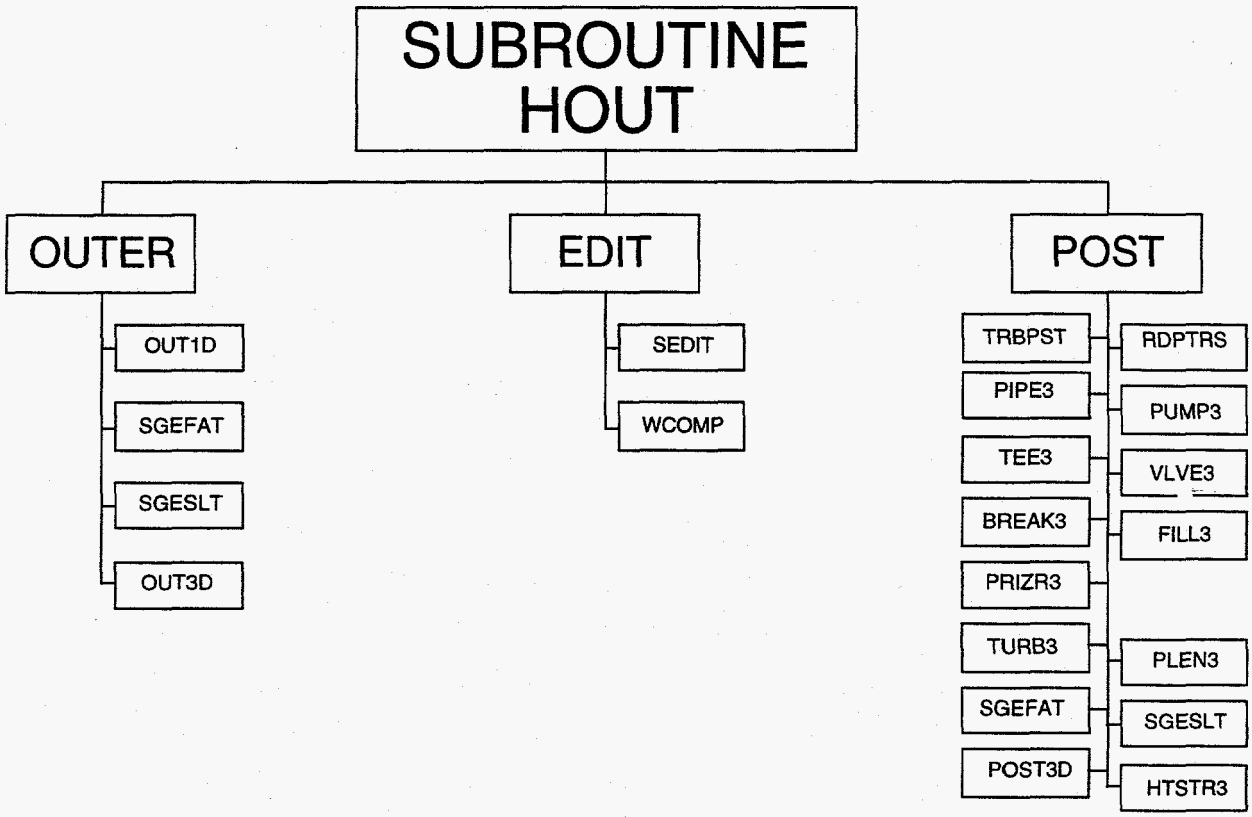


Fig. A.6.a. Top level of solution outer iteration (Hout).

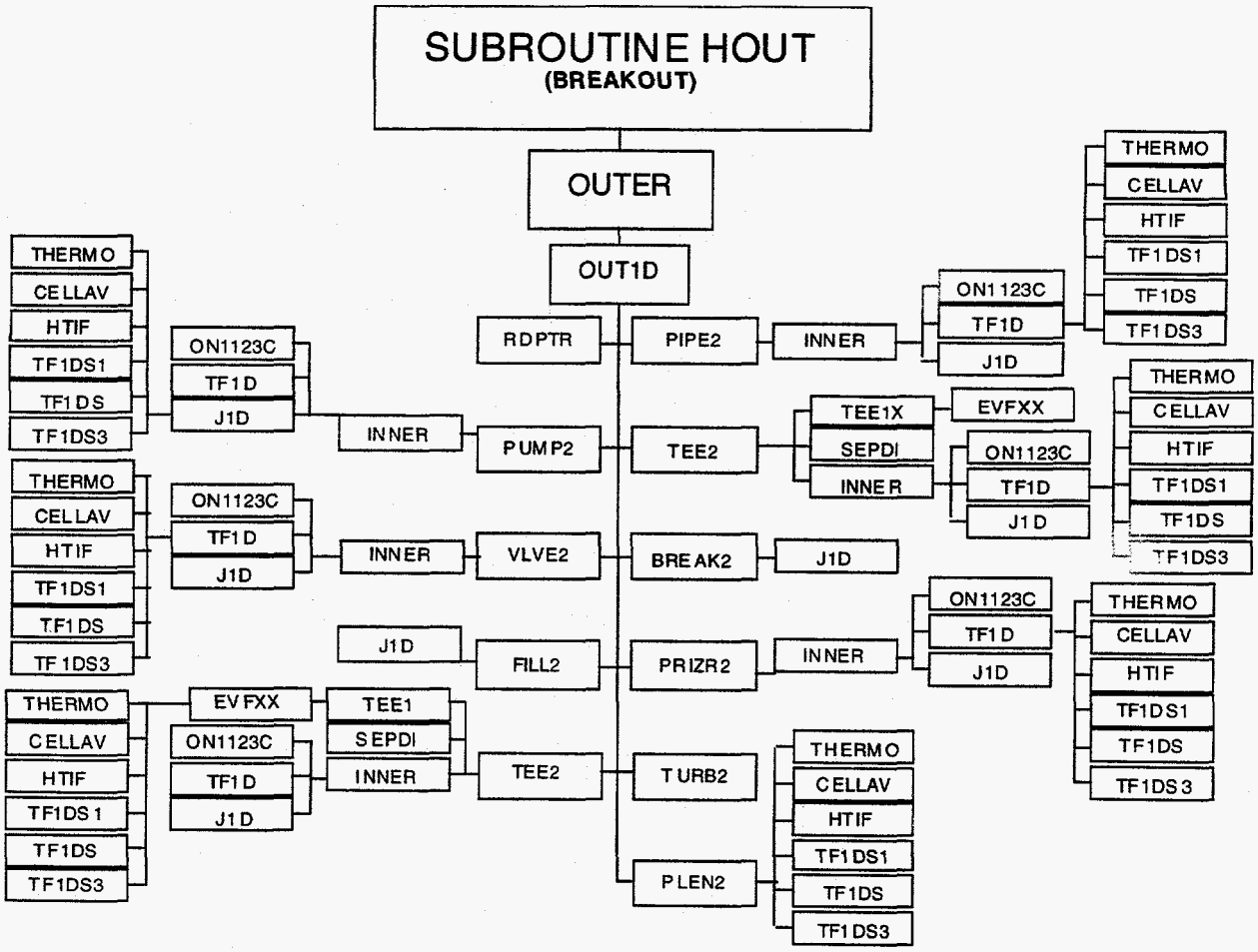


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout).

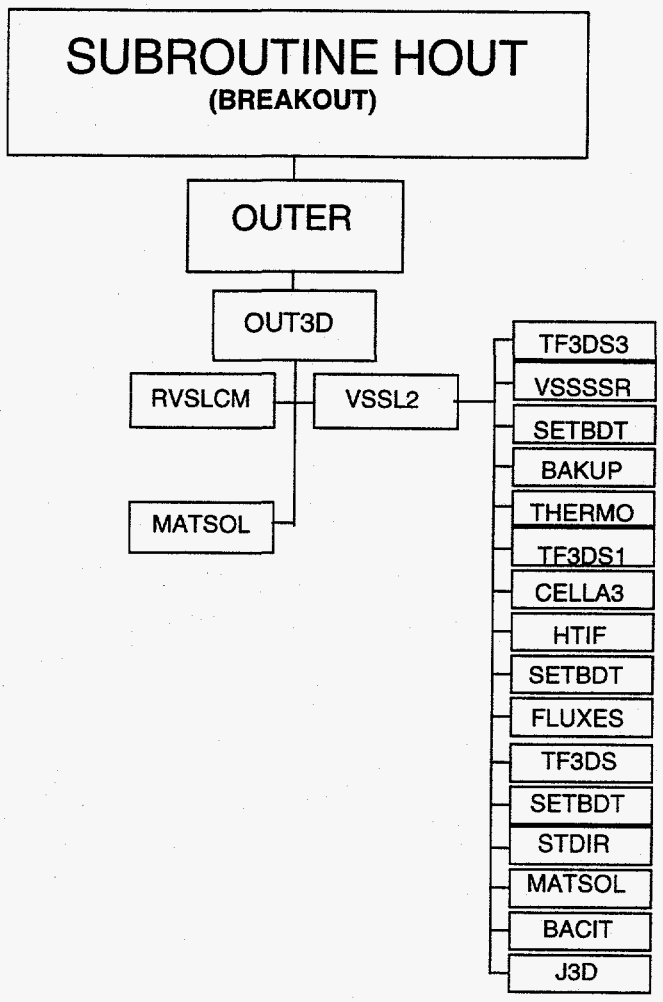


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

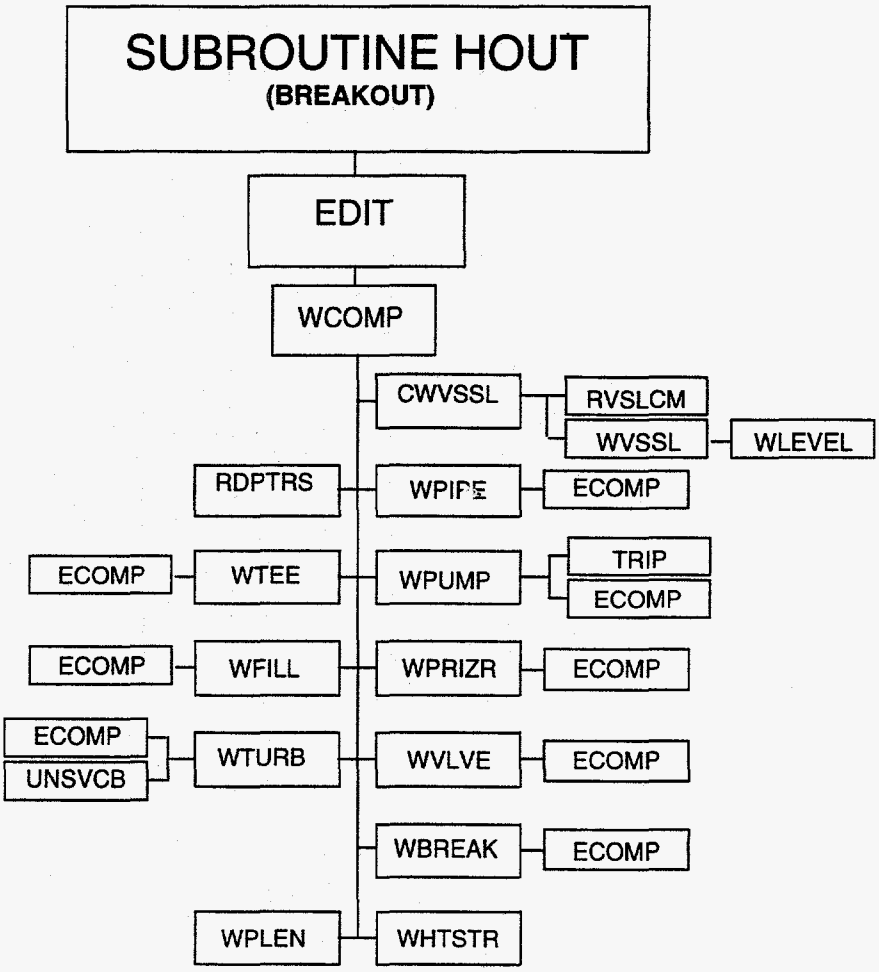


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

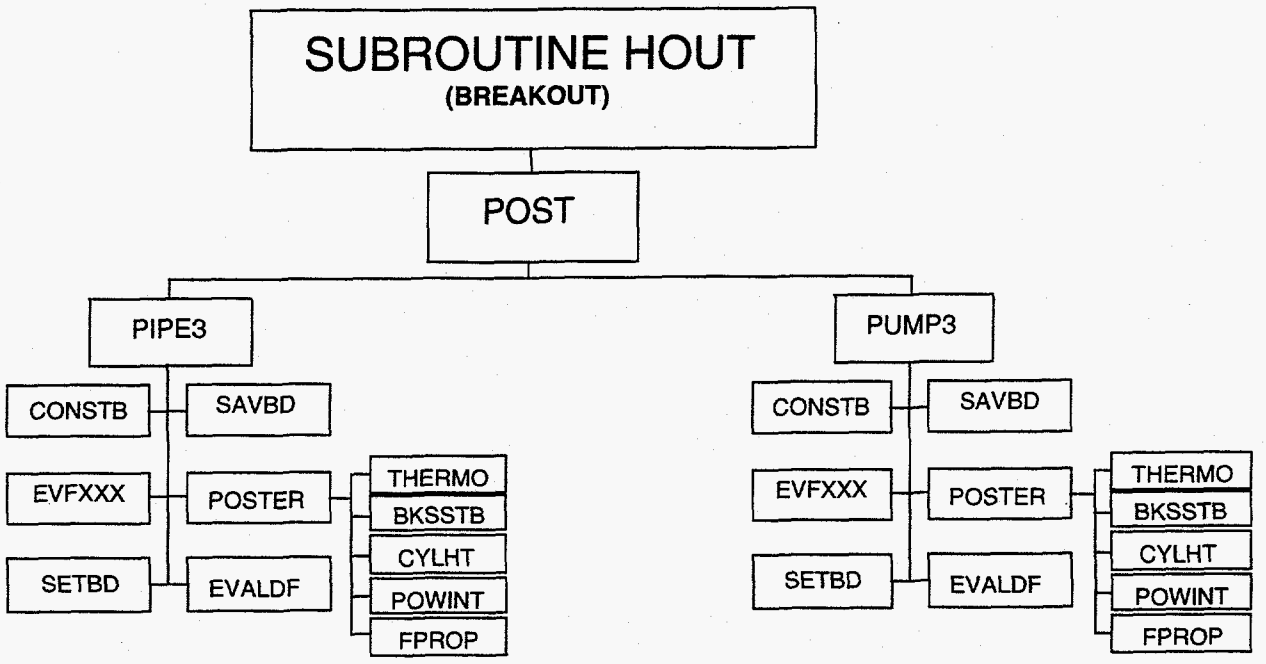


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

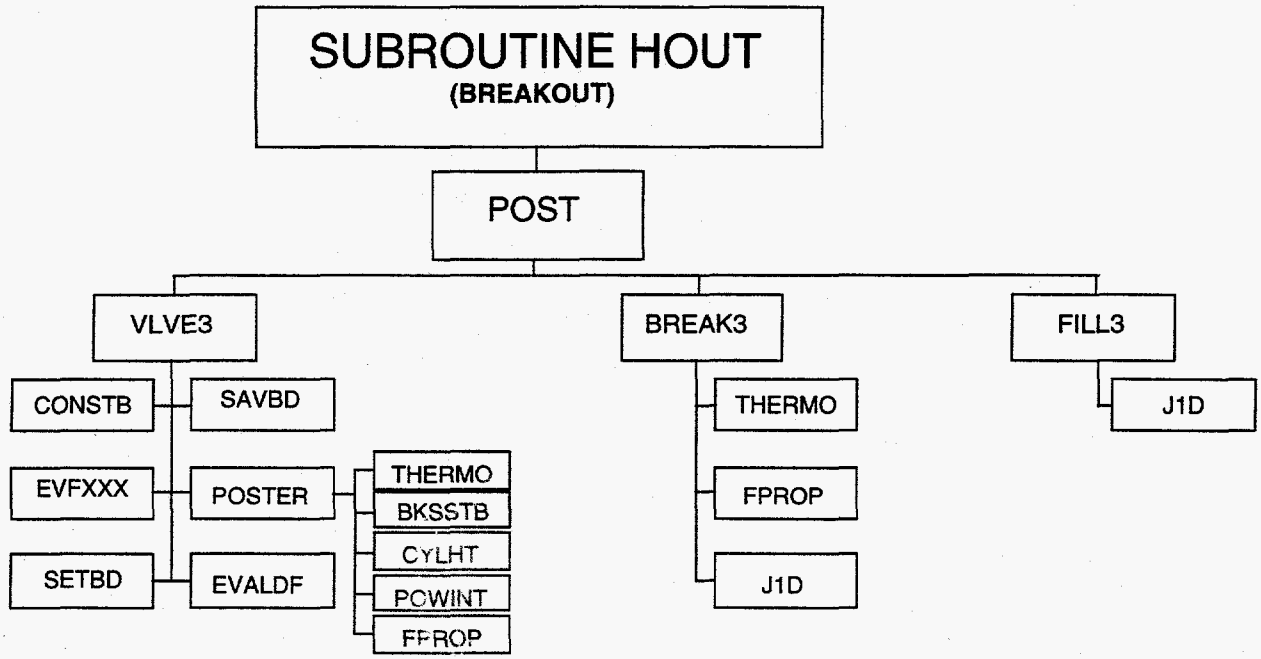


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

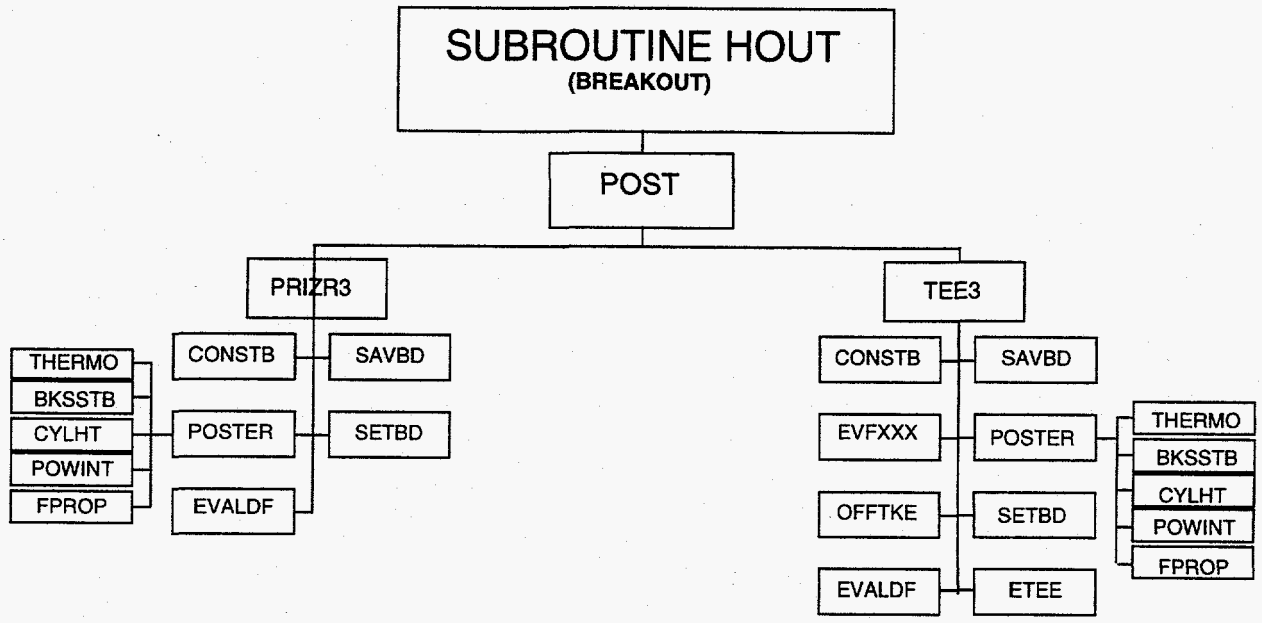


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

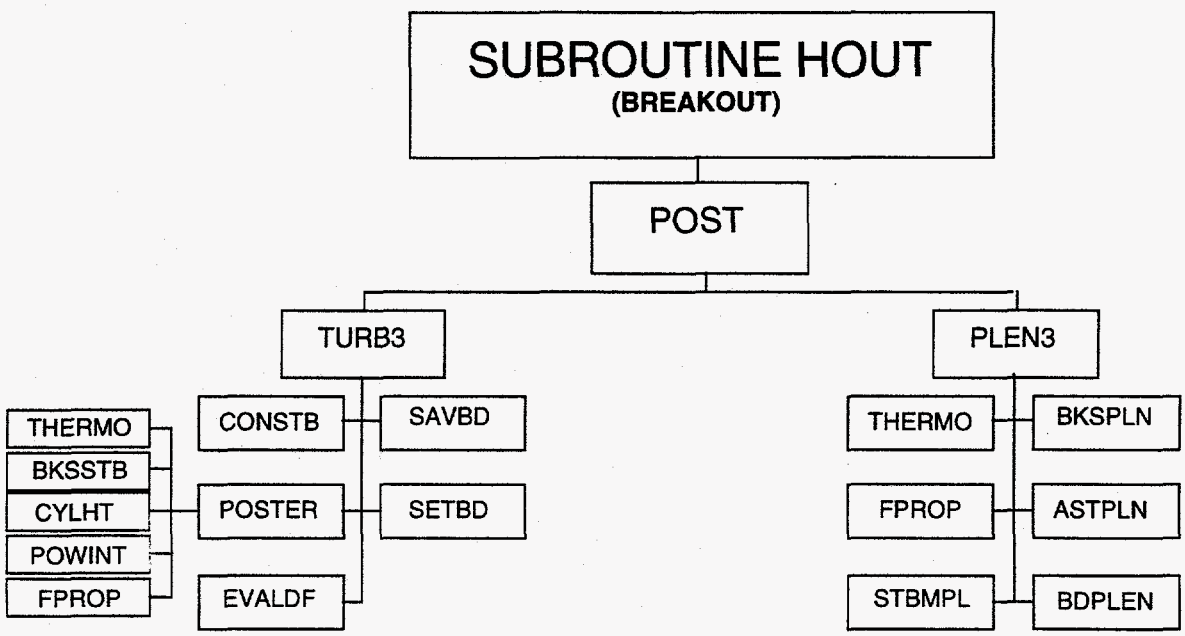


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

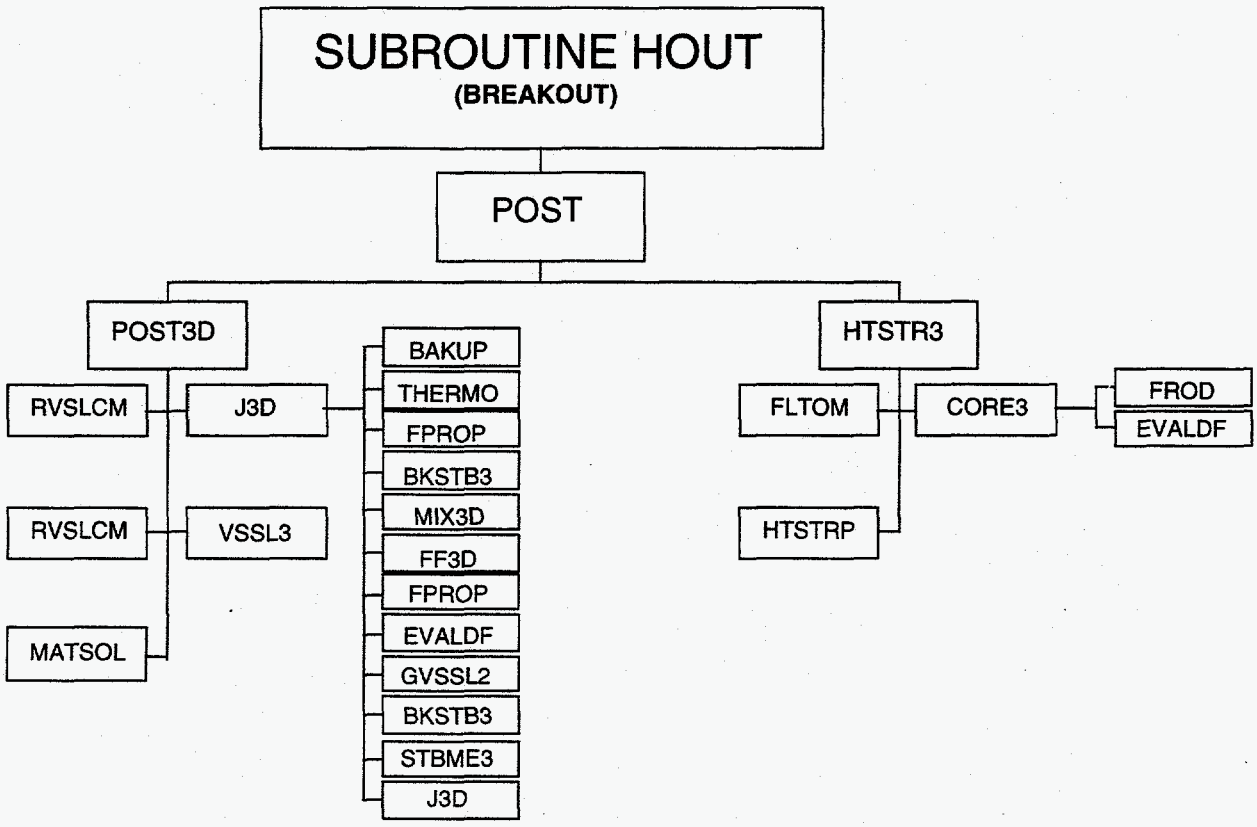


Fig. A.6.b. Breakout of top level of solution outer iteration (Hout) (cont).

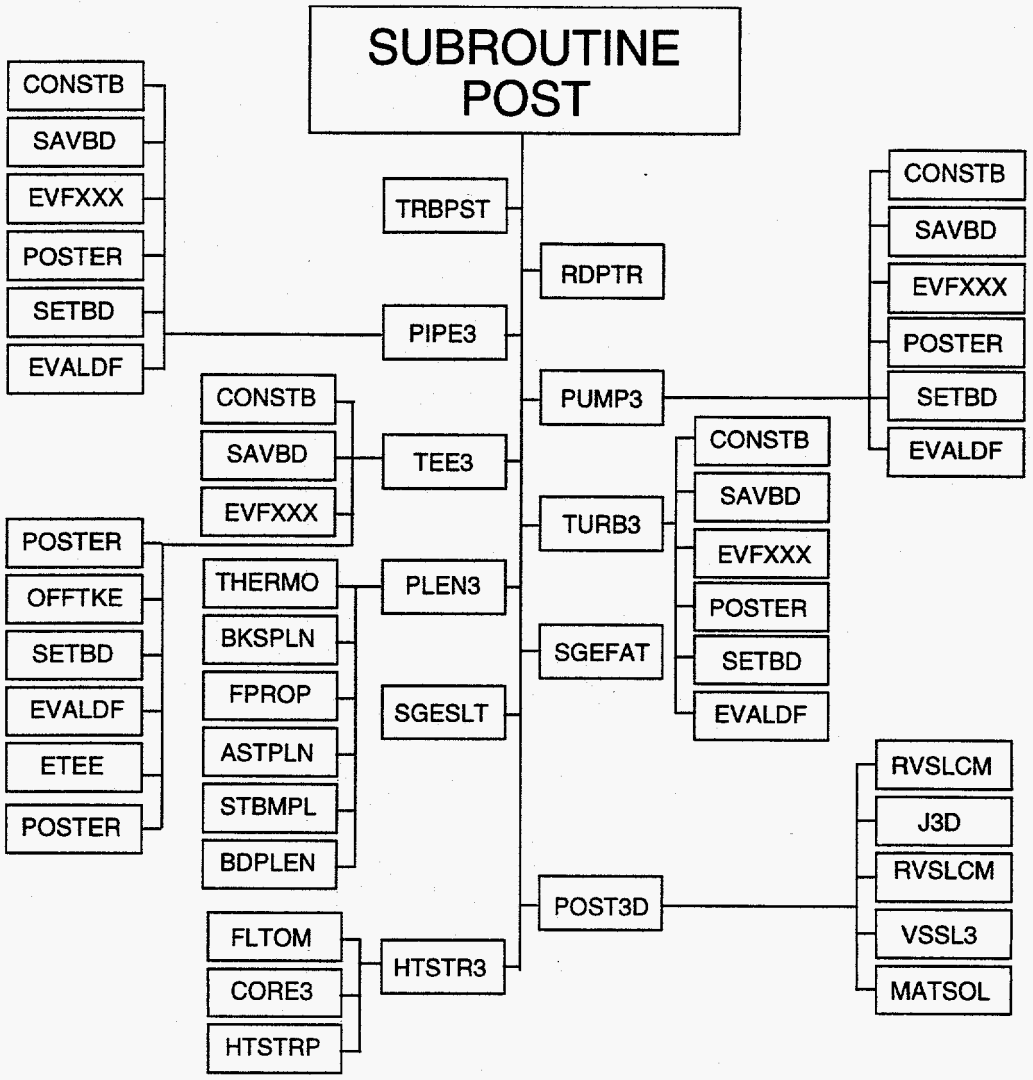


Fig. A.7.a. Top level of solution postpass solution (Post).

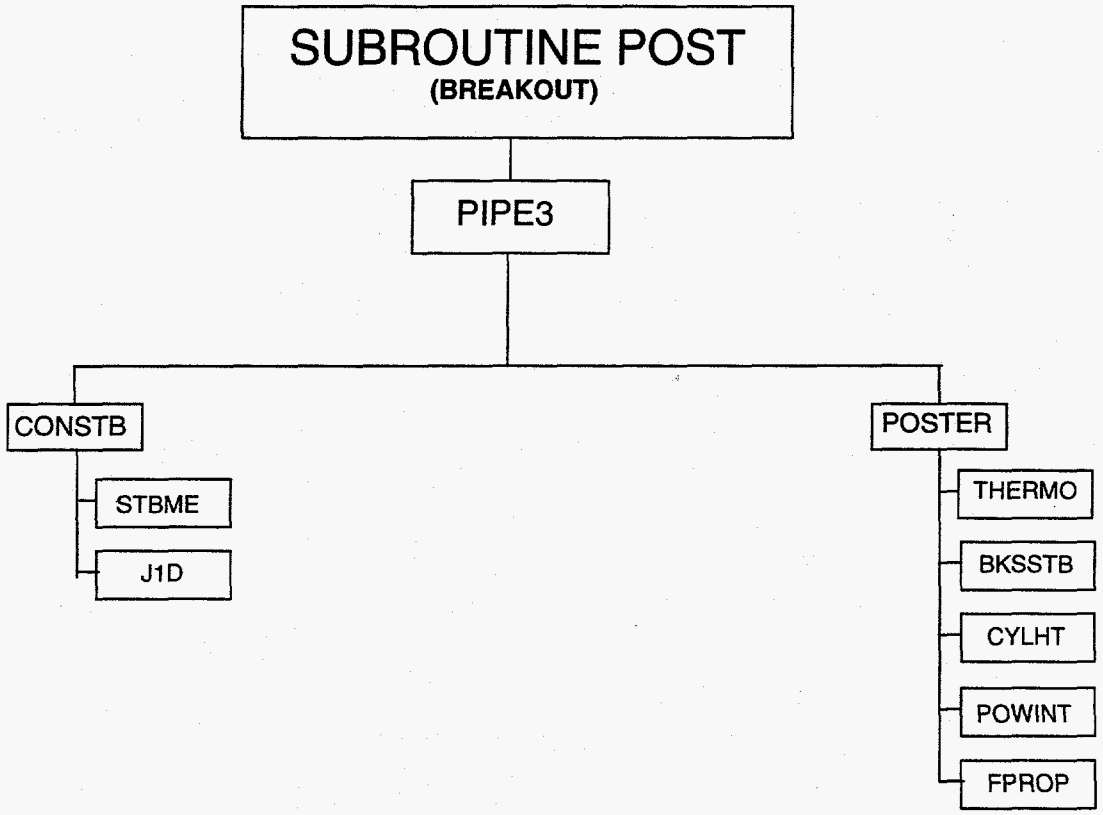


Fig. A.7.b. Breakout of top level of postpass solution (Post).

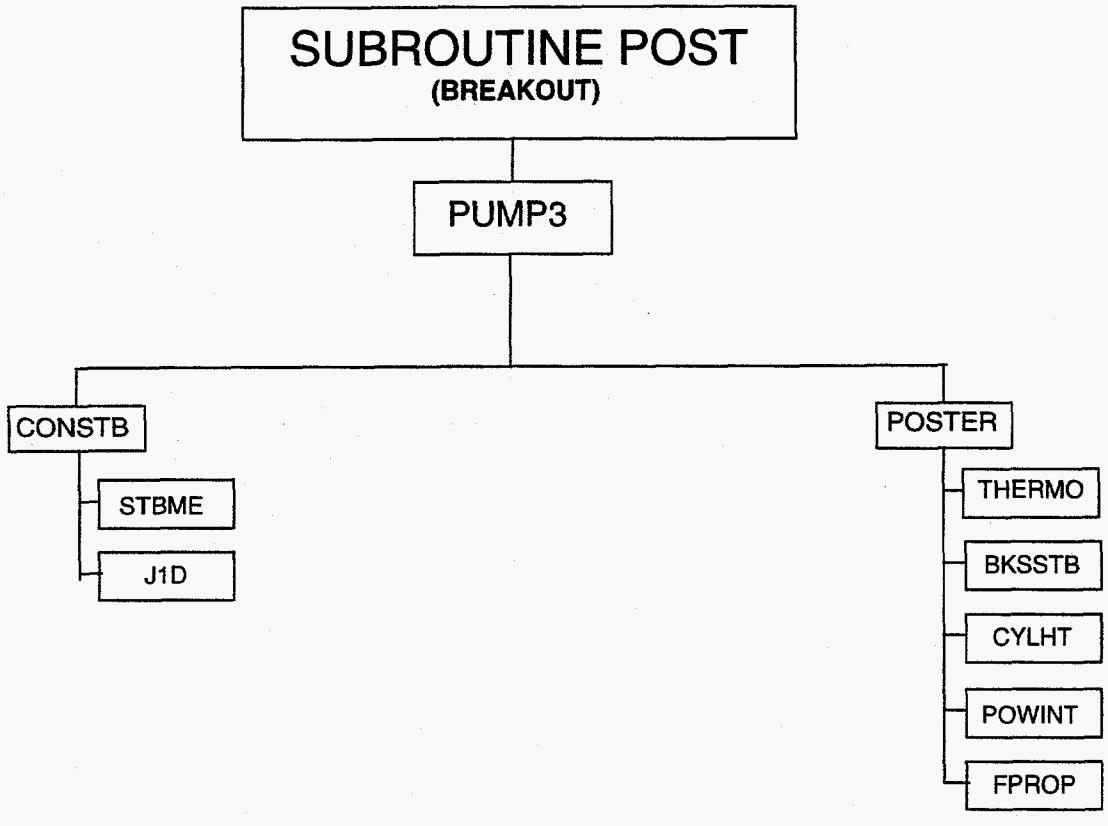


Fig. A.7.b. Breakout of top level of postpass solution (Post) (cont).

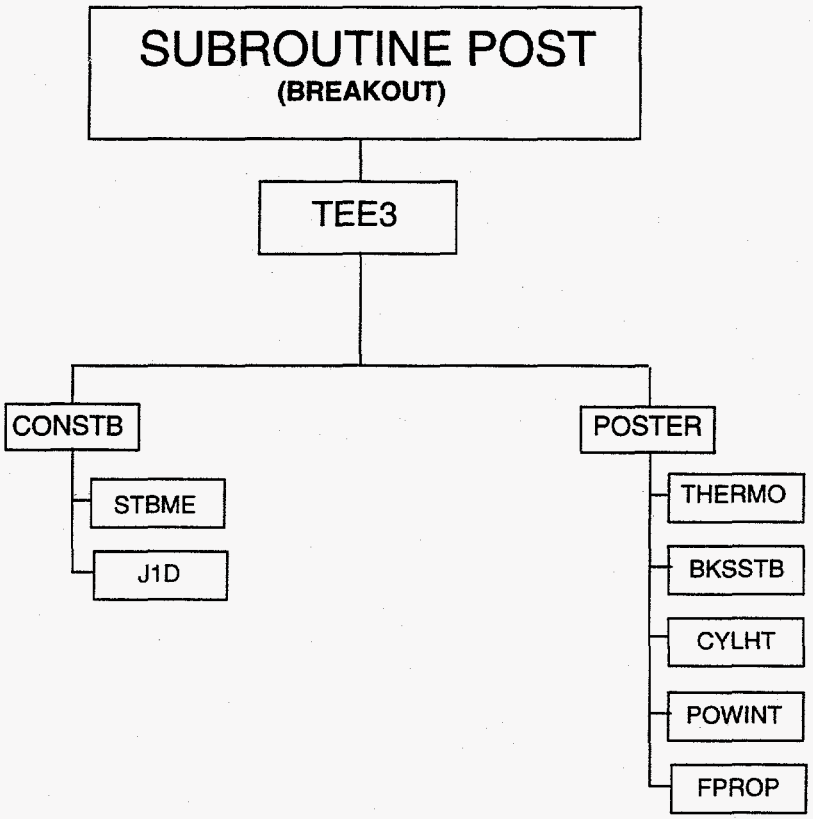


Fig. A.7.b. Breakout of top level of postpass solution (Post) (cont).

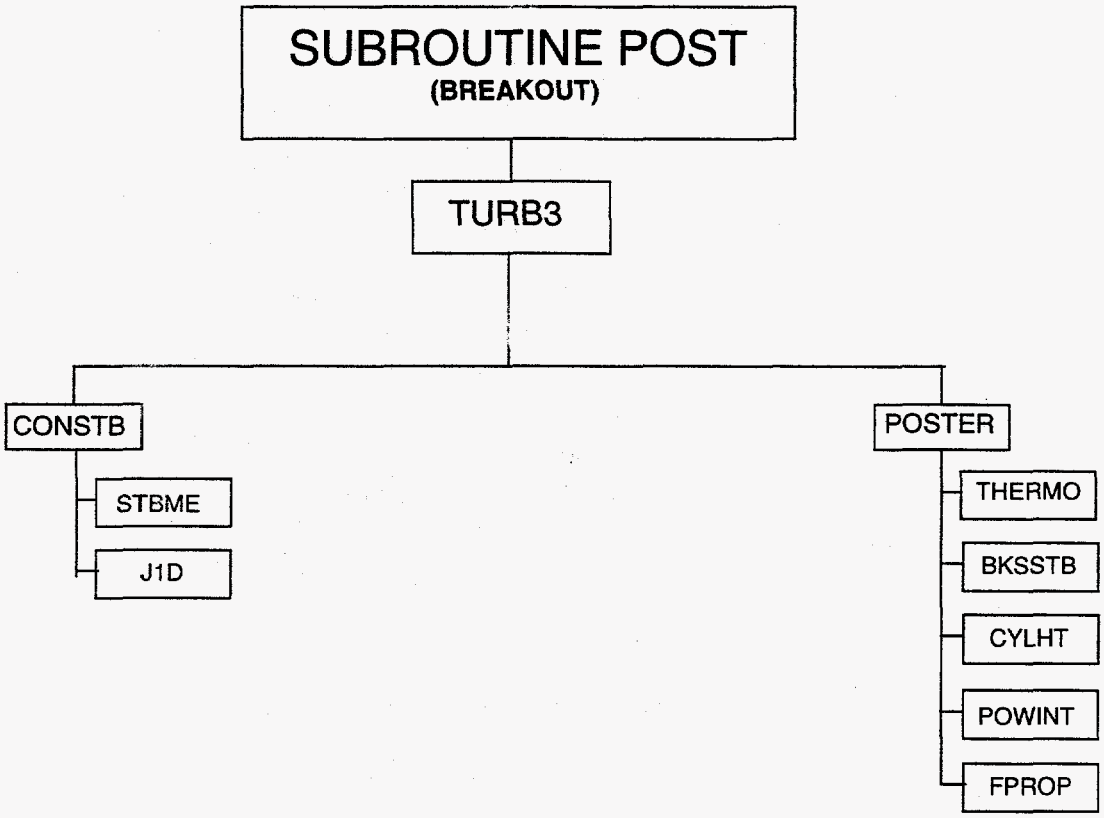


Fig. A.7.b. Breakout of top level of postpass solution (Post) (cont).

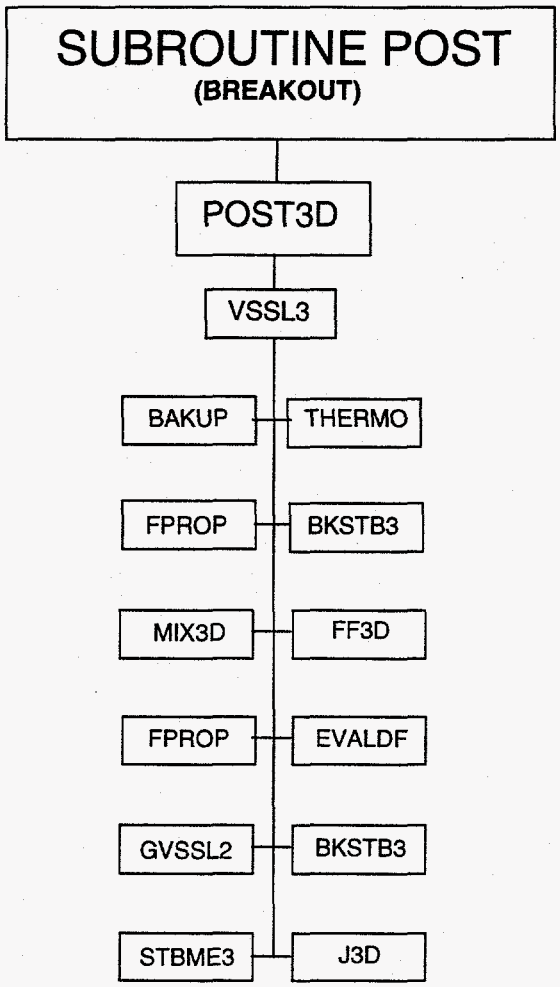


Fig. A.7.b. Breakout of top level of postpass solution (Post) (cont).

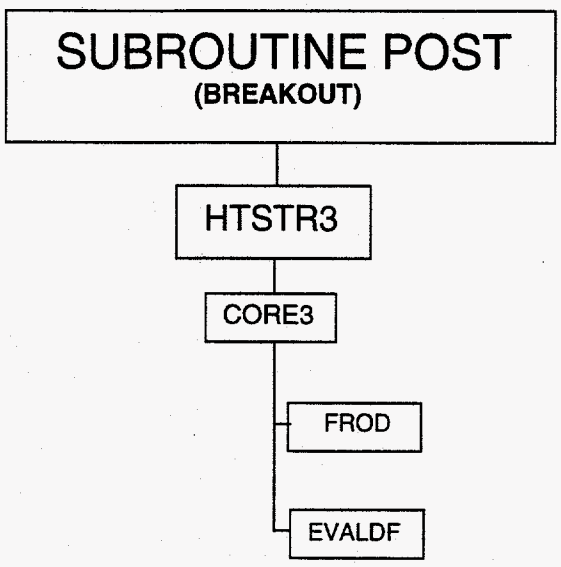


Fig. A.7.b. Breakout of top level of postpass solution (Post) (cont).

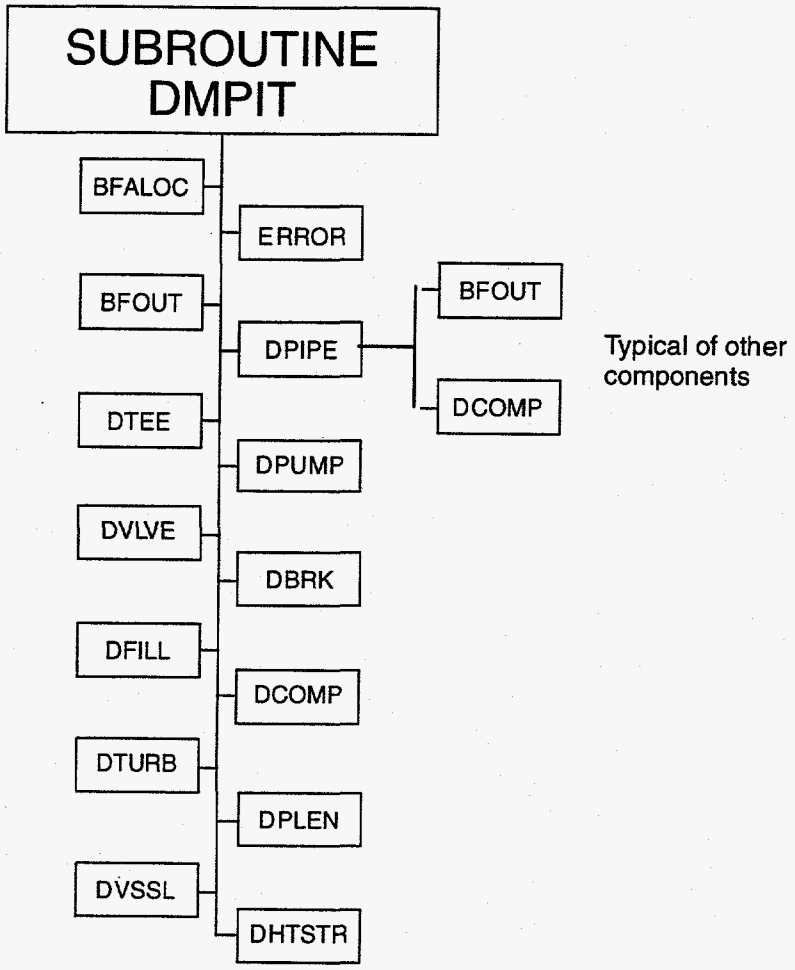


Fig. A.8. Output of restart data (Dmpit).

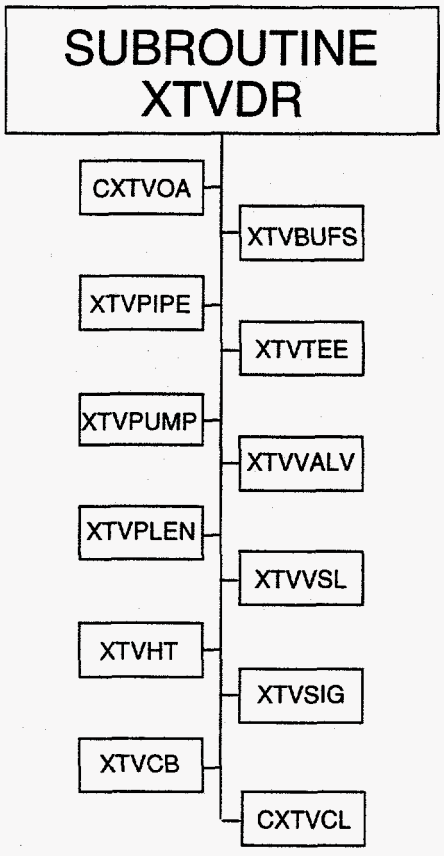


Fig. A.9. Output of graphics information (Xtvdr).

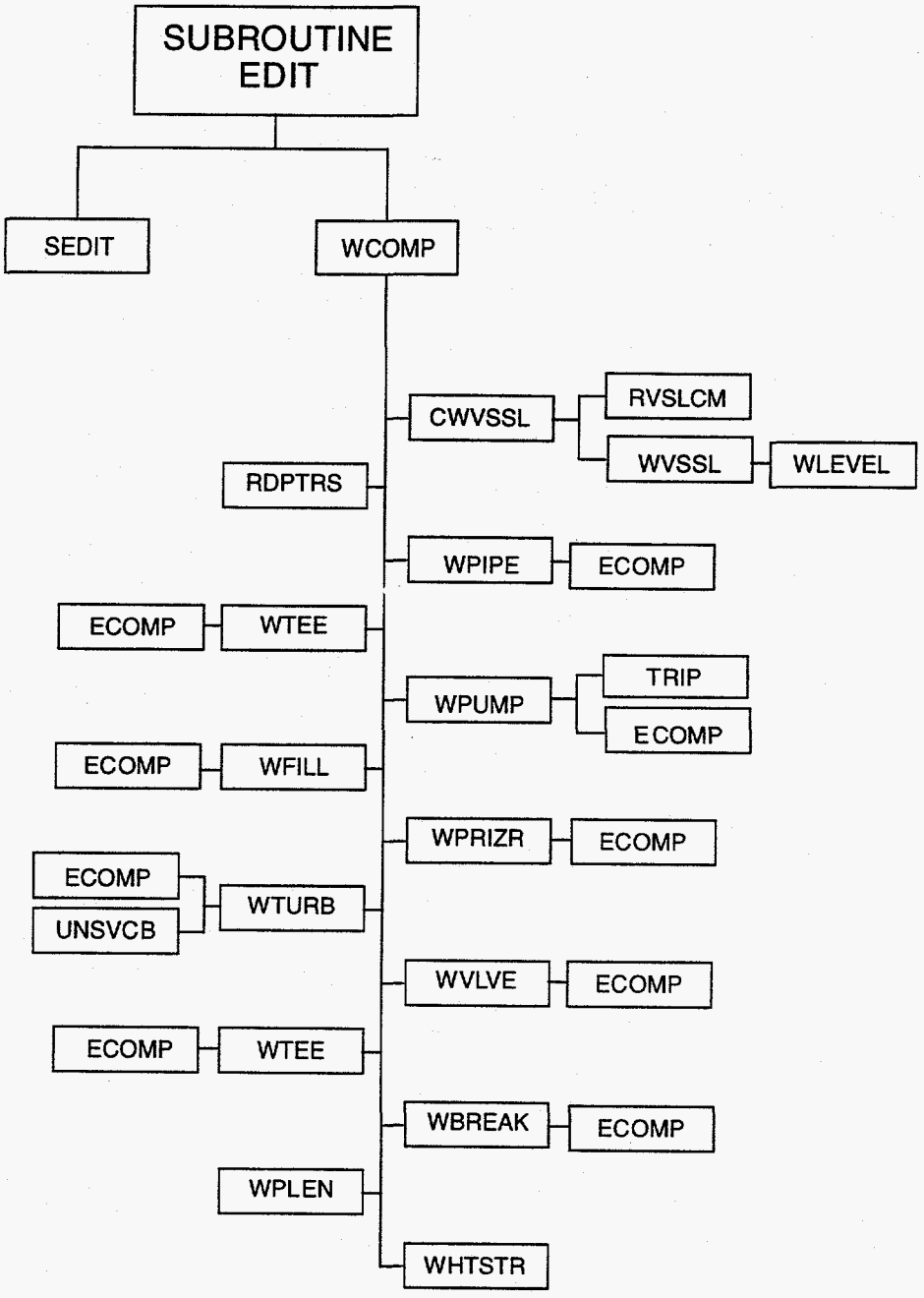


Fig. A.10. Detailed output (Edit).

APPENDIX B

SUBPROGRAM DATA INTERFACES

The full data interface for a given subprogram includes variables passed through the argument list, variables from common blocks and modules used, and variables available as global variables in a module that contains the subprogram. The description of this full interface for all subprograms is a very lengthy process and becomes obsolete as the program evolves. Rather than provide this description directly, we have elected to save space and maintain flexibility with a different approach. Definitions of variables are provided in App. C or, in special cases, in the subroutine headers themselves. Lists of communicating variables and their paths of communication are generated with a special program (datainfo). Source code for the program and results from executing it with the latest version of TRAC-M are provided on an attached disk.

B.1. Using the Program

The program (datainfo) was designed for more general use than simply TRAC analysis and should be supported by a driving script for most convenient use. It must be executed within the directory containing all source code to be processed and requires a subdirectory named VarInfo. The program reads a file named DoFiles that must be created in VarInfo to obtain a list of files to be processed. For the most basic usage, which is analyzing the communications for a full program, the steps are fairly simple. Before the first such analysis, the source code has been compiled to create an executable named "datainfo", and this executable has been in a directory contained in the PATH environment variable. First, change into the directory containing all source code, and make the necessary subdirectory:

```
mkdir VarInfo
```

Next, create the list of files to be processed by redirecting the output from the list command:

```
ls *.f *.f90 *.h >! VarInfo/Dofiles
```

Either the "*.f" or "*.f90" can be skipped in the above command if it is not appropriate. Now the communications analysis can be performed by typing:

```
datainfo
```

When the program has completed execution, the ".info" files generally will be removed, and single files with all information on communicating variables may be created. Execute the following commands:

```
cd VarInfo
rm *.info
cat *.hdr > full.hdr
```

The file "full.hdr" will be very long for a code like TRAC-M. Check the size and consider the consequences before sending it to a printer.

B.2. Description of the Program "datainfo"

Initial processing is driven by the subroutine "getlines" and results in the creation of intermediate files with the suffix ".info" (left in VarInfo), containing information on all communicating variables defined in program units or include files. The prefix of these intermediate file names is the name of the program unit, or include file. In addition, it creates a file in VarInfo named FileList, containing a list of all files containing source code to be processed for final information on communication interfaces.

The subroutine "mkHeaders" drives the final creation of header files containing a description of each variable communicating information into or out of a given subprogram. Final documentation is structured as header information for the subroutine or function-listing variables providing input to the subprogram, variables carrying output, and variables that are contained in the argument list of a called subroutine. Thus, these could be either input or output. This program currently does not iterate over subprograms to determine the true input/output status of such variables. In addition, the program may be unable to track the true input or output functionality of a variable when conditional branches are present. However, it produces warnings in such cases. Files containing this header information have a prefix matching the subprogram name and a suffix of ".hdr". The processing operates under the following restrictions and assumptions that

- EQUIVALENCES are not processed;
- Not all changes to variable values caused by optional arguments such as "read =" in INQUIRE are detected;
- The program being processed must successfully compile under Fortran 90;
- The ONLY attribute is not applied in a USE statement, nor is the rename feature applied;
- No variables in the MODULE are PRIVATE.

The program "datainfo" was tested in three ways. For each identified class of statements to be processed, a sample statement was inserted into one member of a set of test files and the names of these files placed in "DoFiles". As a second-level test, the source of "datainfo" was processed by "datainfo". This tested the processing of a fairly rich mix of Fortran 90. Results were carefully reviewed based on recent knowledge of the program. As a final step, the communications analysis was applied to Version 2.40 of TRAC-M. Too much information was generated for a thorough analysis of results. This test simply checked for fatal errors in "datainfo" and for obviously bad results for randomly sampled TRAC subprograms.

The existence of full source code for this tool permits adaptation to future needs with relative ease. Recommended changes include complete coverage of optional arguments to I/O statements such as INQUIRE, accommodation of USE options and PRIVATE variables, and an option to continually determine the status of variables found in Call statements. Additions to improve processing of usage within conditional statements are possible but require a higher investment of time. The subroutines outHeader and outVar should be replaced (or supplemented) by subroutines producing equivalent HTML files. This would provide cleaner formatting of the results within documents and would open the option for a browser-based document with links to variable definitions and to associated source code.

APPENDIX C

VARIABLE NAMES

C.1. Standard 1D Array Variable Names

The following list provides the standard variable name [the spatial location (cell center or cell edge)], and a definition of the variable. Variables in or passed from geometry and fluid-state arrays follow a fairly standard naming convention.

One naming convention worth noting is the use of "v". This dates to a time when steam vapor was the only gas followed in a TRAC calculation. Now, mixtures of vapor and noncondensable gas are possible, and variables using this "v", unless otherwise stated, apply to the total gas mixture. Another convention that must be treated with caution is the pattern for new- and old-time variables. When variables ending with "n" (e.g., pn, alpn, tln, or tvn) are present, variables with the root name (e.g., p, alp, tl, or tv) are evaluated at the old time. However, in many low-level subroutines, only the root name appears, and no inference should be made about the time level.

Additional confusion in the time level was introduced during the evolution of TRAC. In rare cases, the root name (vlt, vvt, hig, hil, or qppc) refers to the new-time value, and the addition of the suffix "o" denotes old-time values. Much of this problem has been eliminated through the Dual-derived type (see Section C.2). However, some local usage of this "o" convention persists and has been marked for future elimination.

Variable	Location	Definition
alp	Center	Old void fraction.
alpmn	Center	Minimum value of void fraction among a cell and all of its neighbors.
alpmx	Center	Maximum value of void fraction among a cell and all of its neighbors.
alpn	Center	New void fraction.
alpo	Center	Void fraction at the start of the previous step (step n-1).
alv	Center	Product of the old flashing interfacial HTC and the interfacial area.
alve	Center	Product of the old liquid-side interfacial HTC and the interfacial area.
alven	Center	Product of the new liquid-side interfacial HTC and the interfacial area.
alvn	Center	Product of the new flashing interfacial HTC and the interfacial area.
am	Center	Noncondensable gas mass.
ara	Center	Old stabilizer value for macroscopic noncondensable gas density ("D_a).
aran	Center	New stabilizer value for macroscopic noncondensable gas density ("D_a).

arc	Center	Solute macroscopic density: (1-")D_l c .
arel	Center	Old stabilizer value for (1-")$D_l e_l$.
areln	Center	New stabilizer value for (1-")$D_l e_l$.
arev	Center	Old stabilizer value for "$D_g e_g$.
arevn	Center	New stabilizer value for "$D_g e_g$.
arl	Center	Old stabilizer value for (1-") D_l (macroscopic liquid density).
arln	Center	New stabilizer value for (1-")D_l.
arv	Center	Old stabilizer value for "D_g (macroscopic gas density).
arvn	Center	New stabilizer value for "D_g .
bit	Either	Bit flags from previous timestep.
bitn	Either	Bit flags for current timestep.
chtan	Center	New value of noncondensable gas interfacial HTC times current volume interfacial area.
chti	Center	Old value of vapor-side interfacial HTC times current volume interfacial area.
chtia	Center	Old value of noncondensable gas interfacial HTC times current volume interfacial area.
chtin	Center	New value of vapor-side interfacial HTC times current volume interfacial area.
cif	Edge	Old interfacial drag coefficients.
cifn	Edge	New interfacial drag coefficients.
cl	Center	Liquid conductivity.
conc	Center	Old-solute-mass-to-coolant-mass ratio.
concn	Center	New-solute-mass-to-coolant-mass ratio.
cpl	Center	Liquid specific heat at constant pressure.
cpv	Center	Gas specific heat at constant pressure.
cv	Center	Gas conductivity.
dalva	Center	Derivative of alv with respect to void fraction (currently always set to zero).
dfldp	Edge	Derivative of liquid velocity with respect to pressure.
dfvdp	Edge	Derivative of gas velocity with respect to pressure.
dhldz	Edge	Gravitational head force caused by void gradient.
dr	Center	Storage array for thermodynamic derivatives and enthalpies.
dx	Center	Cell length in flow direction.
ea	Center	Old noncondensable gas-specific internal energy.
ean	Center	New noncondensable gas-specific internal energy.
el	Center	Old liquid-specific internal energy.
elev	Center	Cell-centered elevations.
eln	Center	New liquid-specific internal energy.
ev	Center	Old gas-specific internal energy.
evn	Center	New gas-specific internal energy.
fa	Edge	Cell-edge flow area.
favol	Center	Cell-centered flow area.
finan	Center	Inverted annular regime weighting factor (currently not used).
fric	Edge	Additive friction factors (generally for form losses).

fsmlt	Center	Interphasic area multiplier during condensation.
gam	Center	Old vapor generation rate per unit volume.
gamn	Center	New vapor generation rate per unit volume.
grav	Edge	Gravitation terms (cosine of the angle between the direction of increasing cell index and a vector directed vertically upward).
gravol	Center	Cell-averaged inclination cosine.
hd	Edge	Hydraulic diameters.
hdht	Center	Heat-transfer hydraulic diameters.
hfg	Center	Latent heat of vaporization.
hgam	Center	Contribution to phase change from subcooled boiling.
hig	Center	New HTC between inside wall and noncondensable gas.
higo	Center	Old HTC between inside wall and noncondensable gas.
hil	Center	New HTC between inside wall and liquid.
hilo	Center	Old HTC between inside wall and liquid.
hiv	Center	New HTC between inside wall and vapor.
hivo	Center	Old HTC between inside wall and vapor.
hla	Center	Sum of all products of liquid HTC with heat-transfer area.
hlatw	Center	Similar to HLA, except that the product includes wall surface temperature.
hva	Center	Sum of all products of vapor HTC with heat-transfer area.
hvatw	Center	Similar to HVA, except that the product includes wall surface temperature.
idr	Center	Heat-transfer regime.
lccfl	Edge	CCFL flag.
matid	Center	Structural material identifications.
nff	Edge	Wall friction-correlation options.
p	Center	Old total pressure.
pa	Center	Old noncondensable gas partial pressure.
pan	Center	New noncondensable gas partial pressure.
pn	Center	New total pressure.
qp3f	Center	QPPP factor applied to the wall heat source.
qppc	Center	New CHF.
qppco	Center	Old CHF.
qppp	Edge	Profile of the wall volumetric heat source.
regnm	Edge	Flow-regime number.
rhs	Center	Storage for SETS weighting factor xvset (amount of cell-centered implict mass or energy flux).
rmvm	Edge	Mixture density times mixture velocity.
roa	Center	Old noncondensable gas density.
roan	Center	New noncondensable gas density.
rol	Center	Old liquid density.
roln	Center	New liquid density.
rom	Center	Mixture density (old time level).
rov	Center	Old gas density (steam plus noncondensable gas).
rovn	Center	New gas density.

rvmf	Edge	Gas mass flow.
s	Center	Old solute mass plated on structure.
sidx	Edge	Stratified interfacial area.
sig	Center	Surface tension.
sn	Center	New solute mass plated on structure.
tl	Center	Old liquid temperature.
tln	Center	New liquid temperature.
trid	Either	Storage for stabilizer linear system.
tsat	Center	Saturation temperature at the total pressure.
tssn	Center	Saturation temperature for steam pressure.
tv	Center	Old gas temperature.
tvn	Center	New gas temperature.
tw	Center	Old wall temperatures.
twn	enter	New wall temperatures.
visl	Center	Liquid viscosity.
visv	Center	Gas mixture viscosity.
vl	Edge	Old liquid velocity.
vlalp	Center	Liquid mass flux that enters the cell from the cell edges located above the cell.
vln	Edge	New liquid velocity.
vlt	Edge	New stabilizer liquid velocity.
vlto	Edge	Old stabilizer liquid velocity.
vlvc	Center	Liquid velocity at a neighboring cell edge where the donor-celled liquid fraction is maximum.
vlvol	Center	Cell-centered liquid velocity.
vm	Edge	Old mixture velocity.
vmn	Edge	New mixture velocity.
vol	Center	Cell volume.
vr	Edge	Relative velocity.
vrv	Center	Cell-averaged relative velocity.
vv	Edge	Old gas velocity.
vvn	Edge	New gas velocity.
vvt	Edge	New stabilizer gas velocity.
vvto	Edge	Old stabilizer gas velocity.
vvvoL	Center	Cell-centered gas velocity.
wa	Center	Wall area in the current volume for component wall heat transfer.
wat	Center	Total heat-transfer area in the current volume associated with heat structures.
wfhf	Edge	Weighting factor for stratified-flow regime.
wfl	Edge	Wall friction factor for liquid.
wfv	Edge	Wall friction factor for gas.

C.2. Dual-Derived-Type Component Arrays

The following variables are allocated array pointers in the derived type containing information that are must be stored at both new- and old-time levels on each timestep for 1D components. Note that this group contains one anomaly. The variable tw interacts with the fluid as a surface wall temperature, but is a 2D array containing all temperatures associated with any wall conduction calculation. For the jth fluid cell, tw(1,j) provides the wall surface temperature for heat transfer to the fluid cell center. This convention is different from that used in the heat-structure component, where the structure temperature array aligns with the edges of the fluid cells.

Variable	Location	Definition
alp	Center	Void fraction.
alv	Center	Product of liquid-side flashing interfacial HTC and interfacial area.
alve	Center	Product of liquid-side evaporation interfacial HTC and interfacial area.
ara	Center	Stabilizer value for ρ_a.(macroscopic noncondensable gas density).
arel	Center	Stabilizer value for $(1-\alpha)\rho_l e_l$.
arev	Center	Old stabilizer value for $\rho_g e_g$.
arl	Center	Old stabilizer value for $(1-\alpha)\rho_l$ (macroscopic liquid density).
arv	Center	Old stabilizer value for ρ_g (macroscopic gas density).
bit	Either	Bit flags from previous timestep.
chti	Center	Product of vapor-side interfacial HTC and current volume interfacial area.
chtia	Center	Product of noncondensable gas interfacial HTC and current volume interfacial area.
cif	Edge	Interfacial drag coefficients.
conc	Center	Solute-mass-to-coolant-mass ratio.
ea	Center	Noncondensable gas specific internal energy.
el	Center	Liquid-specific internal energy.
ev	Center	Total gas-specific internal energy.
gam	Center	Vapor generation rate per unit volume.
hig	Center	HTC between inside wall and noncondensable gas.
hil	Center	HTC between inside wall and liquid.
hiv	Center	HTC between inside wall and vapor.
p	Center	Total pressure.
pa	Center	Noncondensable gas partial pressure.
qppc	Center	CHF.
roa	Center	Noncondensable gas density.
rol	Center	Liquid density.
rov	Center	Gas density (steam plus noncondensable).
s	Center	Solute mass plated on structures.
tl	Center	Liquid temperature.
tv	Center	Gas temperature.

tw	Center	Wall temperatures aligned with the center of the fluid cells but with the edge (node) of the radial conduction cells.
vl	Edge	Liquid velocity.
vlt	Edge	Stabilizer liquid velocity.
vm	Edge	Mixture velocity.
vv	Edge	Gas velocity.
vvt	Edge	Stabilizer gas velocity.

C.3. Hydro1D-Derived-Type Component Arrays

The following variables are allocated array pointers in the derived type containing variables that are defined only once (or once in a timestep) for 1D components. They are grouped by those associated with the fluid and those associated with the embedded heat structure (pipe wall).

Variable	Location	Definition
alpmn	Center	Minimum value of void fraction among a cell and all of its neighbors.
alpmx	Center	Maximum value of void fraction among a cell and all of its neighbors.
alpo	Center	Void fraction at the start of the previous step (step n-1).
am	Center	Noncondensable gas mass.
arc	Center	Solute macroscopic density: (1-")D_1 c .
cl	Center	Liquid conductivity.
cpl	Center	Liquid specific heat at constant pressure.
cpv	Center	Gas specific heat at constant pressure.
cv	Center	Gas conductivity.
dalva	Center	Derivative of alv with respect to void fraction (currently always set to zero).
dfldp	Edge	Derivative of liquid velocity with respect to pressure.
dfvdp	Edge	Derivative of gas velocity with respect to pressure.
dhldz	Edge	Gravitational head force caused by void gradient.
dr	Center	Derived-type array for thermodynamic derivatives and enthalpies. Type components are:
	deldp	Partial derivative of liquid-specific internal energy with respect to total pressure (temperature held constant).
	deldt	Partial derivative of liquid-specific internal energy with respect to temperature (total pressure held constant).
	devat	Partial derivative of noncondensable gas-specific internal energy with respect to temperature (total pressure held constant).
	devap	Partial derivative of noncondensable gas-specific internal energy with respect to pressure (temperature held constant).
	devdp	Partial derivative of gas-specific internal energy with respect to total pressure (temperature held constant).

	devdt	Partial derivative of gas-specific internal energy with respect to temperature (total pressure held constant).
	dhlsp	Derivative of liquid-specific saturation enthalpy with respect to pressure, evaluated at the total pressure.
	dhvsp	Derivative of steam-specific saturation enthalpy with respect to pressure, evaluated at the steam partial pressure.
	drolp	Partial derivative of liquid density with respect to total pressure (temperature held constant).
	drolt	Partial derivative of liquid density with respect to temperature (total pressure held constant).
	drovp	Partial derivative total gas density with respect to pressure (temperature held constant).
	drovt	Partial derivative of total gas density with respect to temperature (total pressure held constant).
	drvap	Partial derivative of noncondensable gas density with respect to pressure (temperature held constant).
	drvat	Partial derivative of noncondensable gas density with respect to temperature (pressure held constant).
	dtsdp	Derivative of saturation temperature with respect to pressure, evaluated at the total pressure.
	dtssp	Derivative of saturation temperature with respect to pressure, evaluated at the steam partial pressure.
	hlst	Liquid-specific enthalpy, evaluated at the total pressure.
	hvst	Steam-specific enthalpy, evaluated at the steam partial pressure.
dx	Center	Cell length in flow direction.
elev	Center	Cell-centered elevations.
fa	Edge	Cell-edge flow area.
favol	Center	Cell-centered flow area (volume divided by cell length).
finan	Center	Inverted annular regime weighting factor (currently not used).
fric	Edge	Additive friction factors (generally for form losses).
fsmlt	Center	Interphasic area multiplier during condensation.
grav	Edge	Gravitation terms (cosine of the angle between the direction of increasing cell index and a vector directed vertically upward).
gravol	Center	Cell-averaged inclination cosine.
hd	Edge	Hydraulic diameters.
hdht	Center	Heat-transfer hydraulic diameters.
hfg	Center	Latent heat of vaporization.
hgam	Center	Contribution to phase change from subcooled boiling.
hla	Center	Sum of all products of liquid HTC with heat-transfer area over all heat-structure components connected to a cell.
hlatw	Center	Similar to HLA, except that the product includes wall surface temperature.

hva	Center	Sum of all products of vapor HTC with heat-transfer area over all heat-structure components connected to a cell.
hvatw	Center	Similar to HVA, except that the product includes wall surface temperature.
lccfl	Edge	CCFL flag.
nff	Edge	Wall friction-correlation options.
regnm	Edge	Flow-regime number.
rhs	Center	Storage for SETS weighting factor xvset (amount of cell-centered implict mass or energy flux).
rmvm	Edge	Mixture density times mixture velocity (mass flux).
rom	Center	Mixture density.
.rvmf	Edge	Gas mass flow.
sidx	Edge	Stratified interfacial area.
sig	Center	Surface tension.
trid	Either	Storage for stabilizer linear system.
tsat	Center	Saturation temperature.
tssn	Center	Saturation temperature for steam pressure.
visl	Center	Liquid viscosity.
visv	Center	Gas viscosity.
vlalp	Center	Liquid mass flux that enters the cell from the cell edges located above the cell.
vlvc	Center	Liquid velocity at a neighboring cell edge where the donor-celled liquid fraction is maximum.
vlvol	Center	Cell-centered liquid velocity.
vol	Center	Cell volume.
vr	Edge	Relative velocity.
vrv	Center	Cell-averaged relative velocity.
vvvol	Center	Cell-centered gas velocity.
wfhf	Edge	Weighting factor for stratified-flow regime (1.0 is fully stratified).
wfl	Edge	Wall friction factor for liquid.
wfv	Edge	Wall friction factor for gas.

The following variables are associated with conduction from a heat structure directly associated with the component (pipe wall). Variables cpw, cw, matid, drw, rn, and rn2 are assumed to be constant along the length of the 1D component and are associated only with "nodes-1" conduction cell centers or, in the case of rn2, "nodes" conduction cell edges. Variables qppp and row can vary throughout the conduction mesh. They are associated with the conduction cell edge and center, respectively, and both align with the center of the fluid cells. The remaining variables vary only along the length of the 1D component and are alligned with the center of the fluid cells (dimensioned ncells).

Variable	Location	Definition
cpw	Center	Specific heat of wall material.
cw	Center	Wall conductivity.
matid	Center	Structural material identifications.
drw	Center	Radial mesh size.
emis	Center	Wall emissivity.
hol	Center	HTC between outside wall and liquid.
hov	Center	HTC between outside wall and vapor.
idr	Center	Heat-transfer regime (integer).
qp3f	Center	QPPP factor applied to the wall heat source.
qppp		Profile of the wall volumetric heat source, values for each combination of conduction cell edge and fluid cell center.
rn	Edge	Radii at the wall radial conduction cell edges (locations of temperatures).
rn2	Center	Radii at wall conduction cell centers.
row	Center	Material density at the center of each (ncells*(nodes-1)) wall conduction cell.
tchf	Center	CHF temperature, one for each hydro cell.
tol	Center	Liquid temperature outside wall. Exterior boundary condition for each hydro cell.
tov	Center	Vapor temperature outside wall. Exterior boundary condition for each hydro cell.
wa	Center	Wall area in the current volume for component wall heat transfer.
wat	Center	Total heat-transfer area in the current volume associated with heat structures.

C.4 Special 1D Shared Scalar Data—Module OneDDatM

REAL Variables

alpst The primary leg fluid void fraction to be convected into the Tee component side leg by the Tee offtake model (located at cell number "jcell").

ardmin Minimum value of the difference between the flow-area ratios one mesh-cell distance from a junction interface, with a Plenum component, and at the junction interface with a Plenum component for flow from the Plenum component.

arn No factor for applying flow-area ratios in the momentum-convection term.
- 0.0 = apply area ratios,
- 1.0 = do not apply area ratios.

ary Yes factor for applying flow-area ratios in the momentum-convection term.
- 1.0 = apply area ratios,
- 0.0 = do not apply area ratios.

c1a Fraction of liquid velocity at the left face of the Tee primary-leg junction cell that contributes to the momentum transfer into the Tee side leg.

c1av	Vapor velocity fraction at the left face of the Tee primary-leg junction cell that contributes to the momentum transfer into the Tee side leg.
c2a	Fraction of liquid velocity at the right face of the Tee primary-leg junction cell that contributes to the momentum transfer into the Tee side leg.
c2av	Vapor velocity fraction at the right face of the Tee junction cell that contributes to the momentum transfer into the Tee side leg.
ct	Cosine of the angle between the low numbered segment of the primary leg and the secondary leg, unless the cosine is positive, in which case Ct is zero.
ctp	Cosine of the angle between the low numbered segment of the primary leg and the secondary leg, unless the cosine is negative, in which case Ct is zero.
dvjp	Derivative of the pump (or turbine) momentum source term with respect to fluid velocity (assumes homogeneous flow).
fl1	Temporary storage for liquid mass-flow corrections for mass-conservation checks at low-numbered cell face (component junction).
fl2	Temporary storage for liquid mass-flow corrections for mass-conservation checks at high-numbered cell face (component junction).
fljp	K-factor turning plus abrupt flow-area change loss times the side-leg RHO*FA*VM**2 at a Tee internal junction that is to be assigned to the primary-side interfaces that flow into Jcell.
fljs	FRIC turning plus abrupt flow-area change loss at a Tee internal junction that is to be assigned to the side-leg internal-junction interface.
fv1	Temporary storage for vapor mass-flow corrections for mass-conservation checks at low-numbered cell face (component junction).
fv2	Temporary storage for vapor mass-flow corrections for mass-conservation checks at high-numbered cell face (component junction).
havlv	Temporary storage for the hydraulic diameter when the valve is open.
qtp	Direct energy deposited per unit length in the current 1D section during the current timestep.
s01	Factor (+1 or -1) necessary to make the velocity at the low-numbered cell face match the velocity in the component evaluating the momentum equation for that component junction.
s02	Factor (+1 or -1) necessary to make the velocity at the high-numbered cell face match the velocity in the component evaluating the momentum equation for that component junction.
salt	Source term to liquid for compressible work to the primary cell with a Tee junction.
savt	Source term to vapor for compressible work to the primary cell with a Tee junction.
ssac	Air mass source to the primary cell with a Tee junction.
sse	Total energy (liquid plus gas) source to the primary cell with a Tee junction.
ssmc	Total mass source to the primary cell with a Tee junction.
ssmom	Momentum source to cell face msc from a pump or turbine component.
ssvc	Gas mass source to the primary cell with a Tee junction.
ssve	Gas energy source to the primary cell with a Tee junction.
vjs	Mean velocity at the Tee-side-leg face joining to the primary (not used).

INTEGER Variables

i01	Index to the Network matrix and Network variable array providing the Network equation and variable at the junction adjacent to the current component's low-numbered cell. Zero if no network equation exists at that junction.
i02	Index to the Network matrix and Network variable array providing the Network equation and variable at the junction adjacent to the current component's high-numbered cell. Zero if no network equation exists at that junction.
i03	Index to the Network matrix and Network variable array providing the Network equation and variable at the junction between a Tee primary and secondary side. Zero if there is no Tee junction.
iacc2	Flag set to non-zero value if Pipe is used to model an accumulator.
ibks	Counter indicating whether the code is in a setup or back-substitution pass in the flow equation solutions. See Secs. II.1.4, II.1.5, and II.1.6.
icme	Index for referencing the portion of the iou array needed for the current 1D section.
il	1D Network loop number currently being computed.
iphsep	Phase-separation evaluation flag of the Tee offtake model, triggers special value for alpst.
isflg	Steam Generator flag (not used).
islb	Velocity calculation flag for component left (low-numbered face) junction.
isrb	Velocity calculation flag for component right (high-numbered face) junction.
ivpvlv	Interface number of the adjustable-valve flow area. Zero if no valve.
jstart	Array index for the cell at the left end of 1D segment.
lpindx	Index to the start point for the current network loop in contiguous constant arrays as dvb, drl, drv, dra, drel, and drev.
msc	Cell number for Tee primary leg connection source terms, or face number for a pump of turbine momentum source.
nc2	Array index for cell beginning a Tee side leg.
njn	Number of network junction in the current loop.
nstg	Counter for the number of steam separators.
ntee	Counter to locate Tee-side-leg information in the iou array.

APPENDIX D

PROPOSED FUTURE IMPROVEMENTS TO INFORMATION PASSING

Two tasks are scheduled that will improve the data interfaces within the code significantly. They also lay the groundwork for communication in a parallel execution of the main computational engine of the consolidated code. The first task will fully separate the evaluation of terms in the flow equations from the solution of the resulting system of linear equations. Once completed, this will provide a well-defined location for equation terms and eliminate the need for generation of this data for 1D components before evaluation of the equations in 3D components. The second task deals directly with the problem of intercomponent data communication, requiring only one request at initialization to establish automatic information passing between components. This is being implemented as a system service, with sufficient generality to permit later use by higher-order and more implicit difference methods.

Details of the solution modularization and resulting data interfaces are presented in Sec. D.1. The proposed intercomponent communication procedure is outlined in Sec. D.2.

D.1. Modularization of the Linear Equation Solutions

One major driving force behind this task is a need for flexibility within the code structure for replacement of numerical methods during the development cycle. At this stage, we have only a limited idea of the difference and solution methods that will be applied 5 or 10 years from now. The method of equation solution is not totally independent of the set of equations to be solved. However, for a given set of difference equations, many solution procedures will exist, and one best solution method may not exist for all problem nodalizations. Separation of the equation solution permits quick adoption of one or more solution methods. Even in instances where a new difference method drives a need for a new family of solution methods, this separation makes the division of labor and the isolation of testing easier.

The structure of arrays chosen for coefficients is to some extent governed by the structure of the equations to be solved. It is not the goal of this project to create a data interface general enough to handle all possible sets of equations. It is the goal to create an interface that is clean and restricted enough in the scope of the application that its replacement will be relatively simple if a major change is required in difference equations and solution methods.

D.1.1. Theory of the Solution Procedure

To keep the emphasis on the methods used, an initial discussion is built around a simple 1D single-phase flow model; a specific example of flow in a closed loop is illustrated in Fig. D-1. Cells and cell faces in Fig. D-1 have been given absolute numbers to facilitate discussion of full-system equation coupling. In terms of component numbering, cells 1–4 in this figure can be considered cells 1–4 of Pipe 1, and cells 5–8 in the figure could be cells 1–4 of Pipe 2.

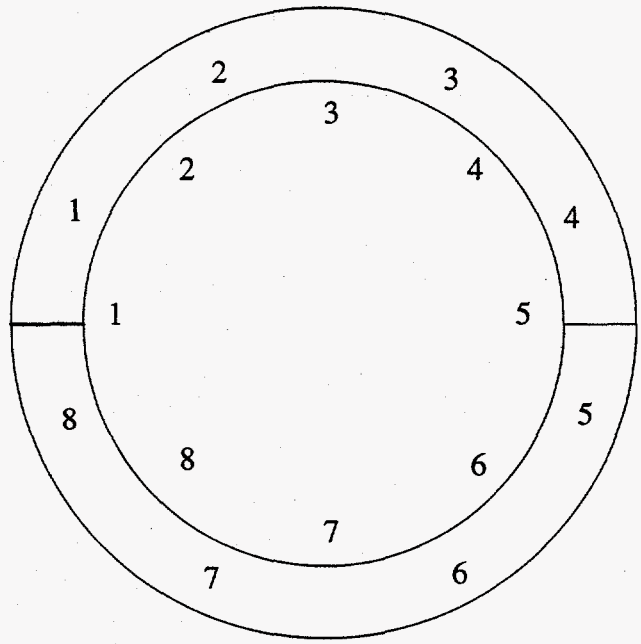


Fig. D-1. Two pipe-flow loop.

The nature of the underlying numerical method (SETS) and form of the solution procedure are not tightly linked to the details of the equations used in TRAC. These details are available in the TRAC-PF1/MOD2 Theory Manual and will not be repeated here. The simple set of equations representing 1D flow in constant area pipes is

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho V) = 0 \qquad \text{(D-1)}$$

$$\frac{\partial \rho e}{\partial t} + \frac{\partial}{\partial x}(\rho e V) = -p \cdot \frac{\partial V}{\partial x} \qquad \text{(D-2)}$$

and

$$\frac{\partial V}{\partial t} + V \cdot \frac{\partial V}{\partial x} = -\frac{1}{\rho} \cdot \frac{\partial p}{\partial x} - K \cdot V|V|$$

(D-3)

Here, K is a wall friction coefficient that may be a function of velocity and fluid properties.

A staggered spatial mesh is used for the difference equations, with thermodynamic properties evaluated at the cell centers and the velocity evaluated at the cell edges. Here, uniform cell lengths and constant area are assumed. When values of thermodynamic properties are required at cell edges, they are obtained from a donor cell formula:

$$\langle Y_{j+1/2} \rangle = w_{j+1/2} Y_j + (1 - w_{j+1/2}) \cdot Y_{j+1}$$

where

$$w_{j+1/2} = 1.0 \quad , V_{j+1/2} \geq 0$$

and

$$w_{j+1/2} = 0.0 \quad , V_{j+1/2} < 0$$

(D-4)

Here, Y may be any state variable. With this definition of averaging, spatial differences of flux terms become

$$\frac{\partial}{\partial x_j}(YV) = \frac{[w_{j+1/2} Y_j + (1 - w_{j+1/2}) \cdot Y_{j+1}] \cdot V_{j+1/2} - [w_{j-1/2} Y_{j-1} + (1 - w_{j-1/2}) \cdot Y_j] \cdot V_{j-1/2}}{\Delta x}$$

$$= \frac{\langle Y \rangle_{j+1/2} \cdot V_{j+1/2} - \langle Y \rangle_{j-1/2} \cdot V_{j-1/2}}{\Delta x}$$

(D-5)

For our purposes, the numerical approximation to the momentum flux term $V \cdot \nabla V$ is taken to be

$$V_{j+1/2}\frac{[(1-w_{j+1/2})V_{j+3/2}+(2w_{j+1/2}-1)V_{j+1/2}-w_{j+1/2}V_{j-1/2}]}{\Delta x}$$

(D-6)

The actual momentum transport term in TRAC-M is more complex, involving area scaling, but results in the same form of linear equations that will be seen here.

D.1.1.1. Stabilizer Motion Equations

The SETS equations implemented in TRAC-M begin each timestep with a solution of a stabilizer motion equation, with the following general form:

$$\left(\tilde{V}^{n+1}_{j+1/2}-V^{n}_{j+1/2}\right)/\Delta t+\tilde{V}^{n}_{j+1/2}\frac{\left[(1-w_{j+1/2})\tilde{V}^{n+1}_{j+3/2}+(2w_{j+1/2}-1)\tilde{V}^{n+1}_{j+1/2}-w_{j+1/2}\tilde{V}^{n+1}_{j-1/2}\right]}{\Delta x}$$

$$+\frac{1}{\langle\rho\rangle^{n}_{j+1/2}}\cdot\frac{\left(p^{n}_{j+1}-p^{n}_{j}\right)}{\Delta x}+K^{n}_{j+1/2}(2\tilde{V}^{n+1}_{j+1/2}-V^{n}_{j+1/2})\left|V^{n}_{j+1/2}\right|=0.$$

(D-7)

This equation is purely linear in the unknown stabilizer velocities. When the tilde and superscript are dropped for simplicity, the general form for this linear system for the flow loop in Fig. D-1 is

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & a_{1,8} \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,5} & a_{4,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} \\ a_{8,1} & 0 & 0 & 0 & 0 & 0 & a_{8,7} & a_{8,8} \end{pmatrix}\begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{pmatrix}=\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{pmatrix}$$

(D-8)

One standard trick in linear algebra to solve this problem is to break it into blocks that can be solved more easily. One obvious approach would be to isolate the last row and column of the matrix:

$$\left(\begin{array}{ccccccc|c} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & a_{1,8} \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,5} & a_{4,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} \\ \hline a_{8,1} & 0 & 0 & 0 & 0 & 0 & a_{8,7} & a_{8,8} \end{array}\right)\left(\begin{array}{c} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{array}\right) = \left(\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{array}\right) \qquad \text{(D-9)}$$

This then can be written more clearly as the following problem:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} & 0 & 0 \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} - \begin{bmatrix} a_{1,8} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ a_{7,8} \end{bmatrix} \cdot V_8 \qquad \text{(D-10)}$$

and

$$a_{8,1} V_1 + a_{8,7} V_7 + a_{8,8} V_8 = b_8 \qquad \text{(D-11)}$$

Equation (D-10) is solved to obtain velocities V_1 through V_7 as linear functions of V_8. The existence of two constant vectors on the right-hand side of the equation means that two solutions of a 7 x 7 system are required. However, using a lower-upper (LU) decomposition method makes the cost of the second solution insignificant compared to the cost of the first. Once these solutions are available, the specific linear expressions for V_1 and V_7 as functions of V_8 are substituted into Eq. (D-11), and a value of V_8 is obtained. Back substitution of this value into the equations for the other velocities completes the solution.

In Fig. D-1, there is nothing unusual about V_8 from the standpoint of the chosen component structure. Although a final implementation of the solution procedure may function as described above, the initial implementation must follow the current network solution procedure. In that case, the velocities at the component junctions (V_1 and V_5) take on special significance as network variables. The full linear system is partitioned to isolate the junction variables and junction equations:

$$\left(\begin{array}{c|ccc|c|cc|c} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & a_{1,8} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,4} & a_{4,5} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} \\ a_{8,1} & 0 & 0 & 0 & 0 & 0 & a_{8,7} & a_{8,8} \end{array}\right)\left(\begin{array}{c} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{array}\right) = \left(\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{array}\right) \qquad \text{(D-12)}$$

Actual solution of the linear system follows a process similar to the initial example. The well-structured blocks are isolated as

$$\begin{bmatrix} a_{2,2} & a_{2,3} & 0 \\ a_{3,2} & a_{3,3} & a_{3,4} \\ 0 & a_{4,3} & a_{4,4} \end{bmatrix} \cdot \begin{bmatrix} V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_3 \\ b_4 \end{bmatrix} + \begin{bmatrix} a_{2,1} \\ 0 \\ 0 \end{bmatrix} \cdot V_1 - \begin{bmatrix} 0 \\ 0 \\ a_{4,5} \end{bmatrix} \cdot V_5 \qquad \text{(D-13)}$$

$$\begin{bmatrix} a_{6,6} & a_{6,7} & 0 \\ a_{7,6} & a_{7,7} & a_{7,8} \\ 0 & a_{8,7} & a_{8,8} \end{bmatrix} \cdot \begin{bmatrix} V_6 \\ V_7 \\ V_8 \end{bmatrix} = \begin{bmatrix} b_6 \\ b_7 \\ b_8 \end{bmatrix} + \begin{bmatrix} a_{6,5} \\ 0 \\ 0 \end{bmatrix} \cdot V_5 - \begin{bmatrix} 0 \\ 0 \\ a_{8,1} \end{bmatrix} \cdot V_1 \qquad \text{(D-14)}$$

$$a_{1,1} V_1 + a_{1,2} V_2 + a_{1,8} V_8 = b_1$$

and

$$a_{5,4} V_4 + a_{5,5} V_5 + a_{5,6} V_6 = b_5 \qquad \text{(D-15)}$$

The cell edge at a junction between two components is shared by both components, and a decision must be made as to which of these two components takes responsibility for evaluating the terms in the momentum equation at this edge. The current procedure is to give that responsibility to the one appearing first in the order of component processing. In a later parallel version of the code, the responsibility generally will be assigned to the first of the components processed during problem initialization.

Equations (D-13) and (D-14) are solved for interior velocities as linear functions of the junction velocities:

$$\begin{pmatrix} V_2 \\ V_3 \\ V_4 \end{pmatrix} = \begin{pmatrix} b'_2 \\ b'_3 \\ b'_4 \end{pmatrix} + \begin{pmatrix} a'_{2,1} \\ a'_{3,1} \\ a'_{4,1} \end{pmatrix} V_1 + \begin{pmatrix} a'_{2,5} \\ a'_{3,5} \\ a'_{4,5} \end{pmatrix} V_5 \tag{D-16}$$

and

$$\begin{pmatrix} V_6 \\ V_7 \\ V_8 \end{pmatrix} = \begin{pmatrix} b'_6 \\ b'_7 \\ b'_8 \end{pmatrix} + \begin{pmatrix} a'_{6,1} \\ a'_{7,1} \\ a'_{8,1} \end{pmatrix} V_1 + \begin{pmatrix} a'_{6,5} \\ a'_{7,5} \\ a'_{8,5} \end{pmatrix} V_5 \tag{D-17}$$

Reduction of the blocked network solution method is completed by substituting the results in Eqs. (D-16) and (D-17) into Eq. (D-15), yielding

$$\begin{aligned} (a_{1,1} + a_{1,2}\, a'_{2,1} + a_{1,8}\, a'_{8,1})\, V_1 + (a_{1,2}\, a'_{2,5} + a_{1,8}\, a'_{8,5})\, V_5 \\ = b_1 - a_{1,2}\, b'_2 - a_{1,8}\, b'_8 \end{aligned} \tag{D-18}$$

and

$$\begin{aligned} (a_{5,4}\, a'_{4,1} + a_{5,6}\, a'_{6,1})\, V_1 + (a_{5,5} + a_{5,4}\, a'_{4,5} + a_{5,6}\, a'_{6,5})\, V_5 \\ = b_5 - a_{5,4}\, b'_4 - a_{5,6}\, b'_6 \end{aligned} \tag{D-19}$$

This is a closed system that can be solved for V_1 and V_5. Back substitution of these values into Eqs. (D-16) and (D-17) completes the solution of the system.

D.1.1.2. Basic Equations

After the stabilizer motion equations are solved, the version of SETS implemented in TRAC-M proceeds to solve the "basic" equations for motion, mass, and energy. Apart from the use of stabilizer velocities in the momentum transport term of the motion equation, these equations are equivalent to the standard semi-implicit method used in older versions of TRAC-M. A tilde over a quantity indicates that it is an interim result to

be replaced with a final nontilde value before completion of the timestep:

$$(V_{j+1/2}^{n+1} - V_{j+1/2}^{n})/\Delta t + \tilde{V}_{j+1/2}^{n} \frac{[(1-w_{j+1/2})\tilde{V}_{j+3/2}^{n+1} + (2w_{j+1/2}-1)\tilde{V}_{j+1/2}^{n+1} - w_{j+1/2}\tilde{V}_{j-1/2}^{n+1}]}{\Delta x}$$

$$+ \frac{1}{\langle\rho\rangle_{j+1/2}^{n}} \frac{(p_{j+1}^{n+1} - p_{j}^{n+1})}{\Delta x} + K_{j+1/2}^{n}(2V_{j+1/2}^{n+1} - V_{j+1/2}^{n})\left|V_{j+1/2}^{n}\right| = 0 \qquad \text{(D-20)}$$

$$\frac{(\tilde{\rho}_{j}^{n+1} - \rho_{j}^{n})}{\Delta t} + \frac{[w_{j+1/2}\rho_{j}^{n} + (1-w_{j+1/2})\rho_{j+1}^{n}]V_{j+1/2}^{n+1} - [w_{j-1/2}\rho_{j-1}^{n} + (1-w_{j-1/2})\rho_{j}^{n}]V_{j-1/2}^{n+1}}{\Delta x} = 00 \qquad \text{(D-21)}$$

and

$$\frac{\left(\tilde{\rho}_{j}^{n+1}\tilde{e}_{j}^{n+1} - \rho_{j}^{n}e_{j}^{n}\right)}{\Delta t} +$$

$$\frac{\left[w_{j+1/2}(\rho e)_{j}^{n} + (1-w_{j+1/2})(\rho e)_{j+1}^{n}\right]V_{j+1/2}^{n+1} - \left[w_{j-1/2}(\rho e)_{j-1}^{n} + (1-w_{j-1/2})(\rho e)_{j}^{n}\right]V_{j-1/2}^{n+1}}{\Delta x} \qquad \text{(D-22)}$$

$$+\tilde{p}_{j}^{n+1}\frac{V_{j+1/2}^{n+1} - (\rho e)_{j}^{n}}{\Delta x} = 0$$

The solution begins by solving the motion equation directly to obtain the new time velocity at each cell edge as a linear function of the pressure difference across that edge. This is substituted into the mass and energy equations to eliminate the new-time velocity as an unknown in those equations. For each cell, these two remaining flow equations, combined with the necessary state relationships [ρ(p,T) and e(p,T)], give two nonlinear equations with new-time pressure and temperature as the two independent variables. These equations are linearized (part of a standard Newton iteration) with the substitutions:

$$\begin{aligned} \tilde{T}_{j}^{n+1,i+1} &= \tilde{T}_{j}^{n+1,i} + \delta T_{j} \\ \tilde{p}_{j}^{n+1,i+1} &= \tilde{p}_{j}^{n+1,i} + \delta p_{j} \end{aligned} \qquad \text{(D-23)}$$

The second superscript in these equations is the iteration count. An auxiliary variable is defined as

$$\Delta p_{j+1/2} = \delta p_{j+1} - \delta p_{j} \qquad \text{(D-24a)}$$

The iteration proceeds by making substitutions of the following form into the difference equations:

$$\tilde{p}_j^{n+1} \Rightarrow \tilde{p}_j^{n+1,i} + \delta p_j$$
$$\tilde{T}_j^{n+1} \Rightarrow \tilde{T}_j^{n+1,i} + \delta T_j \qquad \text{(D-24b)}$$

The ith iterate values are all known, and the variations (δp's and δT's) are assumed to be small enough that nonlinear combinations of them (δp, δT, δp2, δT2, etc.) can be ignored. This results in a set of linear equations for each cell in the form:

$$\begin{pmatrix} a_{j,1,1} & a_{j,1,2} \\ a_{j,2,1} & a_{j,2,2} \end{pmatrix} \begin{pmatrix} \delta p_j \\ \delta T_j \end{pmatrix} = \begin{bmatrix} b_{j,1} \\ b_{j,2} \end{bmatrix} - \begin{pmatrix} c_{lj,1} \\ c_{lj,2} \end{pmatrix} \Delta p_{j-1/2} + \begin{pmatrix} c_{rj,1} \\ c_{rj,2} \end{pmatrix} \Delta p_{j+1/2} \quad . \qquad \text{(D-25)}$$

The first row in the above linear system can be considered to be the linearized mass conservation equation and the second to be the linearized energy equation. This system is solved for the cell pressure and temperature variations in the form:

$$\begin{pmatrix} \delta p_j \\ \delta T_i \end{pmatrix} = \begin{pmatrix} b'_{j,1} \\ b'_{j,2} \end{pmatrix} + \begin{pmatrix} c'_{lj,1} \\ c'_{lj,2} \end{pmatrix} \Delta p_{j-1/2} + \begin{pmatrix} c'_{rj,1} \\ c'_{rj,2} \end{pmatrix} \Delta p_{j+1/2} \quad . \qquad \text{(D-26)}$$

At this point, the b′ constants represent the linearized predictions of change in pressure and temperature assuming no further velocity changes at the cell faces. The c′ coefficients account for contributions caused by velocity changes (driven by changes in the pressure gradient).

Completing the solution of the basic equations requires two steps within the current version of TRAC. Within each component, the pressure equations are isolated, and Δp terms on interior faces are eliminated by substitution of the defining Eq. (D-25). For our sample problem, this gives a tridiagonal linear system for each pipe:

$$\begin{bmatrix} 1+c'_{rl,1} & -c'_{rl,1} & 0 & 0 \\ -c'_{l2,1} & 1+c'_{l2,1}+c'_{r2,1} & -c'_{r2,1} & 0 \\ 0 & -c'_{l3,1} & 1+c'_{l3,1}+c'_{r3,1} & -c'_{r3,1} \\ 0 & 0 & -c'_{l4,1} & 1+c'_{l4,1} \end{bmatrix} \begin{bmatrix} \delta p_1 \\ \delta p_2 \\ \delta p_3 \\ \delta p_4 \end{bmatrix} = \begin{bmatrix} b'_{1,1} \\ b'_{2,1} \\ b'_{3,1} \\ b'_{4,1} \end{bmatrix} - \begin{bmatrix} c'_{l1,1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \Delta p_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ c'_{r4,1} \end{bmatrix} \Delta p_5$$
(D-27)

and

$$\begin{bmatrix} 1+c'_{r5,1} & -c'_{r5,1} & 0 & 0 \\ -c'_{16,1} & 1+c'_{16,1}+c'_{r6,1} & -c'_{r6,1} & 0 \\ 0 & -c'_{17,1} & 1+c'_{17,1}+c'_{r7,1} & -c'_{r7,1} \\ 0 & 0 & -c'_{18,1} & 1+c'_{18,1} \end{bmatrix} \begin{bmatrix} \delta p_5 \\ \delta p_6 \\ \delta p_7 \\ \delta p_8 \end{bmatrix} = \begin{bmatrix} b'_{5,1} \\ b'_{6,1} \\ b'_{7,1} \\ b'_{8,1} \end{bmatrix} - \begin{bmatrix} c'_{15,1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \Delta p_5 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ c'_{r8,1} \end{bmatrix} \Delta p_1 \tag{D-28}$$

Here, as before, the absolute cell index number is used from Fig. D-1, rather than the component cell index number. The subscripts on the Δp terms refer to absolute face indices shown in the figure. These equations currently are solved within Tf1Ds to obtain the pressure variations. This function will be moved to a subroutine "triSolve", called from "blockSolver". Results of this solution are

$$\begin{pmatrix} \delta p_1 \\ \delta p_2 \\ \delta p_3 \\ \delta p_4 \end{pmatrix} = \begin{pmatrix} b''_{1,1} \\ b''_{2,1} \\ b''_{3,1} \\ b''_{4,1} \end{pmatrix} - \begin{pmatrix} c''_{11,1} \\ c''_{12,1} \\ c''_{13,1} \\ c''_{14,1} \end{pmatrix} \Delta p_1 + \begin{pmatrix} c''_{r1,1} \\ c''_{r2,1} \\ c''_{r3,1} \\ c''_{r4,1} \end{pmatrix} \Delta p_5 \tag{D-29}$$

and

$$\begin{pmatrix} \delta p_5 \\ \delta p_6 \\ \delta p_7 \\ \delta p_8 \end{pmatrix} = \begin{pmatrix} b''_{5,1} \\ b''_{6,1} \\ b''_{7,1} \\ b''_{8,1} \end{pmatrix} - \begin{pmatrix} c''_{15,1} \\ c''_{16,1} \\ c''_{17,1} \\ c''_{18,1} \end{pmatrix} \Delta p_5 + \begin{pmatrix} c''_{r5,1} \\ c''_{r6,1} \\ c''_{r7,1} \\ c''_{r8,1} \end{pmatrix} \Delta p_1 \quad . \tag{D-30}$$

Results of these solutions finally are combined via the definition of the network junction variables. For this example, the defining equations for the junction variables are

$$\Delta p_1 = \delta p_1 - \delta p_8, \quad \Delta p_5 = \delta p_5 - \delta p_4 \quad . \tag{D-31}$$

Substituting in the δp's from Eqs. (D-30) and (D-31) gives

$$\Delta p_1 = b''_{1,1} - c''_{11,1}\Delta p_1 + c''_{r1,1}\Delta p_5, - b''_{8,1} + c''_{18,1}\Delta p_5 - c''_{r8,1}\Delta p_1 \quad \text{(D-32)}$$

and

$$\Delta p_5 = b''_{5,1} - c''_{15,1}\Delta p_5 + c''_{r5,1}\Delta p_1, - b''_{4,1} + c''_{14,1}\Delta p_1 - c''_{r4,1}\Delta p_5 \quad . \quad \text{(D-33)}$$

These can be rearranged to the final form of the network equations:

$$(1 + c''_{11,1} + c''_{r8,1})\Delta p_1 - (c''_{r1,1} + c''_{18,1})\Delta p_5 = b''_{1,1} - b''_{8,1} \quad \text{(D-34)}$$

and

$$-(c''_{r5,1} + c''_{14,1})\Delta p_1 + (1. + c''_{15,1} + c''_{r4,1})\Delta p_5 = b''_{5,1} - b''_{4,1} \quad . \quad \text{(D-35)}$$

These form a closed system in the network variables; this system is solved directly. Back substitution follows into Eqs. (D-29) and (D-30), giving pressures that can be back substituted into Eq. (D-26) to provide the iteration change in the new-time temperatures.

D.1.1.3. Stabilizer Mass and Energy Equations

The final step in the SETS method is the solution of the stabilizer mass and energy equations. At this point, the new-time velocities have been determined and can be treated as constants in the solution of the equations. The equations vary from the basic mass and energy equations only in that the densities and energies in flux terms are now evaluated at the new time:

$$\frac{(\tilde{\rho}_j^{n+1} - \rho_j^n)}{\Delta t} + \frac{[w_{j+1/2}\rho_j^n + (1 - w_{j+1/2})\rho_{j+1}^{n+1}]V_{j+1/2}^{n+1} - [w_{j-1/2}\rho_{j-1}^{n+1} + (1 - w_{j-1/2})\rho_j^{n+1}]V_{j-1/2}^{n+1}}{\Delta t} = 00$$

(D-36)

$$\frac{(\rho e)_j^{n+1} - \rho e^n{}_j}{\Delta t}$$

$$[w_{j+1/2}(\rho e)_j^{n+1} + (1 - w_{j+1/2})(\rho e)_{j+1}^{n+1}]V_{j+1/2}^{n+1} - [w_{j-1/2}(\rho e)_{j-1}^{n+1} + (1 - w_{j-1/2})(\rho e)_j^{n+1}]V_{j-1/2}^{n+1}$$

$$+ \tilde{p}_j^{n+1}\frac{V_{j+1/2}^{n+1} - V_{j-1/2}^{n+1}}{\Delta t} = 0;$$

(D-37)

These mass and energy equations can be seen to be basically linear in ρ^{n+1} and $(\rho e)^{n+1}$, respectively, with a structure that is basically tridiagonal. For the loop flow problem, the general form of the mass equation can be written as

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & a_{1,8} \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,5} & a_{4,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} \\ a_{8,1} & 0 & 0 & 0 & 0 & 0 & a_{8,7} & a_{8,8} \end{pmatrix} \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \\ \rho_5 \\ \rho_6 \\ \rho_7 \\ \rho_8 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{pmatrix} \quad \text{(D-38)}$$

Lines have been drawn to indicate the current TRAC choice of network junction variables. For a given component junction, the associated network variable is the one adjacent to that junction in the first of the two adjacent components processed. The energy equation takes on a similar form, with the interesting property of using exactly the same matrix on the left side of the equation. TRAC takes advantage of this dual use of the coefficient matrix, using LU decomposition to reduce the solution time of the systems.

The above equation structure should be recognized as being the same as the one resulting from the stabilizer motion equations. The solution proceeds as described in Sec. D.1.1.1, with a few minor variations.

D.1.1.4. Considerations for 3D Solutions

When Vessels are present in current versions of TRAC, the above procedure is followed, with one key exception in each set of equations. When any Vessel variable (velocity, δp, ρ, or ρe) occurs in an equation, it is moved to the right-hand side with its coefficient, and all 1D variables are solved as functions of the unknown Vessel variables. These results are substituted as needed into the difference equations for the Vessel to give a closed set of equations that can be solved for all Vessel variables. Values for Vessel variables are back substituted into the 1D equations, and final values for all 1D unknowns are obtained.

Specific examples of this process are provided here for the system illustrated in Fig. D-2. As in Fig. D-1, cells are given "absolute" numbers rather than a combination of component numbers and cell numbers. For this example, cells numbered 1 through 5 are in a Pipe and cells 6 through 9 are in a 3D (collapsed to 2D here) Vessel.

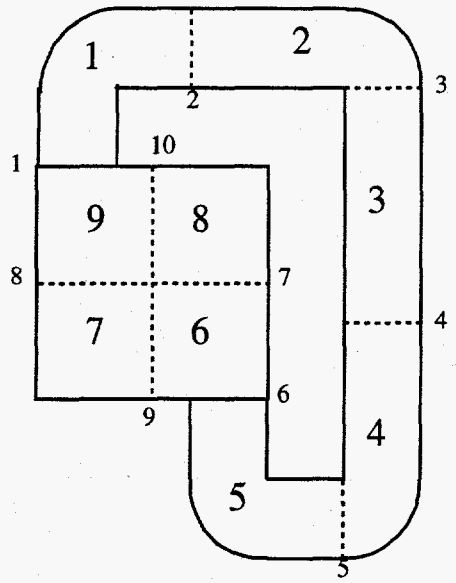


Fig. D-2. Flow loop with 2D Vessel.

The full system of stabilizer momentum equations for the flow loop in Fig. 2 is represented by Eq. (D-39). The last block in the coefficient matrix is associated with the radial velocities V9 and V10 and is completely isolated from equations for the axial velocities in the same 3D region. This reflects the fundamental structure of the 3D stabilizer momentum equations. For example, the axial stabilizer momentum equations evaluate contributions from axial velocities only implicitly. Radial and azimuthal velocities appearing in momentum transport terms are evaluated explicitly. This results in no coupling coefficients between velocity variables in the axial momentum block and those in the radial (or azimuthal) blocks.

Solution of the 1D portion of this system proceeds as before, isolating the 1D block as

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & 0 & 0 & 0 & 0 & a_{1,8} & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{4,3} & a_{4,5} & a_{4,6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{5,4} & a_{5,5} & a_{5,6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{6,5} & a_{6,6} & a_{6,7} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{7,6} & a_{7,7} & a_{7,8} & 0 & 0 \\ a_{8,1} & 0 & 0 & 0 & 0 & 0 & a_{8,7} & a_{8,8} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{9,9} & a_{9,10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{10,9} & a_{10,10} \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \\ V_9 \\ V_{10} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \end{pmatrix} \qquad \text{(D-39)}$$

and

$$\begin{pmatrix} a_{2,2} & a_{2,3} & 0 & 0 \\ a_{3,2} & a_{3,3} & a_{3,4} & 0 \\ 0 & a_{4,3} & a_{4,4} & a_{4,5} \\ 0 & 0 & a_{5,4} & a_{5,5} \end{pmatrix} \begin{pmatrix} V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix} = \begin{pmatrix} b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} - \begin{pmatrix} a_{2,1} \\ 0 \\ 0 \\ 0 \end{pmatrix} V_1 - \begin{pmatrix} 0 \\ 0 \\ 0 \\ a_{5,6} \end{pmatrix} V_6 \quad . \tag{D-40}$$

This is solved to obtain the equation:

$$\begin{pmatrix} V_2 \\ V_3 \\ V_4 \\ V_5 \end{pmatrix} = \begin{pmatrix} b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \end{pmatrix} + \begin{pmatrix} a'_{2,1} \\ a'_{3,1} \\ a'_{4,1} \\ a'_{5,1} \end{pmatrix} V_1 + \begin{pmatrix} a'_{2,6} \\ a'_{3,6} \\ a'_{4,6} \\ a'_{5,6} \end{pmatrix} V_6 \quad , \tag{D-41}$$

and these results are substituted into the junction equations to obtain:

$$(a_{1,1} + a_{1,2}a'_{2,1})V_1 + a_{1,2}a'_{2,6}V_6 = b_1 - a_{1,2}b'_2 - a_{1,8}V_8 \tag{D-42}$$

and

$$a_{6,5}a'_{5,1}V_1 + (a_{6,6} + a_{6,5}a'_{5,6})V_6 = b_6 - a_{6,5}b'_5 - a_{6,7}V_7 \quad . \tag{D-43}$$

Equations (D-42) and (D-43) give junction velocities as a linear combination of the "3D" velocities as

$$\begin{pmatrix} V_1 \\ V_6 \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_6 \end{pmatrix} + \begin{pmatrix} a'_{1,7} \\ a'_{6,7} \end{pmatrix} V_7 + \begin{pmatrix} a'_{1,8} \\ a'_{6,8} \end{pmatrix} V_8 \quad . \tag{D-44}$$

These two expressions are substituted into the 3D axial flow equations to obtain a final closed set of equations for the 3D axial velocities (V7 and V8). Once the 3D velocities are known, the 1D network junction velocities follow by back substitution, and the internal component velocities are obtained in a final stage of the back substitution.

A similar pattern follows for solution of the basic and stabilizer mass and energy equations. For the basic equations, the network junction variable equations analogous to Eqs. (D-34) and (D-35) are

$$(1 + c''_{11,1})\Delta p_1 - c''_{r1,1}\Delta p_5 = b''_{1,1} - \delta p_9 \qquad \text{(D-45)}$$

and

$$-c''_{15,1}\Delta p_1 + (1 + c''_{r5,1})\Delta p_5 = -b''_{5,1} + \delta p_6 \qquad \text{(D-46)}$$

These equations are solved for the network junction variables as functions of the 3D pressure variations:

$$\Delta p_1 = b'''_{1,1} + a''_{1,6}\delta p_6 + a''_{1,9}\delta p_9 \qquad \text{(D-47)}$$

and

$$\Delta p_5 = b'''_{5,1} + a''_{5,6}\delta p_6 + a''_{5,9}\delta p_9 \qquad \text{(D-48)}$$

The semi-implicit nature of these basic equations means that the linkage between 3D blocks and 1D blocks occurs only through junction variables such as Δp_1 and Δp_5. This is a result of the fact that the only new-time terms shared by adjacent cells are the velocities, which in turn depend only on the junction variables (Δp's). As a result, substitution of Eqs. (D-47) and (D-48) into the 3D equation block produces a closed set of equations that may be solved for final values of all 3D unknowns. Back substitution of 3D pressure variations into Eqs. (D-47) and (D-48) provides final values for network junction variables, which in turn are substituted into the analog of Eq. (D-41) to yield final values of 1D variables.

D.1.2. Software Implementation

New subroutines will be created to perform the full solution of the linear systems generated by the approximations to the flow equations. These will replace statements in

- Femom, Femomx, Femomy, Femomz, Tf1Ds, and Stbme, which begin the solution of the 1D portion of the linear equations, including all local coding for solution of the local tridiagonal matrix structures and generation of terms in the network matrices;
- Tf1Ds and Tf3Ds for cell block reduction;
- Out3D, Post3D, Prep3D, and Vssl2 for Vessel matrix solution;
- Outer, Post, and Prep1D for solution of the network matrix; and
- Bksmom, Tf1Ds3, Tf3Ds3, Bksstb, and Bksstb3, which are directly related to back substitution steps in the solution of the linear equations.

The basic and stabilizer equations involve very different numbers of equations and generate two different matrix structures. As a result, two separate subroutines will be created for solution of global systems of linear equations. The more basic of these, Solver, operates on the stabilizer equations and is described in the following section. The more complex linear system resulting from the Newton iterative solution of the basic equations is solved by "blockSolver", which is described in Sec. D.1.3.2. Descriptions provided in these sections are for planning purposes only. Later studies on timing and memory usage or special needs for parallel processing may change the details of the final implementation.

D.1.2.1. Subroutine Solver

The interface to this subroutine is relatively simple. It uses the module Matrices and so has full access to this data structure. Only two arguments are passed:

1. an abbreviated name for the array of independent variables, and

2. an optional argument set to 'factored' when the coefficient matrix has already been factored by a previous call to Solver.

One example of using this subroutine is the solution of the stabilizer mass and energy equations driven be subroutine Post. The following code would be inserted just before the end of the DO loop on "ibks":

```
IF (ibks.NE.1) CYCLE
 CALL SOLVER('arl')
 CALL SOLVER('arel','factored')
 IF (isolut.ne.0) CALL SOLVER('arc','factored')
 CALL SOLVER('arv')
 CALL SOLVER('arev','factored')
 CALL SOLVER('ara','factored')
ENDIF
```

This choice was made to permit a single point within Solver for association of auxiliary arrays needed by solution methods and to permit transferring of knowledge of the data structure to a lower level for parallel methods based on distributed memory machines.

Selection of arrays to be used in the actual solution will be via pointer association. As an example, the initial implementation will contain allocatable arrays in Matrices such as

```
    TYPE (sparseMatrix), ALLOCATABLE, TARGET :: al(:), ag(:)
REAL, POINTER, DIMENSION (:) :: arlS, arvS, &
&     arelS arevS, araS, arcS, vvtS, vltS, arlRHS,   &
&     arvRHS, arelRHS arevRHS, araRHS, arcRHS,      &
&     vvtRHS, vltRHS
```

```
INTEGER, POINTER, DIMENSION (:) :: splitRowsC, splitRowsE
INTEGER, POINTER, DIMENSION (:) :: splitRows
    TYPE (sparseMatrix), POINTER :: at(:)
REAL, POINTER, DIMENSION (:) :: rhs(:), ans(:)
```

Operations within Solver will be on generic arrays such as "at", "rhs", and "ans", which will be associated at the beginning of the subroutine based on the contents of the first dummy argument "varname". For example,

```
SELECT CASE (varname)
 CASE ('arl')
  at => al
  rhs => arlRHS
  ans => arlS
  splitRows => splitRowsC
 CASE ('vvt')
  at => ag
  rhs => vvtRHS
  ans => vvtS
  splitRows => splitRowsE
...
END SELECT
```

Following this initial decision on array usage, the solution procedure proceeds with the steps outlined in Sec. D.1.1.1. The array splitRows is used to divide the 1D problem into a set of tridiagonal blocks [see Eq. (D-12)]. These block systems are solved and coefficient arrays stored for later back substitution. A substitution of these results is made into the splitting rows to generate the "network" equation system, which is solved with calls to Sgefat and Sgeslt. If 3D components are present, solution of the network equations involves the generation of coefficient arrays multiplying undetermined 3D variables. In this case, a section of code is used to substitute these network results into the 3D equations, and the 3D equations are solved for final values of 3D variables. The initial implementation of Solver will use the existing TRAC Capacitance Matrix coding and data structure to handle the storage and solution of the 3D portion of the problem.

All three major equation solution stages just outlined use LU factorization and store sufficient information so that the factorization need not be repeated. When the subroutine is called with the optional dummy argument "factored" present, processing jumps immediately to the back-substitution step of the 3D solution, then proceeds through back substitution of the network equations and of the initial tridiagonal systems to obtain the final values for the variables (stored in "ans").

The initial implementation of this subroutine will contain equations in exactly the same form as those in current versions of TRAC-M. Implementation of the solution steps above will be programmed to match the reduction and back-substitution coding currently scattered through TRAC as closely as possible to produce minimal perturbation on test problem results. However, because the data structure is somewhat different and the grouping of solution statements has changed, compilers will react to this new implementation differently when producing machine code and results cannot be expected always to match previous solutions to the last bit. The variation of any test results should not be any greater than those produced by using a different compiler or a different optimization level on the same compiler.

Although the solution procedure is formally the same as current code versions, separating the solution steps produces immediate opportunities for more parallel execution. In current versions, the contents of subroutines such as Femomx, Tf3Ds, and STBME3 had to be executed after all similar 1D subroutines. After this modification, these subroutines with their reduced scope of activity could be executed in parallel with 1D subroutines. A second version of Solver will provide the opportunity for additional parallel computation within the solution process. The order of equation reduction will be altered so that operations on the sparse blocks associated with 3D components can be performed at the same time as those for the tridiagonal blocks associated with 1D components. Only the solution of the network matrix will remain as a serial step. The greatly reduced amount of information required by the network matrix will make solving the preceding steps very amenable to a distributed memory environment.

D.1.2.2. Subroutine BlockSolver

Subroutine "blockSolver" communicates entirely through the module Matrices. It has no argument list and thus, none of the special pointer assignments that begin Solver (see previous section).

Solution begins with a block reduction in a loop over all elements of the derived-type array "blocks" (type blockMatrix). This covers the cell block reduction of the linearized basic equations for all cells in the system (1D and 3D). A special derived-type array is set by the subroutine Sparcity to define the splitting of the reduced array into tridiagonal blocks. Array "triBlock" is allocated by the statement

ALLOCATE [triBlock(2,nTriBlocks)] ,

where "nTriBlocks" is the number of "network components" (total number of system components plus the number of Tee's and any other Tee-like components). This array is of derived-type netIndices defined as follows:

netIndices - a derived type providing information about indices needed to (1) locate tridiagonal submatrices within system-wide coefficient and constant arrays, and (2) couple them to the appropriate network matrix.

ilow - index in the system-wide array that starts the submatrix.

ihigh - index in the system-wide array that ends the submatrix.

netNum - index of the network matrix (or loop index) associated with this submatrix.

netLow - index in the network matrix associated with the network variable directly coupled to the low end of this tridiagonal submatrix.

netHigh - index in the network matrix associated with the network equation and variable directly coupled to the high end of this tridiagonal submatrix.

netTee - index in the network matrix associated with the network equation and variable directly coupled to a Tee junction within this tridiagonal submatrix (this will disappear in later versions of the Solver, provided now for consistency with the existing Solver).

The blocks of 1D tridiagonal pressure equations are solved as outlined in Eqs. (D-26) through (D-36). Substitution into the network junction in Eq. (D-32) is governed by two other index arrays created by Sparcity. The derived-type array "junVars" contains information on the ith network junction and is allocated by the statement:

ALLOCATE [junVars(nsplitsE)] ,

where "nsplitsE" contains the number of network junctions in the system. Its "netVarInd" type is defined as

netVarInd - a derived type providing indices to elements attached to the "positive" and "negative" sides of a network junction. Components are

pos - index of an array element to the positive side the network junction and

neg - index of an array element to the negative side of the network junction.

One more array with type netVarInd is needed to provide the indices of adjacent elements in the component "cDpp" of "blocks" containing pressure coefficient information. This array "junCoef" is allocated by the statement

ALLOCATE [junCoef(nsplitsE)] .

These indices are used at each junction to obtain the information needed for substitutions that move from equations such as those represented in Eq. (D-31) to the network system analogous to Eqs. (D-34) and (D-35). These network equations are solved to give the network variables as linear functions of the pressures in any 3D cells adjacent to the network junctions, using calls to Sgefat and Sgeslt. A block of code follows to substitute network variable information into the 3D pressure equations and

transfer 3D pressure coefficients from the blockMatrix data structure to the structure associated with the 3D Capacitance Matrix solution method. The job of solving the 3D equations is passed to a revised version of the existing TRAC-M subroutine Matsol.

Results of the 3D solution are back substituted to provide final values for the network variables (Δp's). These are back substituted into the reduced tridiagonal systems to provide pressures in all 1D cells. Finally, pressures are back substituted into the reduced cell block equations, using components "bp" and "cDpp" of "blocks", to obtain final values of all independent variables.

As with Solver, the above series of steps is designed to match the current solution stages carefully in TRAC-M. The same comments on the matching of test results apply.

It is useful to compare the pressure equations occurring in this subroutine, immediately after cell block reduction with the equations produced by the stabilizer mass and energy equations. They have the same basic structure. As a result, a second version of "blockSolver" will be created that contains the initial cell block reduction for the entire system and passes the work of solving the pressure equations to Solver. This will reduce the maintenance points in the program and concentrate the most complex programming problems associated with parallel implementations into one location (Solver).

D.1.2.3. Implementation of Velocity Solution

Within the current code, the order of the velocity variable array is assigned effectively by the subroutine Srtlp (called by Input). The partition lines shown in the previous equation are established by Setnet during initialization. The revised code would continue to use Srtlp to establish basic array order. However, more information will need to be stored. Data are needed within each component for the subscript in Eq. (D-12), corresponding to the stabilizer velocity at each cell face. Component subroutines will place these subscript (index) values in the array "edgeIndex", within the component data structure (module Gen1DArray for 1D components, PlenArray for the Plenum, and the equivalent in the Vessel). Subscript values will be stored at the same point in the program for cell-centered variables used in mass and energy equations described below. These will reside in an array named "centerIndex". In the revised code, partition lines will be produced by Sparsity and by indices of the rows dividing blocks placed into a dynamically allocated pointer array named "splitRowsE" ("E" for edge) contained in the module "Matrices". A similar array named "splitRowsC "will be created to mark matrices related to cell-centered variables.

The subroutine Sparsity also will have the job of storing indices for the off-band coefficients in the above matrix and for related matrices. These will be stored in arrays with the type "sparseMatrix", defined as follows:

```
TYPE sparseMatrix
REAL :: a(bandWidth)
INTEGER, POINTER, DIMENSION(:) :: index
INTEGER :: nOffBand
```

```
REAL, POINTER, DIMENSION(:) :: aob
END TYPE sparseMatrix
```

In this derived-type "bandWidth" is a parameter residing in module Matrices. The array "a" contains coefficients along the primary band of the matrix, and the array "aob" contains the "off-band" coefficients. The array "index" contains the column index for the corresponding coefficient in "aob". The number of off-band coefficients is stored in the integer "nOffBand". For the current difference equations, two allocatable "sparseMatrix" arrays will be placed in module Matrices, "al" for liquid and "ag" for gas. If necessary for future difference methods, this derived type can be cloned to produce types with more than one bandwidth or altered so that component "a" is an allocatable pointer. The choice of fixed-dimension "bandwidth" was made based on the fixed structure associated with a given difference method and timing results on the use of allocatable pointers within derived types (see the Data Structure Software Design and Implementation document).

Femom contains the LU decomposition loops to produce the results in Eqs. (D-16) and (D-17). It stores the "b' " vector coefficients in the variables "vlt" or "vvt" (as appropriate) and the "a' " coefficients in the array "trid" for later back substitution in Bksmom.

After revisions for modular solution, Femom will only create and store the coefficients for each element in the appropriate "sparseMatrix" derived type array, and Bksmom will transfer solution values only from system-wide arrays ("vltS" and "vvtS") in module Matrices back into the component data structure. The job of solving the full set of linear equations is passed to the subroutine Solver, which is called from subroutine Prep at the end of the first pass through the loop on "ibks".

In the current version of TRAC-M, the coefficients of V_1 and V_5 in Eqs. (D-18) and (D-19) are built with statements in Femom assigning elements in the "aol" or "aov" arrays. Because Femom is called from components, the entirety of one coefficient cannot be evaluated in a single call. When the first Pipe is processed in our example, contributions for which $a_{1,8}$ and $a_{5,6}$ are factors cannot be completed because information on the motion equations at cell faces 6 and 8 are not available. As a result, $a_{1,8}$ and $a_{5,6}$ are stored in the network coefficient array "od", and the job of completing the values in "aol" or "aov" is completed when Femom is called by the other Pipe.

After the separation-of-solution procedure, the final solution step is much less complicated. Subroutine Solver has access to all necessary information to generate each coefficient in Eqs. (D-18) and (D-19) in a single step. Solution of these coupled equations will be accomplished initially with the same Linpack-based subroutines (Sgefat and Sgeslt) used in TRAC-M for this system.

D.1.2.4. Implementation of the Solution of the Basic Equations

The current version of TRAC generates the coefficients $a_{j,i,k}$, $b_{j,i}$, $c_{lj,i}$, and $c_{rj,i}$ in Eq. (D-26) within subroutine Tf1Ds, storing them temporarily in the arrays "a" and "c". These

arrays are overwritten by each cell, and the subroutine saves only the primed coefficients resulting from the solution of each cell's block system. Arrays for thermodynamic variables and derivatives, not needed until the next iteration, are used to store these coefficients.

Separation of the solution procedure from Tf1Ds will utilize an array with type "block-Matrix" in the module "Matrices" for the communication coefficient information to the subroutine "blockSolver".

```
TYPE blockMatrix
REAL :: a(ncvars,ncvars), b(ncvars), bp(ncvars)
INTEGER, POINTER, DIMENSION(:) :: index
REAL,  POINTER, DIMENSION(ncvars,:) :: cDp, cDpp
END TYPE blockMatrix
```

The number of independent variables per cell is an integer parameter "ncvars" contained in "Matrices". Arrays "a" and "b" match the usage above, and "bp" is the primed "b" array. The array "cDp" contains the coefficients of the Dp terms associated with the cell, and "cDpp" contains the primed coefficients obtained after solution of the cell's block system.

The pressure Equations [(D-27) and (D-28)] in tridiagnal form currently are solved within Tf1Ds to obtain the pressure variations. This function will be moved to a subroutine "triSolve", called from "blockSolver".

The equations for the network variables [Eqs. (D-34) and (D-35)] currently are built within Tf1Ds. As with the stabilizer motion equations, terms for each of these equations must be contributed from calls to Tf1Ds by both components adjacent to the junction. Solution of the equations is performed within the subroutine Outer, through calls to Sgefat and Sgeslt. Construction and solution of these equations will be moved to "blockSolver".

D.1.2.5. Implementation of the Stabilizer Mass and Energy Equation Solution

The structure of the stabilizer mass and energy equations should be recognized as being the same as the one resulting from the stabilizer motion equations. Operations associated with Femom now are associated with Stbme, and those associated with Bkmom are done in Bksstb for the mass and energy equations. This similarity will be recognized in the modularized solution by using the same data structure and solution package for all stabilizer equations.

D.1.2.6. Implementation of the 3D Solution

Solution for 1D variables as a function of unknown 3D variables is driven by Prep1D for the stabilizer velocity solution, by Outer for the pressure solution of the basic equations, and by Post for the stabilizer mass and energy equations. Solution of the Vessel equations is done in Prep3D for stabilizer velocity equations, by Vssl2 (or Out3D for multiple Vessels) for the pressure equations, and by Post3D for the stabilizer mass and

energy equations. Back substitution of Vessel information into the 1D equations is driven by Prep1D for the stabilizer velocity solution, by Outer for the pressure equations, and by Post3D for the stabilizer mass and energy equations. In the modularized solution, the 3D equations will be just blocks in the full system matrix and will be treated within Solver or "blockSolver" as appropriate. In the first implementation, reduction to and back substitution from the 3D blocks in Solver and "blockSolver" will be identical to the form used in the original program. Once this is tested, a modified version will be created in which reduction of the 3D block of equations is not dependent on completion of the 1D equation reduction.

D.1.3. Test Plan for Linear System Solution

Testing of the solution modifications will proceed in three phases. The subroutines Solver and "blockSolver" initially will be debugged in isolation from TRAC-M. A small driver will be written to create matrices in a number of configurations, within the data structure of module Matrices. The test problems will result from selection of an answer and multiplication of the matrix by the answer to generate a right-hand side for use in the solution subroutine. Results of the solution procedure will be printed adjacent to the initially selected answer, along with a measure of error.

Initial installation of the Solvers in TRAC-M will be tested with some very simple test problems. The first of these will be similar to the loop in Fig. D-1, but using a pump component to replace one of the pipes as a momentum source. The second will create a loop from the primary legs of three tees, with injection from fills on two side legs and outflow through the remaining side leg. Variations of this will be created to check proper treatment of Tee momentum source terms when side legs join at the first, last, or only cell of the primary leg. A third series will add a simple Vessel to the pump loop. All of these simple tests will have versions that are single-phase gas, single-phase liquid, and bubbly two-phase flow. In principle, the single-phase tests exercise all of the code, but the two-phase tests are needed to check for bugs that introduce false communication between the equations for each phase. All tests should produce identically printed results in codes with and without modifications. The two-phase versions will be watched with a debugger to check the results to machine precision.

The third level of testing will be a comparison of results for the full TRAC developmental assessment matrix (as used for official TRAC-M releases). Printed results generally should match, but differences should be expected for some sensitive problems (e.g., reflood). In these instances, tests will be run to compare these deviations with deviations experienced from the use of different compilers and different optimization levels.

This three-stage test procedure will be used for the initial versions of these Solvers and for later revisions introduced to permit a parallel version of the program.

D.2. Improved Intercomponent Communication

A request-driven communications method will be created based on requests from components or external programs for specifically named variables. The variable names will be passed as ASCII strings. In this initial phase of development, these requests will be made only during the initialization phase of a calculation. However, provisions will be made to permit a dynamic communication process, where the list of variables requested by a given component can occur at any time during the calculation. This will be useful in interactive simulations or for dynamic linking to other programs.

Information transfer has been designed to permit a "read-only" transfer of information. The communication system intentionally prevents direct access by the requesting component to the storage of the requested information in the adjacent component. This is an attempt to localize errors in new components and limit poor programming practices involving alteration of data by unexpected portions of the program. It also lays a groundwork for parallel processing, providing values of communicating variables that are updated only at well-defined synchronization points in the execution of the program.

No global storage will be created to hold component boundary information as is now the case with the component boundary array (bd). It is the responsibility of the requesting component to provide space for the transferred data. For example, consider the initialization process of a problem containing a Pipe component 10 with junctions numbered 1 and 2. To obtain one necessary piece of boundary information, the pipe will ask the communication service for information named "DX" from the first cell beyond junction 1, to be placed in the variable BD1%DX contained in the component 10 data structure. The communications service will consult the data structure to locate the component and cell that is the first beyond junction 1. It will use a pointer location subroutine to determine what variable is associated with the ASCII name "DX". Finally, the service will make pointer associations within the service data structure to the appropriate source element of the DX array and to a destination BD1%DX. These source and destination pointers will reside in a derived-type array and will be used directly during the calculation for transfer of information.

The key to flexibility will be low-level subroutines available to perform operations such as the one described above. These subroutines will not be restricted to obtaining information from the cell immediately adjacent to a junction and will, for example, respond correctly to a request for information in the third cell past a junction for use in a higher-order numerical method. Design of the communications also will permit its use for moving heat structure and control block information, but full implementation of these capabilities is not included in the level of effort for the initial task. In addition, the structure will be designed for later support of parellel virtual machine (PVM) requests for information and transfer of that information via PVM.

D.2.1. Location of Component Junctions

A component must register its flow connections with the system services to permit

correct intercomponent communication. This currently is accomplished within input and restart subroutines (Rpipe, Repipe, etc.) by filling in entries to the JUN array. The revised registration involves passing information to a junction cell data structure with a subroutine call. In this context, registration is required for both standard intercomponent junctions and intracomponent junctions, such as the junction of a Tee side leg to the primary leg. The subroutine doing the work is "junctions" and has the following interface:

SUBROUTINE junctions (compNum, cellNum, junNums , vOutSign, theta, phi, dist, ncAdj)

where the following definitions hold:

compNum	-	input component number for the cell with this junction;
cellNum	-	number for the cell containing the junction to another component (or to the other section of the same Tee);
junNum	-	input component junction number, or generated junction number, for an internal connection;
vOutSign	-	the sign of the velocity associated with flow out from the cell through this junction face (+1 or -1);
theta	-	the angle (degrees) between an inwardly directed-normal-to-the-junction face and the primary positive direction of motion within the component;
phi	-	the angle (degrees) between an inwardly directed-normal-to-the-junction face and a reference vector perpendicular to the primary positive direction of motion within the component;
dist	-	the distance between the cell center and the junction face;
ncAdj	-	number of cells in this component adjacent to the junction face in the direction of the inward normal-to-the-junction face.

When calculating theta in a Vessel, the primary positive direction of motion is taken to be the positive z direction. The reference vector for computing phi is taken to be pointing toward the center of the Vessel. This results in values of phi is

1. zero for a connection in from the outer radial cell face,
2. 90 degrees for a connection in from the high-numbered cell theta face,
3. 180 degrees for a connection from an inner radial cell face, and
4. 270 degrees for a connection from the low-numbered cell theta face.

For registration of an intracomponent junction such as a Tee-side-leg connection, a unique junction number must be generated. This is accomplished with a reference to the function "interiorJunNum", which returns a new unique (and negative) number with each call. For example, in a Tee component, the following coding would be appropriate for registration associated with the side leg:

```
      junSide=interiorJunNum()
      dist=.5*wjcell(jcell,cost,g1dAr(cci)%hd,g1dAr(cci)%dx)
      angle = acos(cost)*180/pi
      CALL junctions (num, jcell, junSide, 1, angle, 0, dist, 1)
    CALL junctions (num, ncell1+2, junSide, -1, 0, 0,  &
.5*g1dAr(cci)%dx(ncell1+2), ncellt-ncell1-1 )
```

The subroutine "junctions" installs the information from the dummy argument list into a derived-type array for further processing to index and locate boundary information:

```
TYPE junctionCellsT
 INTEGER cco, compNum, cellNum, juncNum, vOutSign, otherSide
 REAL theta, phi, cosTheta, dist
END TYPE junctionCellsT

TYPE (junctionCellsT), ALLOCATABLE, TARGET :: junCells(:) ,
```

where the components of the type have definitions matching those given above for arguments plus the following :

cco	-	index for the component in ordered arrays [e.g., g1dAr(cco)];
otherSide	-	index of the element in junCells containing information on the other side of this junction; and
cosTheta	-	cosine of theta.

D.2.2. Transfer of Component Boundary Information

During initialization, a component can set up for information transfer on several different schedules. Transfer is scheduled for calculation setup only, either once per timestep or once per cycle through components. Variables containing fixed geometry or index or flag information are transferred only during the initialization phase. This transfer occurs at every pass through components during initialization but does not continue beyond the start of the first timestep. Some variables become "old-time" quantities simply by transfer of the "new-time" value from the previous timestep. These are scheduled for transfer at the beginning of each timestep. Of the remaining variables, some may be generated only once during a specific phase of a timestep. However, modifications to numerical methods may alter the points at which such variables are recalculated. To retain maximum flexibility, this information is transferred after each cycle through all components. Consideration can be given to further refinement of the scheduling after the advanced code reaches a higher level of maturity.

Most flow of information is as boundary values requested by a component. This is scheduled through calls to the subroutine "GetBDfor" during initialization. Its interface is in the form

SUBROUTINE GetBDfor (compNum, juncNum, offset, varName, localStore),

where the following definitions hold:

compNum	-	input component number for the requesting component;
junNum	-	input component junction number or generated junction number for an internal connection;
offset	-	number of cells or faces that the desired information is offset to the other side of the junction face (currently 1 for cell center information and 0 or 1 for cell face information);
varName	-	name (or alias) of the variable containing information needed;
localStore	-	a pointer to local storage.

This subroutine will have a generic interface to accept the pointer "localStore" as either a scalar or vector Integer or Real type. It will use low-level pointer assignment subroutines such as "GetArrayPointer" to associate the variable name (varName) with a pointer to the appropriate memory location and obtain a flag indicating whether the variable is to be transferred at the beginning of the calculation, beginning of the timestep, or on each cycle. A sorted version of "junCells" will be used to trace the location of the information.

The result of calling GetFor is one or more entries in one of six communications tables. These are established by the following type definitions.

```
TYPE transferRealT
   REAL, POINTER, source, destination
END TYPE transferRealT
TYPE transferIntT
   INTEGER, POINTER, source, destination
END TYPE transferIntT
TYPE (transferRealT), ALLOCATABLE :: transOnce, transStep, transCycle
TYPE (transferIntT), ALLOCATABLE :: itransOnce, itransStep, itransCycle
```

At appropriate points in the code, a very simple subroutine is called that loops through the elements in one of the transfer arrays making the necessary assignments.

```
SUBROUTINE oneTransfer
IMPLICIT NONE
INTEGER i
DO i =1,nTransOnce
  transOnce(i)%destination = transOnce(i)%source
ENDDO
RETURN
END
```

This subroutine is contained in the same module as the data structure and thus does not require a USE statement to access transOnce.

D.2.3. Communication to Components in Other Processes

Once basic intercomponent communication is functioning for standard TRAC problems, a class of components will be created named "Exterior". Input for the component will simply list the component number and a table of junction numbers and associated connection information. Connection information will include the cell to which the junction connects and whether the exterior component is responsible for computing the velocity at that face. The Exterior component subroutines and associated data structures will be very minimal, providing code for negotiating and passing boundary information to and from a parallel task that performs the detailed calculations for the actual Exterior component (or block of components).

Communication between the Exterior component and its connection point in a parallel code will in some ways be similar to the transfer initiated by "GetFor". The initialization routine (IExterior) will pass out a request for an ordered data stream giving the ASCII variable name and location for each item required, along with a request for frequency of transmission. If the target program uses different nomenclature for physical variables, an intermediate translator program will intercept this request and pass it on to the target program. The initialization routine also will receive a data request stream from the target (translated if necessary) and schedule data transmissions.

D.2.4. Communication for Heat Structures, Control Blocks, and Related Models

Once the above communications are functioning, the next task will be to create similar table-driven transfers for heat structures, signal variables, control blocks, and the radiation model. These components have direct knowledge of the connecting component and cell numbers. Table-driven transfer could be created by calls to initialization subroutines GetFrom and PutTo that directly reference the component and cell numbers from or to which information is to be transferred. Transfers would in some instances require different scheduling than discussed above.

D.2.5. Testing of Communications

The second and third phases of the test set described in Sec. D.1.3 should cover all of the communications functionality. These test problems will be reviewed to ensure that all component types are tested for all flows that include liquid and gas phases, noncondensables, and solute.