RETRAN-3D — A Program for Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems

Volume 2: Programmer's Manual

NP-7450(A), Volume 2, Revision 10
Research Project 889-10

Computer Code Manual, September 2014

Prepared by
COMPUTER SIMULATION & ANALYSIS, INC.
410 Memorial Dr., Suite 205
Idaho Falls, ID 83402

Principal Investigators
M. P. Paulsen
C. E. Peterson
G. C. Gose
J. H. McFadden
J. G. Shatford
J. L. Westacott
H. J. Kadakia

EPRI • 3412 Hillview Avenue, Palo Alto, California 94304 • PO Box 10412, Palo Alto, California 94303 • USA
800.313.3774 • 650.855.2121 • askepri@epri.com • www.epri.com

ORGANIZATION(S) THAT PREPARED THIS DOCUMENT

Computer Simulation & Analysis, Inc.

NOTE

For further information about EPRI, call the EPRI Customer Assistance Center at 800.313.3774 or e-mail askepri@epri.com.

REVISION 10 CHANGE PAGES FOR EPRI NP-7450 VOLUME 2
RETRAN-3D PROGRAMMER'S MANUAL

The pages listed below are the replacement pages that comprise Revision 10 of the RETRAN-3D Programmer's Manual, Volume 2. These pages are either replacements pages for those with the same page number in Revision 9 of the Programmer's Manual, or additional pages for those with new page numbers. Those pages which have changes are indicated by "Revision 10" in the lower, outside corner of the page, and the change is indicated by a vertical bar adjacent to the change, along the outside margin of the page.

Section	Pages	Description of Changes
	Title Page	Changed revision number and date
	viii, ix, xiii	Update Contents, Illustrations, & Tables
II	II-1, II-2	Changed supported platforms.
IV	IV-49	Changed supported platforms.
VI	VI-1-VI-4, VI-11, VI-13 VI-14, VI-20, VI-22 VI-24-VI-27, VI-37, VI-38	Changed supported platforms.
B	B-1, B-2	Changed supported platforms
C	C-1, C-2, C-25-C-60	Added code modifications for MOD004.7.1.

PROPRIETARY INFORMATION - WITHHOLD UNDER 10 CFR 2.390

ABSTRACT

RETRAN-02 has proven to be a versatile and reliable computer program for use in best-estimate transient thermal-hydraulic analysis of light water reactor systems. The RETRAN-3D computer program is an extension of the RETRAN-02 program designed to provide analysis capabilities for (1) BWR and PWR operational transients, (2) small break loss-of-coolant accidents, (3) anticipated transient without scram, (4) long-term transients, (5) transients with thermodynamic nonequilibrium phenomena, (6) mid-loop operation with noncondensable gas present, (7) transients where 3-D dimensional power shapes and reactivity feedback effects are important, and (8) BWR stability events. RETRAN-3D also maintains the complete analysis capabilities of RETRAN-02.

The RETRAN-3D computer code is written in standard Fortran 95 to provide for ease in maintenance and installation on new computer platforms. It uses dynamic memory allocation to assign memory at execution time; thus, memory use is tailored to the problem being executed.

This report (the second of a four-volume computer code manual) describes the programming aspects of the RETRAN-3D code. The three companion volumes describe the theory and numerical algorithms; the required input, output, and sample problems; and the verification and qualification for RETRAN-3D.

Abstract

ACKNOWLEDGMENT

The RETRAN-3D Programmer's Manual has the programming conventions and practices used for the RETRAN-3D code and its implementation using the Fortran 95 language.

The vision, guidance, and encouragement offered by Lance Agee, the EPRI RETRAN project manager during the development of RETRAN-3D, and other members of the EPRI staff are gratefully appreciated. The participation of the RETRAN User Group members has made the continued development and maintenance of RETRAN-3D possible.

The preparation of a document like the RETRAN-3D Computer Code Manual requires the assistance of many individuals. We are grateful to Pam Richardson for editing and typing this report. Our final acknowledgment is given to all of the CSA staff for their support and interest in the RETRAN project.

Acknowledgment

CONTENTS

Section **Page**

Section **Page**

Contents

ILLUSTRATIONS

TABLES

I

INTRODUCTION

The RETRAN computer programs are a series of best-estimate transient thermal-hydraulic analysis program used to analyze the behavior of light-water reactors or experimental facilities subjected to postulated transient conditions. RETRAN-3D is the latest result of an extensive code development effort sponsored by EPRI since 1975. The objective of this development effort has been to provide a versatile and reliable thermal-hydraulics code that can be used for best-estimate analysis of light-water reactor systems. The RETRAN-01 code,[I-1] was released in December 1978, RETRAN-02 was released in April 1981,[I-2] RETRAN-03[I-3] in 1991, and now RETRAN-3D which represents the results of recent development efforts focusing on applications affecting current operational issues. These include midloop operations with noncondensable gas present, multidimensional power and reactivity feedback events, and BWR stability events.

In 2007 the RETRAN User Group (RUG) initiated a development effort to convert the RETRAN-3D source code to Fortran 95. The intent of this code modernization effort was to improve the long term viability of RETRAN-3D by replacing or eliminating obsolete programming and language features, using new language constructs that simplify the structure of the code, and the elimination of obsolete programming techniques. Specifically, the FTB variable dimensioning feature that had been used to implement dynamic dimensioning was replaced with Fortran 95 features for dynamic memory allocation. Other source code revisions eliminated obsolete coding constructs and replaced them with Fortran 95 features with the intent to improve the long term use of the code with new compilers and operating systems and to also reduce future maintenance costs.

As a result of the conversion effort, the RETRAN-3D source code is now written completely in Fortran 95. The SLIB77 program that had been used to maintain the source code is no longer used. The contractor responsible of on going development and maintenance of the code is using a version control program to maintain the revision history for the code. Prior to the Fortran 95 version, three separate SLIB77 program libraries were use to maintain the code. They were also included with source code transmittal packages. Source code from the three program libraries has been combined into a single source code archive. This Fortran 95 source code is now included with the source transmittals.

Programming guidelines were developed as part of the conversion effort. They are discussed in Section II.1.1 and will allow different programmers to produce uniform and consistent source code that will be easier to maintain and revise in the future. Major implementation changes were made in the following areas

- dynamic memory allocation,
- restart file structure and content,
- major edit subroutines, and
- minor edit variables.

Discussions of each of these implementation changes are given in Section II.

With the conversion to Fortran 95, RETRAN-3D can be easily transported to a variety of computer platforms, a development requirement long imposed by EPRI and the RUG.

Topics presented in this document have little to do with the theoretical basis of the RETRAN-3D computer program or actual use and application of the code, but rather, deal with the programming aspects of the code package. Specifically, RETRAN-3D utilizes dynamic memory allocation to allow for efficient usage of main memory storage, where the main memory required for any given problem depends on the nodalization detail used for the problem and the RETRAN-3D modeling options that are used.

The Programmer's Manual discusses the programming techniques used to implement dynamic memory allocation, as well as coding conventions used to ensure that the code package is transportable to a variety of computing platforms and operating systems. A discussion of each I/O unit used by the code package is presented with particular emphasis placed on the structure and use of RETRAN-3D data files (restart files or data sets). Also contained in the manual is the general program flow from the executive subprogram, RMAIN, to each of the program modules, and program flow within each module. Installation and maintenance of the code package are also covered.

II

PROGRAMMING PRACTICES

The RETRAN-02 and RETRAN-3D transient thermal-hydraulics analysis codes have been widely used as analysis tools by the nuclear power industry in support of licensing applications and operational issues. RETRAN-3D is the most recent of these codes. It is written entirely in Fortran 95, thus facilitating RETRAN-3D installation on a variety of microcomputers and workstations using various Fortran 95 compilers and a single source. Currently supported platforms include personal computers running Windows or Linux workstations and related Fortran 95 compilers. Note that HP, IBM, and SUN workstations using proprietary versions of UNIX and FORTRAN 95 are no longer formally support. However, the UNIX options have been retained so the installation and execution scripts should work for these platforms with minimal change.

This section provides specific guidance regarding the use of the programming language, memory allocation, defining minor edit variables and the content of the restart file.

1.0 PROGRAMMING LANGUAGE

RETRAN-3D source code is written entirely in Fortran 95 and uses the free form source code feature provided by the language that allows up to 132 columns to be used for programming content. While coding can begin in column 1, the RETRAN-3D source uses the convention where statement or format labels are right justified in columns 1 through 5, column 6 is not used, and programming instructions begin in column 7 unless indentation is used on the line.

The RETRAN-3D source code is generally written in lower case. Upper case can be use as needed for format statements to provide the desired output. Upper case can also be used to help make comments more readable, e.g., sentences should begin with a capital letter. Most output formats are written in upper case. New additions or revisions should be done in a manner consistent with the existing code.

Programming guidelines were established for use with the RETRAN-3D in an attempt to standardize the content and structure of the RETRAN-3D source code. This should aid programmers as they either modify or maintain the code. These are summarized in the following sections, which also include programming constructs that should be avoided.

1.1 Programming Guidelines

Programming guidelines have been established for use with the RETRAN-3D code in an attempt to standardize the structure and content of the new Fortran 95 source code. This will aid

programmers as they develop new models, modify existing models or maintain the code. The guidelines are summarized in the following sections. Obsolete programming constructs that should be avoided are also included.

By following the programming guidelines, different programmers will be able to produce uniform and consistent source code that will be easier to maintain and revise in the future. This will require programmers to follow the intent of the guidelines and also use them in a manner consistent with existing coding style (when in Rome, do as the Romans do). The goal is to keep the source appearance and structure as if it had a single author rather than many authors with disparate programming styles.

With free form source code,

- variable names can be up to 31 characters in length. Make variable names meaningful in the context they are use.
- function and subroutine names can also be up to 31 characters in length. Give them descriptive names that will help document their purpose or function.
- Fortran 95 requires interfaces in calling subroutines for every function referenced,
- blanks are significant in names,
- up to 39 continuation lines can be used (& at end of line),
- while Fortran 95 allows multiple statements per line (; is statement separator), their use is discouraged for RETRAN-3D,
- the **!** is the comment symbol and can be placed any where on a line (comments follow),
- the **include** option inserts source text from a file is supported, but will not find significant use in RETRAN-3D since public data storage is defined in data modules rather than common blocks (historically **include** statements were used to insert common block specifications).

Fortran 95 compilers recognize source files with ".f90" extensions as Fortran 95 source code, but some do not recognize ".f95" extensions. Consequently, ".f90" extensions are used for source code files even though they are Fortran 95 compliant.

1.2 Source Code Structure

Source code indentation is used to clearly identify the body of **do** loops and **if** blocks. For **do** loops, the **do** and **end do** statements will begin in the same column and intermediate statements will be indented by 3 spaces. **If** and **case** constructs are treated similarly where the control statements are at the same level and intermediate coding is indented by 3 columns. If there are many nested blocks, 2 columns may be used for indentation in order to retain more columns for instructions.

Functions and **subroutine**s should include prolog information as comments that define the purpose of the subprogram and any dummy arguments. It should also include the name of all subprograms that call it, unless it is called by a large number of subroutines (more than 6). A template for the subroutine prolog is shown below. Each dummy argument should be defined and the use of the argument indicated, whether input, output or input/output. This information will also be included in intent specifications as indicated below

```
      [Subroutine or function subprogram_name]  ([dummy1,] [dummy2])
!
!  A description of the subroutine or function should be included
!  here.
!
!  subprogram_name is called by subroutine sub_1, [sub_2,] …
!
!     [dummy1] = definition of dummy argument 1          (input)
!              = Also each value for control variables
!     [dummy2] = definition of dummy argument 2          (input/output)
```

Following the subprogram prolog, all data modules required by the subprogram must be referenced via **use** [**module_name**] statements. Modules are used to create storage pools that can be accessed from multiple subroutines. They replace common blocks used in most Fortran 77 programs. Modules can contain scalar variables, fixed dimensions arrays or deferred shape arrays or pointers. For RETRAN-3D, modules for most deferred shape arrays also contain related coding that allocates memory for the arrays at execution time. Section II.2.1 Module Use and Requirements identifies the standards used for RETRAN-3D modules.

The **kind_specs** module is a RETRAN-3D specific module that contains parameters that should be used to define the precision of various variable types, e.g., **rkd**, **ikd** and **lkd** for real, integer and logical data types, respectively. They should be used to specify the precision of all real, integer and logical data types used in RETRAN-3D. The **real** date type precision specification included in **kind_specs**, **rkd**, defines real variables and arrays to be double precision. The use of the **kind_specs** module will facilitate changing the precision of the code if required in the future by simply modifying the parameters shown below in the **kind_specs** module. The precision of character data should use the **ckd** parameter, using the length attribute to define the string length, e.g. **len=8** for an 8-character string. The **kind_specs** module as currently defined is shown below.

```
!=====================================================================
!  Define global kind specifications
!=====================================================================
!
      module kind_specs
!
      integer(kind=4), parameter :: &
            ikd=4, & !  integer kind specification (kind=4)
            lkd=4, & !  logical kind specification (kind=4)
            rkd=8, & !  real kind specification (kind=8)
            ckd=1    !  character kind specification (only kind=1 is
                     !  currently supported)
!
      end module kind_specs
```

When modules are included in a subprogram, they should be used in a manner similar to the following example.

```
!
!  Data specification modules
!
      use kind_specs
      use [module_name1]
      use [module_name2]
      …
      …
      …
      use [module_namen]
```

Following the data module **use** specifications, an **implicit none** statement should be included in each subprogram. It should be preceded by a comment line to help make it more visible in the source code. With the **implicit none** specification, all local variables must be explicitly typed. This helps to insure that undefined variables are not inadvertently used because of a typographical error. It also makes the programmer explicitly define the variable type while the coding is being developed, reducing the chance that a variable with the wrong type will be used.

```
!
      implicit none
```

If a subprogram uses dummy arguments, type specifications should be included for each argument. These specifications should include the associated precision for each type using the parameters defined in the **kind_specs** module. The specification statements should be preceded by the comments shown below (or similar comments). A type specification and associated precision is also required for any function name.

The intended use of each dummy argument should also be specified using an **intent** statement where **(in)** indicates the argument is input, **(out)** indicates the argument is output or calculated within the subprogram and **(inout)** indicates that the argument is both input and output. An example code segment follows.

```
!
!  Dummy Arguments
!
      integer(kind=ikd)   :: [dummmy1]
      real(kind=rkd)      :: [dummy2]
      intent (in)     [dummy1]
      intent (inout)  [dummy2]
```

Following the dummy argument specifications, specifications for local variables should be given if local variables are used in the subprogram. They should be preceded by the comments (or similar comments) as shown in the following code segment. Any real, integer and logical variables should use the appropriate type size parameter to define the precision, e.g., **rkd**, **ikd**, **lkd** and **ckd,** from the **kind_specs** module.

```
!
!  Local variables
!
      integer(kind=ikd)   :: var1
      real(kind=rkd)      :: var2
      logical(kind=ikd)   :: var3
```

Comments should be preceded and followed by blank comment lines to help the comment stand out from the remainder to the source code. This aids in making the source code more readable. The actual text of comments should also be indented similar to the instructions it applies to. Typically comments should be indented 3 columns less than the associated instructions, which should generally follow the comment, i.e. comments should preceded the instructions being described, including **do, else**, **else if**, **case**, etc.

Fortran 95 type specifications include a number of attributes that can be used as needed. The **kind** attribute for type specifications provides a clean way to control numeric precision. The **kind_specs** module contains parameters that should be used to define the precision of real, integer, and logical variables. The **dimension** attribute of the type statements should be used for arrays rather than Fortran 77 style **dimension** statements. Array bounds or rank should be specified with the dimension attribute. The length of character data types should be specified as needed. Generally, local variables should not use the save attribute. If a variable needs to be saved so its value is retained after a subprogram is exited, it should be placed in a module. Examples are illustrated in the following code segment.

```
!
!  Local variables
!
      real(kind=rkd), dimension(4) :: a,  d
      character(kind=ckd,len=30)   :: title
```

Program flow should generally be designed to flow from the top of a subprogram to the bottom. The use of **go to** statements should be minimized; however, there are situations where **go to** statements may be the most appropriate control construct. The use of **label** statements should be minimized. When **label**s are used, it is preferable to use them with continue statements rather than executable statements.

Computed and assigned **go to** statements should not be used since they are classified as detrimental to good programming practices; rather, **select case/case** constructs should be used. **If then**/**else** constructs are also useful alternatives to assigned or computed **go to** constructs.

Do loops should be terminated with **end do** statements where possible (rather than labels) and control transfers should use the **cycle** and **exit** options where possible. There will be occasions when a **go to** transfer will be required, but they should only be used if the other options won't provide the necessary control transfers. **Do while**, and associated **end do** should be used to control iterative loops rather than backward **go to** statements when practical.

Additionally, the following general guidelines should also be followed when writing code for RETRAN-3D.

- **Module**s offer a cleaner solution to most situations where **entry** points might be used in Fortran 77, particularly for large subprograms. For this case a private data module can be used to save the necessary data for related calls (entries). The use of entry points should be limited to small compact subprograms.

- Where possible, whole array processing should be used since it is now possible to treat whole arrays as objects in Fortran 95. It is preferable to write

  ```
  a       = 0.0d0
  ```

  to initialize array **a** rather than initializing each element of an array using a do loop. Similarly, whole array operations can be performed such as

  ```
  a       = b*sin(a)
  ```

  where **a** and **b** are arrays. This is also preferable to using a **do** loop to operate on the individual array elements.

  Partial array processing can also be used to zero part of an array, e.g. if array a is rank (1:100), the first n elements (n<100) can be zeroed using

  ```
  a(1:n) = 0.0d0
  ```

  to zero them. Similarly, partial array processing can be used to move elements from one array to another. If array **b** has the same rank as array **a**,

  ```
  b(51:100) = a(1:50)
  ```

  moves the first 50 elements of array **a** to the final 50 elements of array **b**.

- The use of **equivalence** statements can make programs difficult to understand and maintain. They can also hinder data flow analysis and, consequently, automatic compiler optimizations. For these reasons, their use should be limited in RETRAN-3D.

- Generic intrinsic functions should be used rather than type specific forms. Intrinsic functions and new features available in Fortran 95 can be used when writing RETRAN-3D source code.

- **Return** statements are not required at the end of a subprogram and should not be included. A return is automatically issued in Fortran 95.

- All **format** statements should be placed at the end of the subprogram, making it easier to locate them. They should be ordered by increasing **label** number.

- Include all separators between fields in format statements. Some compilers will accept missing separators but others will consider missing separators to be errors. The following example first illustrates good programming practice and the second poor practice. Use

```
100 format (i6,a8)
200 format (i8, 1p, 5e13.5)
```

rather than

```
100 format (i6a8)
200 format (i8, 1p5e13.5)
```

- Do not wrap strings in format statements across continuation cards. The following example illustrates good programming practice, followed by poor practice. Use

```
100 format (' do not wrap hollerith strings across' &
     ' continuation cards, terminate the string'    &
     ' then start new string')
```

rather than

```
100 format (' do not wrap hollerith strings across continuation cards &
   &as is done here')
```

- Character data types should be used to store Hollerith data when possible. Because all RETRAN-3D input data is stored in a real(kind=8) array when processed by the **inp** free form input processing package, character data must be extracted using in memory **write**s into a **character** type variable (see Section V.2.0 for additional discussion).
- Limit the magnitude of large literal floating point constants to exponents of 75 and small literal floating point numbers to exponents of -75. This will help prevent unnecessary overflow and underflow problems.
- When defining real data constants, use the d specification, e.g., 1.0d0 rather than 1.0.
- Use consistent data types when performing arithmetic and logical tests. The following example illustrates both good and poor practice. Use

```
i = ione   ! i and ione are integer(kind=4)
i = 2
if (i.eq.3) then
z = 5.0d0  ! z is a real(kind=8)
```

rather than

```
i = one     ! i is integer(kind=4) and one is real(kind=8)

i = 1.0d0
if (i.eq.zero) then   ! zero is a real(kind=8)
z = 5       ! z is a real(kind=8)
```

All **function** and **subroutine end** statements should include the subroutine name as shown in the following example.

```
subroutine subroutine_name ([arg1,] [arg2,] …… [argn])
…
…
end subroutine_name
```

Avoid using Fortran **read** operations from Fortran Unit 5, since they will conflict with the free form input processor INP which uses Unit 5.

1.3 Obsolete Code Constructs

As noted above, computed and assigned go to statements should not be used. Instead, the case and block if constructs should be use.

Additionally, the following statements shown on the left are obsolete. The indicated statements on the right should be used in their place.

• print n,...	write (*,n) ...
• accept n,...	read (*,n) ...
• type n,...	write (*,n) ...
• pause 'message'	write (*,*) 'message' `read (*,'()'`

2.0 DYNAMIC MEMORY ALLOCATION

RETRAN-3D uses the dynamic memory allocation features of Fortran 95 to assign main memory storage strictly based on the size of the problem and program options that are used. Thus, most storage for data arrays is allocated dynamically at execution time, not at compile time via dimensioned local or global arrays. There are a few situations where fixed dimension arrays are used. For these arrays, fixed dimensions will be specified as part of the type specifications.

In some instances, arrays will be used in a single subroutine and will not need to be made available to other subroutines. These will typically be input processing subroutines that are only called once. For these situations, either the fixed dimension type specifications or dynamic allocation will be included locally in the subroutine. If the subroutine is only called once, any dynamically allocated memory should be de-allocated before exiting the subroutine.

When specific storage blocks need to be made available globally to other subroutines, modules are used to define the storage specifications and attributes. The use of modules in RETRAN-3D is described in the following sections.

2.1 Module Use and Requirements

Modules are used to define scalar variables, arrays and derived data type structures that can be accessed by multiple subprograms. They include specifications for data type and precision. Precision specifications should use the **kind** specification with the appropriate parameter from the

kind_specs module (see Section II.1.2) to define the precision. All required attributes should be specified with the type specification. Modules should also include associated procedures, e.g. coding used to allocated and initialize the arrays or special functions or subroutines that operate on the information stored in the module.

When a module contains a function or subroutine that references an external subroutine or function, an **interface** block must be provided for the called subprogram in the calling subprogram. Such interface blocks could be incorporated into the calling subprogram using an include statement; however, in RETRAN-3D it is preferred to put the interface block into a **module** and then add a **use** statement to the calling subprogram.

By convention, modules that contain interface blocks are give filenames that begin with **s_**. This allows them to be easily and quickly located in directory and other related lists. The name following the **s_** prefix can be the name of the associated subroutine, thus avoiding duplicated object file names,

Two basic module types are used in RETRAN-3D. They are static and dynamic modules. All variables and arrays in these modules should have the **save** attribute specified as part of the type specification attributes. This will insure that values are maintained as control is transferred from one subroutine to another.

Modules should be programmed so they contain variables that define the dimensions for arrays that are contained within the model. For example, **module volumes** contains pointer arrays that have elements for each control volume. Their rank is (1:num_vol) were **num_vol** is the number of control volumes for the current problem. In RETRAN-3D, **module problem_dimensions** also contains dimensions and other control variables and variable **nvol(1)** is also the number of control volumes. When a subroutine requires knowledge of the number of volumes, either **nvol(1)** or **num_vol** are equivalent. However, it may be possible to exclude use of **module problem_dimensions** if other control or dimension variables are not needed. It is preferable to use the dimension variables in a module.

Comments should be included at the beginning of a module to identify its use. They should also indicate any subroutines or functions that are included in the module. Variables included in a given type declaration are placed in alphabetic order to aid in locating them when reviewing the code for use. A comment should follow each variable or array to define its use. Comments should also be included to describe any special uses or related data. Any information that might aid a programmer in using or modifying a module should be included as comments. Modules should be self documented.

By convention in RETRAN-3D, filenames for module source code are prefixed with an **m_**. This allows modules to be quickly identified in directory and other related lists. An exception is for modules that contain interface blocks. As noted above, their filenames will begin with s_ prefixes.

2.1.1 Static Modules

Static data modules are comprised solely of scalar variables and/or fixed dimension arrays. They should be used rather than common blocks. The following code segment provides an example of a static module used to store information for the off-rated power initialization feature. It contains both input data and procedure results.

```
!=============================================================================
!  Specifications and data definitions for off-rated conditions initialization
!  procedure
!=============================================================================
      module orcip
!
      use kind_specs
!
      integer(kind=ikd), parameter ::  &
            nlp =  4, &      !  maximum number of loops
            npj = 10, &      !  number of add primary junction specs
            nsj = 10, &      !  number of add secondary jun specs per sg
            nsg = 30         !  number of sg volumes per sg (mass)
!
      real(kind=rkd), save  ::  &
         bialev = 0.0d0, &   !  pzr level bias (ft)
         biapow = 0.0d0, &   !  power bias (mw)
         biaprs = 0.0d0, &   !  pzr pressure bias (psia)
         biatav = 0.0d0, &   !  averge loop flow bias (deg f)
         biawrs = 0.0d0, &   !  rcs flow bias (gpm)
         fulpow = 0.0d0, &   !  nominal full core power (mw)
           grcs = 0.0d0, &   !  rcs volumetric flow mgpm
           hrcs = 0.0d0, &   !  energy balance option (0-off,1-sg pres.)
          pcpow = 0.0d0, &   !  percent power (%) used for table lookup
          powip = 0.0d0, &   !  system power (mw) after bias is applied
         ppower = 0.0d0, &   !  pump power adjustment option
           pzrl = 0.0d0, &   !  pressurizer level
           pzrp = 0.0d0, &   !  pressurizer pressure
           trcs = 0.0d0, &   !  rcs target average temperature
           wrcs = 0.0d0      !  rcs mass flow rate lbm/sec
 !
      real(kind=rkd), dimension(nlp), save :: &
         biamas = 0.0d0, &   !  level / mass inventory bias (lbm)
         biapsg = 0.0d0, &   !  sg pressure bias (psia)
         biawsg = 0.0d0, &   !  sg feedwater/steam flow bias (lbm/sec)
         rlpfrc = 0.0d0, &   !  loop rcs flow fraction
          sgmas = 0.0d0, &   !  sg target level / mass inventory
            sgp = 0.0d0, &   !  sg pressure
          sgrcr = 0.0d0, &   !  sg recirculation ratio
          sgwfw = 0.0d0, &   !  sg fw/steam flow
          sgwrr = 0.0d0      !  sg recirculation flow
!
      real(kind=rkd), dimension(npj), save :: &
         rapfrc = 0.0d0      !  primary mass flow rate fraction of rcs flow
!
      real(kind=ikd),  dimension(nlp, nsj), parameter  ::  &
         rasfrc = 0.0d0      !  mass flow rate as a fraction of sg fw
!
      integer(kind=ikd), parameter  ::  &
         igtprs = 0, &       !  gen data table number for pzr pressure
         igtpzl = 0, &       !  gen data table number for pzr level
         igttav = 0, &      !  gen data table num for rcs avg loop temp (deg f)
         igtwrs = 0, &       !  gen data table num for rcs vol flow rate (gpm)
         ilpvol = 0, &       !  core lower plenum volume number
```

```
          ipzr = 0, &      !  pressurizer volume number
         itolim = 0, &     !  maximum outer iteration loops (mass/level)
         itvjun = 0, &     !  turbine stop valve junction number
          nloop = 0        !  number of loops defined
!
            …
            …
            …
!
      logical(kind=lkd), parameter  ::  &
          iebal = .false., &   !  flag - energy balance (sg pres) model used
         iordbg = .false., &   !  flag - turn debug writes on to units 65 & 66
         ipcvrg = .false., &   !  flag - steady state outer loop converged
         ipoutr = .false., &   !  flag - outer iteration used (sg level/mass)
          iproc = .false., &   !  flag - initialization procedure used
         ipwtav = .false.      !  flag - wrcs & tavg procedure model used
!
      end module orcip
```

As shown in the example above, comments are included at the start of the module to identify its use. Comments are also provided that describe each variable and array. When fixed dimension arrays are used, it is good practice to use parameters to define the dimensions. This facilitates making revisions to the dimensions if required at a later date.

Note that the scalar variables and arrays are initialized as part of the type specification statements. This is a good programming practice and should be used.

Table II.2-1 contains a list of the static data modules used in RETRAN-3D. It also includes a description of the information included in the module. Refer to the source code for a complete list of variable and array definitions for each module.

2.1.2 Dynamic Modules

Most of the storage assigned to memory in RETRAN-3D uses deferred shape pointer arrays that are allocated at execution time. The size of these arrays will be determined from the input model prior to allocating the space. Note that functions **ncards** and **nitems** from the **inp** free form input processing package (see Sections V.2.11 and V.2.12) are often used to determine required problem size before the actual input data are processed. This allows memory to be allocated before the input is processed.

Modules are used to define the array types, attributes, and definition (via comments). They will also contain coding used to allocate the required memory for each array used. These dynamic allocation subroutines are typically given names of the form **allok8_xxx**, where **xxx** is a unique character string that identifies the associated function. The following code segment is from **module volume** that defines the arrays used to store volume related information. It contains dynamic memory allocation subroutine **allok8_volumes**, which allocates the dynamic memory associated with control volumes. The code segments from **module volumes** illustrate a typical dynamic module from RETRAN-3D.

Table II.2-1

RETRAN-3D Static Data Modules

Module Name	Description
arrcom	Contains control arrays and communication vectors used to transfer information between the 3-D kinetics and thermal-hydraulics subroutines.
edit_headers	Contains textual information documenting the code version, licensed organization, problem title and date that the problem was run.
errdata	Contains communications blocks used to pass error descriptions and related problem results to the error log processing subroutine **errlog**.
inp_cards	Contains the **rdata** array where subroutine INP returns card data. It also contains function **idata** that is used to fetch integers out of the **rdata** array.
ncgasc	Contains arrays used to store constants for the noncondensable gas defined. Used by property routines **gastkv** and **gaseos**.
orcip	Contains problem specific target initial conditions and related information used by the off-rated condition initialization procedure option for steady-state initialization.
restart_block_list	Contains the derived type data declaration and subroutine **build_restart_block_list** used to build a list of dynamically allocated data blocks that will be included in restart files.
retran_configuration	Contains constants and data used to initialize retran-3D.
unit	Contains commonly used conversion factors. It also contains the conversion factors used to convert RETRAN-3D output to si units.

```
!=======================================================================
!  Specifications and space allocations for volume related arrays
!
!  This module also contains the following subroutines
!
!      allok8_volumes
!=======================================================================
!
      module volumes
!
      use kind_specs
!
      integer(kind=ikd), save :: &
          num_vol, &  !  number of volumes
         siz_rvol, &  !  size of array allocated for real vol pointer arrays
         siz_ivol, &  !  size of array allocated for integer vol pointer arrays
         siz_lvol     !   size of array allocated for logical vol pointer arrays
!
!
! All of the following real, integer and logical pointer arrays are Rank (1:num_vol)
! Space is allocated and pointers associations are made in subroutine
! allok8_volumes
!
      real(kind=rkd), target, allocatable, save :: r_vol(:)
      real(kind=rkd), pointer, save :: &
        afdai(:), &  !  normal to interfacial area density ratio
       areaif(:), &  !  interfacial area density ( ft**2 / ft**3 )
       armaso(:), &  !  previous time step noncondensible mass
       armass(:), &  !  noncondensible mass
         aved(:), &  !  average density
         avex(:), &  !  average mixture quality = gas mass / total mass
            …
            …
            …
       xvapol(:), &  !  previous time step xvapor
       xvapor(:), &  !  vapor mass / water mass
       zlqlev(:), &  !  equivalent collapsed liquid level
         zvol(:)     !  volume height
!
      integer(kind=ikd), target, allocatable, save :: i_vol(:)
      integer(kind=ikd), pointer, save :: &
         ibub(:), &  !  bubble data index
       icndns(:), &  !  mode 20 recondensation flag
       idxetr(:), &  !  index to subfile to 2-reg enth tran model
       idxlst(:), &  !  index for connection list (w shs model)
       iequat(:), &  !  equation set flag - model option
          ifr(:), &  !  flow regime index
            …
            …
            …
            …
!
      contains
!
!=======================================================================
     subroutine allok8_volumes
!
!  Subroutine to allocate space and set pointers for vol related arrays
!     Note: num_vol must be defined before allok8_vol is called
!
!     allok8_volumes is called from subroutine invol
!
      use kind_specs
      use restart_block_list
```

```
!
!  Interface modules
!
      use s_r_pntr
      use s_i_pntr
      use s_l_pntr
!
      implicit none
!
      character(kind=ckd,len=8) block
      integer(kind=ikd) l, i2, status
      logical(kind=lkd) allocatd
!
      real(kind=rkd),    pointer :: rdum(:)  !  dummy real pointer
      integer(kind=ikd), pointer :: idum(:)  !  dummy integer pointer
      logical(kind=lkd), pointer :: ldum(:)  !  dummy logical pointer
!
      if (num_vol .le. 0) then
         write (6,110) num_vol
         call fail
         return
      endif
!
      if (allocated (r_vol)) return
!
!-------------------------------------------------------------------------------
!  Allocate space and assign pointers for real arrays.  The first pass determines
!  size of block to allocate and the second pass assigns the pointer arrays
!-------------------------------------------------------------------------------
!
      do l = 1, 2
         allocatd = l.eq.2
         i2       = 0
!
         call r_pntr (allocatd, num_vol, i2,  afdai, r_vol)
         call r_pntr (allocatd, num_vol, i2, areaif, r_vol)
         call r_pntr (allocatd, num_vol, i2, armaso, r_vol)
         call r_pntr (allocatd, num_vol, i2, armass, r_vol)
         call r_pntr (allocatd, num_vol, i2,   aved, r_vol)
         call r_pntr (allocatd, num_vol, i2,   avex, r_vol)
            …
            …
            …
         call r_pntr (allocatd, num_vol, i2, xvapol, r_vol)
         call r_pntr (allocatd, num_vol, i2, xvapor, r_vol)
         call r_pntr (allocatd, num_vol, i2, zlqlev, r_vol)
         call r_pntr (allocatd, num_vol, i2,   zvol, r_vol)
!
!      allocate space
!
         if (.not.allocatd) then
            siz_rvol   = i2
!
            allocate (r_vol(siz_rvol), stat=status)
!
!        check status of allocate - write error if allocate failed
!
            if (status .ne. 0) then
               block      = 'real'
               write (6,100) block, siz_rvol, status
               call fail
            endif
!
!        Add block to list of blocks included in restart file
```

```
!
            rdum   => r_vol
            block  = 'r_vol'
!
            if (num_vol /= 1 ) call build_restart_block_list (block, &
                                   rdum, idum, ldum, siz_rvol, 1, 2)
!
!     initialize real arrays to 0.0d0
!
         else
            r_vol      = 0.0d0
         endif
      enddo
!
      end subroutine allok8_volumes
!
      end module volumes
```

The initial coding in the example volume module defines the arrays that will be allocated dynamically. Note that the example shows arrays **r_vol** and **i_vol,** which will be used for the real and integer volume data. Similar arrays are used for any logical arrays that might be necessary. The following discussion is for the real array, but it would also apply to the integer and logical arrays.

All real arrays that are needed to store volume related information, e.g. air mass – **armass**, average density – **aved**, etc., are defined as deferred shape **pointer** arrays. Each **pointer** specification also includes a comment string that defines the use of the array. The pointers are defined in alphabetical order to aid in finding pointer definitions when viewing the source code.

The memory required for the **r_vol** array will depend on the number of volumes in the RETRAN-3D model and the number of pointer arrays used to describe a volume. The number of volumes is known when the memory is allocated by calling subroutine **allok8_volumes** with is part of the module. The number of **pointer** arrays is determined by the first calls made to subroutine **r_pntr** (when l=1). The number of variables could have been hard wired but it would have to be changed anytime a new pointer array is added or deleted. With the method shown, a call can be added with the new array name in a call to subroutine **r_pntr**. Likewise, an array can be deleted by deleting the call to subroutine **r_pntr**. Note that corresponding changes would have to be made to the type specifications at the top of the volumes module.

The following code illustrates the **r_pntr** subroutine and the operation that it performs.

```
      subroutine r_pntr (allocatd, nsiz, len, rpntr, rtargt)
!
!  Subroutine to determine space requirements and assign array pointers for
!  real arrays.
!
!     len      = cummulative length of target array              input/output
!     allocatd = locical used to assign pointer values           input
!              = .false. - calculate space required
!              = .true.  - make pointer association
!     nsiz     = size of pointer array                           input
!     len      = cummulative length of target array              input/output
!     rpntr    = pointer to associate with target                input/output
!     rtargt   = target  array                                   input
```

```
!
      use kind_specs
!
      implicit none
!
      integer(kind=ikd) len, nsiz, low
      logical(kind=lkd) allocatd
!
      real(kind=rkd),   pointer :: rpntr(:)
      real(kind=rkd),   target  :: rtargt(:)
!
      low   = len + 1
      len   = len + nsiz
!
      if (allocatd) then
         rpntr => rtargt(low:len)
      endif
!
      end subroutine r_pntr
```

During the second pass through the loop (l=2 and **allocatd**=.true.), subroutine **r_pntr** associates the individual pointer arrays with the **r_vol** block. The first pointer array will be associated with the beginning of **r_vol** and each subsequent array will be offset by **num_vols** from its predecessor in the **r_pntr** call sequence. Table II.2-2 illustrates the pointer associations. Note that consecutive addresses in **r_vol** (and each pointer array) increase with row. Also each new array actually resides in memory immediately after the preceding array, i.e. the individual one dimensional arrays are mapped into **r_vol,** which is also a one dimensional array.

Table II.2-2

Pointer Array Associations in the Volumes Real Data Block r_vol[1]

afdai(1)	areaif(1)	armaso(1)	…	zlqlev(1)	zvol(1)
afdai(2)	areaif(2)	armaso(2)	…	zlqlev(2)	zvol(2)
afdai(3)	areaif(3)	armaso(3)	…	zlqlev(3)	zvol(3)
afdai(4)	areaif(4)	armaso(4)	…	zlqlev(4)	zvol(4)
afdai(5)	areaif(5)	armaso(5)	…	zlqlev(5)	zvol(5)
afdai(6)	areaif(6)	armaso(6)	…	zlqlev(6)	zvol(6)
…	…	…	…	…	…
…	…	…	…	…	…
afdai(n)	areaif(n)	armaso(n)	…	zlqlev(n)	zvol(n)

[1] n = num_vol

Each of the arrays will have a rank of **1:num_vol**. They are accessed as if they are fixed dimension arrays with the same rank.

After the pointer arrays are associated and initialized, there is a call to subroutine **build_restart_block_list**. This call adds the block name, pointer type, pointer, and block length to a list that will be used to write a restart file (see Section II.3.0). For **module volumes**, array

r_vol will be written to the restart file, which includes all individual pointer arrays associated with the **r_vol** array. This simplifies the coding required to write (and read) the restart file, as will be seen later. In summary, the content of any array included in a block via a call to subroutine **r_pntr** will not only be assigned storage, it will also be included automatically in the restart file.

Blocks are also assigned for **integer** and **logical** arrays in the volumes module. Similar memory allocation is used for other component related modules, e.g., junctions, valves, pumps, etc.

In the discussion above, the pointers and dynamically allocated arrays are one-dimensional. Several components in RETRAN-3D use two-dimensional pointer arrays. They are typically rank (1:n,1:2) where the second dimension or subscript is used to identify that the variable is either related to the upstream (1) or downstream (2) side of a junction, or it is related to the left surface (1) or right surface (2) of a heat conductor. The necessary two-dimensional arrays are defined and allocated in the appropriate module, i.e., **junctions** and **conductors**.

Allocatable array **r_jun** is used to define the data array for real one-dimensional junction pointer arrays, while **r2_jun** is used for real two-dimensional arrays. Where subroutine **r_pntr** is used to define the size of **r_jun** and to make pointer assignments for each of the real one-dimensional pointer arrays associated with **r_jun**, subroutine **r2_pntr** performs the same task for real two-dimensional pointer arrays.

Subroutine **build_restart_block_list_2d** (actually and entry point in subroutine build_restart_block_list) is similar to **build_restart_block_list** in that it makes the necessary entries into the restart block list.

Similar two-dimensional blocks are assigned for **integer** and **logical** arrays in the appropriate module where the corresponding one-dimensional arrays are defined.

Table II-2.3 contains a list of the dynamic data modules used in RETRAN-3D. It also includes a description of the information included in the module and whether a module includes two-dimensional arrays. Refer to the source code for a complete list of variable and array definitions for each module.

2.2 Segmented Arrays

Segmented arrays (SA) are primarily used for three-dimensional kinetics calculations. They are dynamically allocated arrays that are logically divided into sets of data. A SA can be visualized as a memory block with m columns and n rows. The number of columns is referred to as the set size and a row of data is referred to as a set, which represent a data structure that is set size words long. All data within a SA resides in memory and is readily available, but it is typically accessed one set (row) at a time. SAs are used in the Fortran 95 version of RETRAN-3D to replace the use of FTB PROCESS files [II.2-1], which are obsolete.

Table II.2-3

RETRAN-3D Dynamic Data Modules

Module Name	Description
2rentr	Contains information needed for the two-region enthalpy transport option.
accumulators	Contains additional information required for accumulator volumes.
adjoint	Contains information needed for three-dimensional kinetics adjoint calculation.
auxiliary_dnb	Contains information required for the auxiliary dnb model.
auxialiary_file	Contains the file format information used when the auxiliary data file (tape60) is written.
bc_file	Contains information for boundary condition volumes that use RETRAN-3D restart file to provide the BC information.
bicgs	Contains information required for bi-conjugate gradient methods.
bubble_rise	Contains additional information used by the bubble rise model.
channel_geometry	Contains array required for channel geometry.
channel_model	Contains information needed for the channel model.
conductors	Contains the 1D and 2D arrays used by the heat conduction model.
control	Contains the arrays used by the control system model.
core	Contains the arrays used by powered heat conductors.
coupl	Contains information required for coupling coefficient arrays.
decay_heat	Contains arrays and flags used for the decay heat model.
dynamic_gap	Contains the arrays used by the optional dynamic gap conductivity model.
fibwr	Contains the arrays and scalar variables used by the optional fibwr leakage model.
fills	Contains the arrays and scalar variables used by the fill model.

Table II.2-3 (Cont'd)

Module Name	Description
flow_split	Contains the arrays and scalar variables used by the steady-state flow split model.
gen_data-tables	Contains the arrays and scalar variables used to store the general data tables.
general_transport	Contains the arrays and scalar variables used by the optional generalized transport model.
grid_loss	Contains the arrays and scalar variables used by the optional grid loss model to store the input coefficients and control varialbles.
heat_exchangers	Contains the non-conducting heat exchanger model control flags and constants.
itrctrl	Contains information required for three-dimensional kinetics iteration control parameters.
junctions	Contains the 1D and 2D arrays use to store junction geometric descriptions and results.
kin	Contains information required for three-dimensional kinetics arrays.
lscoef	Contains arrays required for linear system coefficient.
material_properties	Contains tables and index arrays used to store the material properties used by the heat conduction model.
meshs	Contains information required for three-dimensional kinetics and neutronics mesh.
minor_edit_search	Contains the minor edit search list array and associated subroutines to build the search list, locate requested minor edits in the list, and retrieve the minor edit variable values.
minor_edit_summary	Contains the arrays used to store the minor edit summary values and the subroutine that is used to write the summary.
minor_edits	Contains the arrays used to store the requested minor edits and storage use to store minor edit values between edit dumps to the output file.
params	Contains information required for three-dimensional kinetics parameters.

Table II.2-3 (Cont'd)

Module Name	Description
point_kinetics	Contains the arrays and scalar values used to store the point kinetics results.
problem_dimensions	Contains the arrays used to store the problem options and dimensions.
profile_fit	Contains the arrays used to store the input and default coefficients for the profile fit option.
pumps	Contains the arrays used to store the centrifugal pump descriptions, performance curves and results.
restart	Contains variables used to read and write restart files. It also contains the subroutines that perform the read and write procedures.
rmap	Contains information required for three-dimensional kinetics mapping arrays.
scram	Contains the arrays and scalar variables used to store scram and power versus time curves.
separator	Contains the arrays and scalar variables use to store the separator performance curves and results.
solvec	Contains information required for solution vector arrays.
sparse_matrix	Contains the control vectors that describe the structure of the sparse matrix used to obtain the solution to the thermal-hydraulic balance equations.
spec_htc	Contains the arrays associated with the heat transfer option to specify a heat transfer coefficient.
ss_conductors	Contains arrays used to save heat conductor information for steady-state initialization.
ss_junctions	Contains arrays used to save junction information for steady-state initialization.
ssvar	Contains arrays required for three-dimensional kinetics steady-state variables.

Table II.2-3 (Cont'd)

Module Name	Description
ss_volumes	Contains arrays used to save volume information for steady-state initialization.
system	Contains the arrays used to save the system related solution results.
th_pow	Contains the arrays used to save the loop power levels for steady-state initialization.
three_dim_kinetics	Contains information required for feedback, power, setfix, etc.
time_dependent_volumes	Contains the arrays and scalar variables used to define time-dependent volume boundary conditions specified by tabular input.
time_steps	Contains arrays and scalar variables used by the time-step size selection algorithms.
trips	Contains the arrays and scalar variables used by the trip logic.
turbines	Contains the arrays and scalar variables used by the optional turbine model.
two_reg_neq	Contains the arrays and scalar variables used by the optional two-region nonequilibrium model.
valves	Contains the arrays and scalar variables used by the optional valve model to describe the characteristics and status of valves.
vbc	Contains the arrays and control information used to write a VIPRE-02 boundary condition file.
wallf	Contains the arrays and control information used by the optional wall friction model.
work_arrays	Contains the scratch or work arrays used by various models.
xseca	Contains information required for three-dimensional kinetics cross-section arrays.

Each SA has a unique identification number (ID). It is either 1 or 0. An ID of 1 suggests that space is allocated for the SA. After knowing the size of each set and total number of sets, space is allocated by calling the **allocate** statement. The SA ID is set to 1 and set size is saved. Structurally each SA contains an ID, set size, and the allocated SA. Since each set is stored contiguously in the allocated space, accessing individual sets of data requires moving through the SA by incrementing the index by the set size. The index for the first set is 1, the index for the second set is (1 + set size) and so on. The index for a given set is used to access all data within that set.

The following code segment illustrates the allocation and use of a SA where there are 3 variables for each column in the neutronics mesh (not to be confused with the set size). The set size for the example is 3*ncol.

```
!
!  If new xsection model is used (newxec  >  0), then idthvn is
!  described and initialized based on reference values
!
      if ( newxec > 0 ) then
!
!     Set up idthvn with the corresponding reference values
!
         allocate(thvn(3*ncol*nrp),stat=status)
         if (status .ne. 0) then
            block = 'thvn'
            routine = 'thflat'
            call error_allok8 (block,3*ncol*nrp,routine,status,0)
            call fail
         end if
         thvn    = 0.0
         idthvn  = 1
         ssthvn  = 3*ncol
         numthvn = nrp
!
!     Get the composition type for each node
!     and initialize with tmref=0, tfref & dmref from propr
!
         idxcmp = 1
         idxthn = 1
         do k = 1, npln
            jcmap = idxcmp
            jij   = jcmap
            ij    = jij
            do j = 1, nrow
               jt = idxthn
               do i = 1, ncol
                  nftype = icomp(ij)/64
                  do ipos = 1, nfuel
                     if ( ipropr(1,ipos) == nftype ) go to 5
                  end do
                  write (6,9010) nftype
                  call fabend
    5             continue
                  thvn(jt)   = 0.
                  thvn(jt+1) = propr(4,ipos)
                  thvn(jt+2) = propr(3,ipos)
                  jt = jt + 3
                  ij = ij + 1
```

```
        end do
        idxthn = idxthn + ssthvn
      end do
      idxcmp = idxcmp + sscomp
    end do
```

3.0 RESTART FILE

RETRAN-3D has an optional feature that creates a restart file, which can be used to continue or restart a previous problem that saved a restart file. Restart files can also be used to provide boundary condition information to a subsequent problem. For example, a restart file saved for a system analysis can be use to provide the plenum boundary conditions for a subsequent hot channel problem that only models the plenums and a single hot channel between them.

The scheme for generating restart files is an integral part of the memory allocation scheme. As storage blocks are allocated, attributes of the block are stored in a restart block list. This list contains the block type (real, integer or logical), a pointer to the block, the length of the block and the name of the block. The content of all 1-D, 2-D, and 3-D blocks comprise a restart record. Restart records are written or read using the information in the restart block list. This significantly minimizes the maintenance effort required to keep restart functioning correctly as changes are made to the code.

As discussed in Section II.2.1.2, separate dynamically allocated data blocks are defined for each data type. One block may contain data that defines model options, features, geometry, etc. that will remain constant for the duration of the problem. This static data is not included in the dynamic restart records. It is initialized during input processing for the restart job. Information in these static blocks can be revised at restart time using the generalized restart option. Another block type contains the time varying or dynamic blocks, which can be included in the restart file. A null block contains information that is not required for either restart or other program features such as **reedit** or time-dependent boundary conditions. For example, null blocks might include scratch data arrays used by matrix solvers or arrays used to provide communication between subroutines. Several modules contain mixed data blocks, a combination of static blocks and dynamic blocks. For generalized restart more flexibility is given to the user to change several junction, volume related parameters. For this purpose some modules may be divided into static data blocks and dynamic data blocks. For example, relative roughness can be changed during generalized restart for volumes. Therefore for volume module relative roughness is saved under static data blocks and remaining variables are saved under dynamic data blocks. Refer to the source code for more details on types of blocks and modules that comprises the different types of block. Modules definition has **build_restart_block_list** routine and block type defines which type of block it is associated with.

All blocks that need to be included in a restart record are added to the restart block list that is generated as data blocks are allocated. Each restart record is written by looping through the restart block list, which uses a derived data type to store block pointers, block type, block length, and block name. Restart files are binary files written using unformatted (binary) reads and writes.

Restart problems begin by proceeding through the same input processing as RETRAN-3D problems. This allows the same coding to be used for allocating memory for both RETRAN-3D and RESTRT problems. This also allows generalized restart to function similarly to previous code versions. Information such as model options, tables, and trip of control system data reside in the static memory blocks, which allows them to be revised using replacement cards as previously done via generalized restart. Following completion of the input processing, restart data is read into the storage blocks and the transient solution is continued.

The following sections discuss the content of RETRAN-3D restart data files as well as how they are used for restart problems, for problems where printed edits are obtained for previous solutions, and to provide time dependent boundary conditions.

3.1 File Content

The content of a restart file is determined by the restart file block list, which is created as dynamic memory blocks are created during input processing. As memory blocks are allocated for various features or options using the corresponding **allok8_xxxx** subroutine, a call to subroutine **build_restart_block_list** will add the required information to the derived data type restart file block list. Refer to the example shown in Section II.2.1.2 for the **allok8_volumes** subroutine where the **r_vol** block was allocated and included in the restart file block list by the call to subroutine **built_restart_block_list**. Table II-3.1 shows the information saved for each data block included in the derived data type list (**rest_list**).

Blocks that are defined as static blocks are only written to the restart file header record and are not included in the transient restart records. Only blocks that are defined to be dynamic are included in transient records. The header record is written by looping through the restart block list and only writing static records. Each block write is preceded by a write that contains the block name and length. The block is then written. A similar method is used for transient restart records where only dynamic blocks are written. The order of both static and dynamic blocks written to a restart file is determined by their order in the restart block list. An edit of the restart block list is included in the output file.

Section IV.2.1 describes the restart file structure and content in more detail.

The content of static data modules (see Section II.2.1.1) are not included in restart data files. They are primarily used to provide communication between subroutines and their content is either defined during input processing or when a subroutine uses it to communicate with another. As such, their content does not need to be included in the restart file. If a static data block (fixed dimensions) needs to be included in a restart file, it should be put in a dynamic module. The **problem_dimension** and **system module**s are examples. The variables in both storage blocks are scalar. They were typed as pointer arrays with rank 1 so they can be included in the restart file. For this reason, they are always referenced with a subscript of 1, e.g., the number of volumes in the **problem_dimension** module is **nvol(1)** and the normalized power in the **system** module is **pnorm(1)**.

Table II.3-1

Restart Block List Derived Data Type

Field Name	Description
block_name	A unique 12-character field defining the block name
r_restart_pntr	The pointer for real blocks
i_restart_pntr	The pointer for integer blocks
l_restart_pntr	The pointer for logical blocks
r2_restart_pntr	The pointer for 2D real blocks
i2_restart_pntr	The pointer for 2D integer blocks
l2_restart_pntr	The pointer for 2D logical blocks
r3_restart_pntr	The pointer for 3D real blocks
i3_restart_pntr	The pointer for 3D integer blocks
l3_restart_pntr	The pointer for 3D logical blocks
block_size	The length of the data block
pntr_type	The pointer type – 1 = real 2 = integer 3 = logical 4 = 2D real 5 = 2D integer 6 = 2D logical 7 = 3D real 8 = 3D integer 9 = 3D logical
block_type	The block type – 0 = block is not written to restart file 1 = block is written to the restart file header record (contains only static data) 2 = block is written to the restart data records (contains dynamic or transient data)

When a restart file is created during a problem run, a summary edit is written to the output file (written by calling **edit_restart_statistics** from **retran**). It begins with the header record information that documents the code version used to create the restart file, the problem title and date that it was created. The edit also lists all of the block names, pointer types and block lengths for the static and dynamic blocks included in the restart block list (as well as the null blocks that are not included in the restart file.

3.2 File Use

RETRAN-3D restart files are used by program option **restrt** to restart or continue a previous problem solution. They can also be used to obtain printed output edits of solution information archived on restart files using the **reedit** program options. Restart files can be used to provide boundary conditions for subsequent analyses. Pressure boundary conditions, power and power fraction (for problem that originally ran with 1-D kinetics) can be retrieved from a restart file for

a previous problem solution. The following sections describe the use of restart tapes for **restrt** problems and **retran** or **restrt** problems that use them to define time dependent boundary conditions.

3.2.1 Restart Problems

For a **restrt** problem, an RETRAN-3D restart file created by a prior **retran** problem is attached with a file name of TAPE13 to Fortran unit 13. This file is read by subroutine **inrstr**. It first calls subroutine **open_restart_file** to open and check the file to insure that it is a RETRAN-3D restart file. Subroutine **get_static_blocks** is then called to obtain the old problem dimension information from the **i_dim** static block and the original problem input from the **r_cards** block. Subroutines **open_restart_file** and **get_static_blocks** are included in the **restart** module. After obtaining the required information from the static data blocks in the restart file, subroutine **inrstr** returns to subroutine **restrt** (an entry point in subroutine **retran**). Subroutine **restrt** then repeatedly makes calls to **read_restart_record** and **write_restart_record** to first move the information in the restart record into memory, and to then write it to a new restart file (if one is to be written). This process is repeated until the record is reached where the restart problem is to continue the transient solution. This read-write process makes a new copy of the restart file through the record where the restart solution resumes.

If a new restart file is not written (selected by user input), subroutine **read_restart_record** issues dummy reads to skip the data blocks until the restart data record is reached. The calls to subroutine **write_restart_record** will be skipped also. As with the other subroutines that read or write restart files, subroutine **read_restart_record** also resides in module **restart**.

Once the restart problem solution begins, all subsequent data records are written by subroutine **edit**, which calls subroutine **write_restart_record**.

3.2.2 Re-edit Problems

For a **reedit** problem, a RETRAN-3D restart file created by a prior **retran** problem is attached with a file name of TAPE13 to Fortran unit 13. This file is read by subroutine **inedte**. It first calls subroutine **open_restart_file** to open and check the file to insure that it is a RETRAN-3D restart file. It then calls subroutine **allok8_data_blocks** to allocate the static and dynamic data blocks included in the restart file. Subroutine **allok8_data_blocks** also defines the associations for the pointer arrays that belong to each data block. Subroutines **open_restart_file** and **allok8_data_blocks** are included in the **restart** module. After allocating space for all blocks in the restart file, subroutine **inedte** returns to subroutine **reedit**. Subroutine **editre** is then called. It makes repeated calls to **read_restart_record** to move the content of a given restart record into memory. Restart records are read until the desired edit time point is reached at which time subroutine **edit** is called to generate the requested major and minor edits, and printer plots. The **open_restart_file, allok8_data_blocks** and **read_restart_record** subroutines are included in **module restart**.

3.2.3 Time-Dependent Boundary Conditions

For a **retran or restrt** problem that obtains boundary conditions from a RETRAN-3D restart file created by a prior **retran** problem, the restart file is attached with a file name of TAPE12 to Fortran unit 12. This file is read by subroutine **tapebc**, which calls subroutine **bc_file_initialization** on the first call to **tapebc.** , Subroutine **bc_file_initialization** then calls subroutine **open_restart_file** to open and check the file to insure that it is a restart file. Subroutine **bc_file_initialization** then allocates integer, logical and real buffers that are used to temporarily hold the content of various static and dynamic data blocks as they are read. The sizes of these blocks are determined in subroutine **bc_file_initialization** using the **mxi_len**, **mxl_len**, **mxr_len, mxi2_len, mxl2_len, and mxr2_len** values that subroutine **open_restart_file** read from the restart file header record. They correspond to the maximum length for the specific block type that appears in the **restart_block_list.**

Subroutine **bc_file_initialization** reads the **i_dim** static block from the restart file header record. The **nvol_bc**, **nbub_bc**, **ncor_bc** and **nodel_bc** variables are defined using the **nvol(1)**, **nbub(1)**, **ncor(1)** and **nodel(1)** values read from the **i_dim** block into the **i_buff** integer read buffer array. The appropriate values are retrieved by using the **loc** function to compute the index into the **i_buff** array. The following example illustrates the logic used to retrieve the number of volumes **nvol(1)** from the integer buffer containing the **i_dim** block. Note that **ikd** is an integer parameter that is used to define the precision of integer variables. It resides in **module kind_spec**.

```
iref     = loc(i_dim(1))
ioff     = (loc(nvol(1))  - iref)/ikd
nvol_bc  = i_buff(ioff)
```

nbub_bc, **ncor_bc**, and **nodel_bc** are defined similarly.

After the necessary dimensions are retrieved from the **i_dim** block, **pointer** arrays are **associate**d with the appropriate buffer arrays to facilitate retrieval of the boundary condition information. This is also accomplished using the **loc** function to identify the order of a particular pointer array in its associated dynamic block. The following example illustrates the logic used to retrieve the elapsed problem time from the real buffer containing the **r_sys** block. Note that **rkd** is a real parameter that is used to define the precision of real variables. It resides in **module kind_spec**.

```
i         =  (loc(timex(1)) - loc(r_sys))/rkd
timex_bc  => r_buff(i:i)
```

The normalize power is retrieved from the system block in a similar manner. Other pointer arrays that have non-unity dimensions are also obtained using the **loc** function. An example for the volume pressure follows.

```
pntr_m1   =  (((loc(p(1)) - loc(r_vol))/rkd) - 1)/nvol(1)
i         =  1 + pntr_m1*nvol_bc
p_bc      => r_buff(i:i+nvol_bc-1)
```

This coding maps pointer array **p_bc(1:nvol_bc)** into the **r_vol** buffer array so the required boundary condition values can be easily retrieved. A similar mapping is done for pointer array **hw_bc(1:nvol_bc)**. The old volume number as obtained from the **i_vol** buffer array using a similar mapping approach as are the other bubble rise and core pointer arrays identified in Table II.3-2.

Table IV.3-2

Data Blocks Used for Boundary Condition Retrieval from a RETRAN-3D Restart File

Block Name	Type	Block Use
i_dim	Static	The problem dimensions from the original problem. Values used include **nvol(1)**, **nbub(1)**, **ncor(1)**, **nodel(1).**
r_vol	Dynamic	This data block contains the pressure and enthalpy arrays, **p(1:nvol_bc)** and **hw(1:nvol_bc)** that contain the boundary condition information that may be needed, where **nvol_bc=nvol(1)** from the **i_dim** block.
r_sys	Dynamic	This data block contains the elapsed problem time and normalized power pointer arrays **timex(1)** and **pnorm(1). timex(1)** is always needed when boundary condition information is obtained from a restart file. The value of **pnorm(1)** will be used if **nodel(1)** is -1 or -2.
r_bub	Dynamic	This block contains the mixture level and old volume number, **zm(1:nbub_bc)** and **ibubol(1:nbub_bc)** that contain the boundary condition information that may be needed, where **nbub_bc=nbub(1)** from the **i_dim** block.
r_core	Dynamic	This block contains the core section power fractions, **qfrac(1:ncor_bc)** that contain the boundary condition information that may be needed, where **ncor_bc=ncor(1)** from the **i_dim** block. The boundary condition power fractions are used for **nodel(1)** is -2 or -3. Note that the number of core sections for the new problem must be the same as for the problem that created the restart file being used.

After performing the required mapping to the buffers used to read the static and dynamic restart blocks, subroutine **bc_file_initialization** returns to subroutine **tapebc**, which then calls subroutine **get_bc_data** to retrieve boundary condition data from the restart file. As each dynamic block identified in Table II.3-2 is read, the required information is retrieved from the read buffer and transferred to other pointer arrays allocated to save specific boundary conditions for a given time value. Once the required dynamic data blocks have been read, all others for the current record are skipped and **get_bc_data** returns to **tapebc**.

Two variables or arrays are used for each boundary condition value, one for the old value time and another for the new time. If the current problem time is greater than or equal to the new time for the boundary values, the new boundary conditions are moved to the corresponding old time values. Subroutine **tapebc** then calls **get_bc_data** to retrieve data from the next restart record. If necessary, the procedure is repeated until the old and new boundary condition times bound the current problem time. Subroutine **tapebc** then computes the boundary condition values for the current problem time using linear interpolation between the two bounding values.

4.0 MINOR EDIT VARIABLES

RETRAN-3D code users define minor edit variables as pairs of an alpha numeric flag and an integer region number. These minor edit request pairs uniquely identify a specific variable for a given component that is included in a special output edit. These edits can include printed history of a set of minor edits(see Volume 3 Section IV.4.0), an auxiliary file containing minor edits (see Volume 3 Section IV.4.1) that can be used by other codes, or printer plots (see Volume 3 Section IV.2.0 - NPRPLT). Control system inputs are also defined using minor edit variables (see Volume 3 Section IV.29.5)

The following sections describe how candidate minor edits are set up, how they are located for use during a RETRAN-3D problem and how the value of a minor edit is obtained.

4.1 Creating a List of Possible Minor Edit Variables

Two types of minor edit variable requests can be defined. One is the basic request where a minor edit request flag is associated with a given pointer array and any of the elements within the array can be requested. The other request is for a subregion, where a second array or subregion is associated with an element of a primary array. An example would be the node temperatures associated with each heat conductor. The procedures for using both of these minor edit request types are discussed in the follow sections

4.1.1 Primary Minor Edit Request Flags

For a variable to be accessible as a minor edit, it must be included in the minor edit list, which contains information about the variable type, its location and other attributes. The list is contained in the **me_list** array, which uses a derived data type to save the required location and attributes for each variable. Each variable that can be accessed as a minor edit is included in the list. Table II.4-1 summarizes the derived type data structure for the list contained in the **me_list** array.

Each candidate minor edit is assigned a unique request flag, which is currently a four-character alpha numeric string stored in field **me_flag**. The variable must be associated with a pointer array. Field **pntr_type** is an integer flag that identifies the pointer type, e.g. **real**, **integer** or **logical**. The **x_edit_pntr (x=r, i, l, r2, i2, or l2**) for the type given by **pntr_type** will be defined as indicated in

Table II.4-1

Minor Edit Variable List Data Structure

Field Name	Type[1]	Description
r_edit_pntr	R-rkd	Pointer to first word of array for real minor edit variable
r2_edit_pntr	R-rkd	Pointer to first word of array for real 2-D minor edit variable
i_edit_pntr	I-ikd	Pointer to first word of array for integer minor edit variable
i2_edit_pntr	I-ikd	Pointer to first word of array for integer 2-D minor edit variable
me_offset_pntr	I-ikd	Pointer to index array for first element in minor edit pointer array. Used with subregion minor edit variables.
me_nod_pntr	I-ikd	Pointer to number of sub-region nodes
l_edit_pntr	L-lkd	Pointer to first word of array for logical minor edit variable
l2_edit_pntr	L-lkd	Pointer to first word of array for logical 2-D minor edit variable
pntr_type	I-ikd	Pointer type flag -- 1 = real 2 = integer 3 = logical
me_blk	I-ikd	Block number minor edit belongs to.
me_units	I-ikd	Integer index identifying minor edit units in the **conv** array in **module unit.**
me_flag	C-ckd len=8	Minor edit request flag
me_title	C-ckd len=12	Minor edit title string for the associated variable

[1]C = Character, I = Integer, L = Logical, R = Real – size. Parameters on right are defined in module kind_specs and indicate the size.

the table above. The two other pointers will have dummy values assigned since any given variable can only be associated with a single pointer type.

The **me_blk** field indicated the data block number to which the minor edit variable belongs. For example, there will be separate blocks described for volume and junction variables. Each block will have a description, which is contained in the block list. This list is also a derived data type array. It is described in Table II.4-2. The list will contain a separate entry for each data block for which a minor edit variable is defined.

Subroutine **build_minor_edit_list** contains the coding used to build the minor edit variable list contained in **me_list**. It is contained in **module minor_edit_search**. The following code segment illustrates how minor edits are added to the **me_list** for the data block for control volumes.

Table II.4-2

Block List Data Structure for Minor Edit Variables

Field Name	Type[1]	Description
me_reg_pntr	I-ikd	Pointer to the region number for the block.
me_reg_test	I-ikd	Region number test flag for the block. > 0 - use sequential region test 1 <= region <= **me_reg_test** = 0 - no region test < 0 - region must match valid **me_reg_pntr**
blk_hdr	C-ckd len=8	Block label string -- i.e. ‘vol 999’, ‘system’, etc.

[1]C = Character, I = Integer, R = Real - size. Parameters on right are defined in module kind_specs and indicate the size.

```
!
      do l = 1, 2
         allocatd = l.eq.2
         n        = 0
         blk_num  = 0
!
!  Add volume related minor edit flags
!-------------------------------------------------------------------------------
!
         reg_test =  -num_vol  ! use olvoln to do region test
!
         call add_blk_entry (blk_num, olvoln, reg_test, ' vol        ')
!
         call add_r_me (n, p      , 'pres    ', 'avg. press. ',  4)
         call add_r_me (n, temp   , 'temp    ', ' avg. temp. ',  1)
```

```
        call add_r_me (n, satp   , 'satp    ', 'sat. press. ',  4)

        call add_r_me (n, satt   , 'satt    ', ' sat. temp. ',  1)
        call add_r_me (n, satvf  , 'stvf    ', '  sat. vf   ',  6)
        call add_r_me (n, satvg  , 'stvg    ', '  sat. vg   ',  6)
        call add_r_me (n, sathf  , 'sthf    ', '  sat. hf   ',  7)
```

The **do** loop executes the coding twice, the first time to compute the number of entries that will be included in **me_list** and the second time to make the list entries. The call to subroutine **add_blk_entry** makes the entries in the block list array **blk_list**. This list contains the text header information contained in field **blk_hdr.** It is used for minor edit variables that belong to a given block. Three different types of region testing can be used to validate a minor edit variable region number that is part of a minor edit request pair. Field **me_reg_test** describes them. In the example code shown above, region numbers are checked against the content of the **olvoln** pointer array for minor edits that belong to the volume block. Other region checks include no region check which would be used for global values where there is no associated region, and sequential regions checking. Sequential check requires the region number to be bounded by 1 and the number of array elements for the block, e.g. the number of core sections.

A region definition made using a call to subroutine **add_blk_entry** must be made for each data block for which candidate minor edit variables are to be included in the **me_list**. After the associated **add_blk_entry** call, a call to subroutine **add_x_me** (where **x=r, i, l, r2, i2, or l2** defines the pointer type**)** must be made for each candidate variable. The type of the called subroutine must agree with the pointer array passed as the second argument. Argument three is the value of **me_flag**, argument four in the title string **me_title**, and the last argument is the index for the units as defined in **module unit**. The field values in the **me_list** for a minor edit variable are added on the second pass through the loop.

4.1.2 Subregion Minor Edit Request Flags

A special form of a minor edit request is used for subregions, where the subregion has a pointer array that contains a block of data that is associated with a single element of a primary array. An example of a subregion is the temperature array that contains the internal node temperatures for a given heat conductor. As such, there are num_cond heat conductors, but each conductor has an array defining the internal node temperatures. Each temperature array can be a different length depending on the nodalization specified in the input model. The following code fragment illustrates how the node temperatures are defined as minor edit variables for heat conductors using the subregion feature.

```
!
!  Add heat conductor related minor edit flags
!-------------------------------------------------------------------------------
!
        reg_test      =  -num_cond  ! use olslbn for region test
!
        call add_blk_entry (blk_num, olslbn, reg_test, 'cond        ')
!
!        if (nslb .le. 0) then
           call add_r_me (n, se     , 'se**    ', 'stord energy', 10)
```

```
            …
            …
            …
            call add_r2_me (n, xlocl , 'xlor    ', 'r local qual sp. heat ', 7,2)
!
!        Subregion minor edit variables
!
            call add_subreg_me (n, tp      , idxtp   , nnode , 't       ', &
                                'node temp   ',  1)
!
         end if
```

In the code fragment above, the primary heat conductor minor edit variables are added to the **me_list** in a manner similar to that discussed above for the volume variables. Subregion edit variables for the node temperatures are added to **me_list** using the call to subroutine **add_subreg_me**. They are defined using a minor edit request flag, **me_flag**, with the left most character(s) being alpha only and the right most characters being numeric. In the example above, the fifth argument is the base request flag, or 't '. It will allow for subregion temperature request ranging from 1 to 999, which would correspond to user request flags of '**t001**' and '**t999**' (note that a region number specifying the associated conductor is required with each flag). The flag is four characters in length so a two-character alpha field, e.g. '**tp** ' would allow two characters for the subregion.

The second argument in the call to subroutine **add_subreg_me** in the example code fragment is the real pointer array that contains the node temperatures that correspond to the minor edit request flag. Argument three is an integer pointer array that contains the index into the subregion array (**tp**) for a conductor. For a given conductor (identified via the region check as discussed above), the third argument or pointer array **idxtp**, contains the index for the **tp** array that contains the first node temperature. The fourth argument is an integer pointer that contains the number of elements or nodes in the subregion or **tp** array. The last two arguments define the label **me_label** and units index **me_units**, respectively. Currently, only real subregion arrays can be included in the **me_list**. A new subroutine with type specific pointer arrays (argument two) would have to be added to allow subregion minor edits with integer or logical data types.

4.2 Checking for Valid Minor Edit Variable Flags

Minor edit variable requests made by users are checked by first identifying if the request is for a valid variable, e.g. one that is in the minor edit variable list **me_list**. Subroutine **me_search** is used to perform this check. While minor edit request flags included in **me_list** are lower case, e.g. **se**** in the above example, **me_search** is coded such that the search is case insensitive. As such, either **se**** or **SE**** will be found as valid minor edit requests for conductor stored energy. Module **me_search** is included in **module minor_edit_search**. After a valid minor edit variable is found, the region number is checked.

For minor edits in the volume block as shown in the code fragment above, they are checked against all valid volume numbers (input values). If a match is found, the index for the matching **olvoln** will also be the index for the volume variable specified. Subroutine **minor_edit_search** writes

error messages to the output file as appropriate if an illegal minor edit variable request flag or region number are input. When a valid request is made, **minor_edit_search** returns an index into the **me_list** array for the variable and an offset that identifies the particular element within the variable pointer array indicated by the region number. The offset is determined as part of the region checking. For example, the region testing for the coding example above indicated that pointer array **olvoln** is to be used for variables in the volume block. When a valid request flag is f ound, the associated region is checked by looping through all elements of **olvoln** until a match is found. Once a match is found, the index for the matching **olvoln** value is also the index for the particular variable. It is the index returned by **me_search** as the offset for the pointer array.

The offset for a minor edit variable whose associated block uses sequential region checking is simply the region number. For variable with no region checking, the offset is zero.

4.3 Retrieving the Value of a Minor Edit Variable

Function **me_value** is used to obtain the value of a minor edit variable. It is a real function, indicating that the minor edit variable is returned as a real value. It requires that the index into the **me_list** and pointer array offset be supplied. These values are saved for each minor edit variable used as either a minor edit request or control block input. They are returned by **me_search** when it is called with a valid minor edit flag and region number. The derived type data element in **me_list** for the requested minor edit contains the pointer type, pointer, and other attributes for the variable. This information is used to retrieve the value, make any data conversions that are required to return the variable as a real data type. False logical variables are returned as 0.0d0 and true values are returned as 1.0d0. Direct mode conversion is used for integer variables.

5.0 MAJOR EDITS

Prior to the Fortran 95 version of RETRAN-3D, major edit output files were written using indirect addressing. This technique had its purpose at one time, but it was no longer viable and the scheme was overly complicated and made it difficult to revise output edits. Consequently, the major edit subroutines (subroutine **edit** and all other edit related subroutines that it calls) were re-written to use direct I/O lists using variables and pointer arrays defined in data modules. For this reason, they are now straightforward and very easy to modify.

6.0 CODE DOCUMENTATION

Documentation for the RETRAN-3D code is included in "RETRAN-3D - A Program for Transient Thermal-Hydraulic Analysis of Complex Fluid flow Systems, Volume 1: Equations and Numerics; Volume 2: Programmer's Manual; Volume 3: User's Manual; and Volume 4: Assessment Manual". The Fortran source code is another important piece of the RETRAN-3D documentation, which internally provides basic code documentation via the liberal use of comment statements. Comments are inserted at the beginning of each subroutine or function

defining the role each subprogram and/or entry point plays. Along with each subprogram definition, the argument list is also defined, noting the specific definition of the arguments, whether they are input, output, or modified internally (both input and output). All the dynamically allocated storage arrays (see Section II.2.1) are described within the appropriate data module.

Each well-defined task required during execution of one of the RETRAN-3D code package program modules has been coded into a single concise subprogram. Within each subprogram, comments have been included to aid in the understanding of the various blocks of coding. All program modules include a main or primary driver which in turn calls an input processing driver, an initialization driver if required, and a driver which directs the execution of a problem. As an example, execution of the **retran** program module is controlled by the driver subroutine RETRAN. Subroutine **retran** in turn serially delegates the responsibility of input processing to the input processing driver **inrtrn**; generating initial conditions or initial values to the driver **ststat**; and finally execution of the transient thermal hydraulics via the driver **tran**. Few, if any, calculations are performed in the driver subroutines, above the minimum required to direct the program flow. The simplicity of the drivers, coupled with comments describing such redirection in program flow, provides a very useful and necessary level of documentation, very similar to a general flow chart.

Section III contains a description of the program options and shows the calling structure of the RETRAN-3D program.

7.0 ERROR MESSAGE HANDLING

Error messaging in RETRAN-3D is handled by use of subroutine **errlog.** It writes error messages and associated information to the output file and a separate errlog file (see Section IV.2.10 for additional details). All errors are assigned unique error numbers and supplemental documentation provides discussion of the error and what a user can do to correct the error. Input processing errors, memory allocation, and errors encountered during the transient solution are handled using the **errlog** subroutine.

The intent of using this method for handling error conditions and writing the associated error messages is to provide users with error messages that are clear and concise and provide direction for correcting the error. Since a useful error description may be longer than a few lines and in some cases the corrective action may have multiple solutions depending on conditions encountered, they are contained in Appendix C of the User's Manual – Volume 3.

Information is passed to subroutine **errlog** by setting the necessary information in the **ERRDATA** communication module prior to calling **errlog.** Subroutine **errlog** will use the necessary information to write a concise and useful error message for the code user. The character descriptions for the error generally contain most of the error description. Other character information describes the subroutine and/or module where the error occurred.

By using subroutine **errlog** to write all error messages a standard look and feel for the error messages can be attained. Most error messages can be processed by simply writing the error

description character variables. When additional information is needed to enhance the content of the error message, the necessary write statements and formats can be added to subroutine **errlog** for the particular error. Subroutine **fail** is also called for any call to **errlog**. This will terminate the problem with an error after processing all or most of the remaining input data.

The data typing and description for the content of the **ERRDATA** module is shown below. It contains character strings that are used to pass subroutine names and other descriptive information to **errlog** as well as integer and real arrays used to pass values that help describe the error conditions that lead to the error.

```
      integer(kind=ikd) , parameter :: &
         lenvec = 50 ,  &    !  dimension of the integer and real
                                communication vectors
         lentbl = 500       !  dimension of the vector use to pass the polate
                                table to errlog
!
      real(kind=rkd) ::  &
           vect(lenvec), &   !  real communication vector
           pol8(lentbl)      !  vector used to pass failed polate table to
                                errlog
!
      integer(kind=ikd) ::  &
           card_num,  &      !  card number
      err_count = 0,  &      !  cumulative error count
            err_num,  &      !  current error number
        locvec(lenvec)       !  integer communication vector
!
       character(50) :: &    !  Transient error processing string
       err_des1,  &          !  string describing the error condition (1st
                                part)
       err_des2              !  string describing the error condition (2nd
                                part)
!
       character(100) :: &   !  Input error processing string
       err_des3,  &          !  string describing the error condition (1st
                                part)
       err_des4              !  string describing the error condition (2nd
                                part)
!
       character(16) :: &
       component             !  component name
!
       character(18) :: &
         decknam,  &         !  subroutine name
       modulenam             !  module name
```

Any new information that may be required by **errlog** should be defined in this module.

When adding error messages to RETRAN-3D, **errlog** should be used to write the error message and associated information. Additionally, Appendix C of the User's Manual - Volume 3, should also be revised to contain a description of the error and the appropriate corrective action. Remember the target audience is the code user, not a code developer. Information may be included with the message that would be useful for a code developer, but the primary focus should be on the user.

Character variables are used to define strings that **errlog** will write in the output file and errlog file. They are defined in the calling subroutine before calling **errlog.** Other character information, integer or real values may also be passed to **errlog** for inclusion in an error-specific error message.

When adding new errors, the programmer may identify an existing category or define a new one, consistent with the existing use. A new error number will also be assigned.

Error information stored in module **ERRDATA** variables is processed by subroutine **errlog**. This subroutine parses the error number as necessary to obtain the error category and two-digit error number and decides the appropriate error type, i.e., input processing error, memory allocation error, or run-time error. The appropriate header information is edited with the error type, then error number specific information is edited.

The following sections contain directions for specific input processing errors, memory allocation errors, and errors encountered during the transient solution.

7.1 Input Processing Errors

Input processing errors are detected as the input is read and processed, prior to executing the transient solution. Each error is assigned a unique four- or five-digit error number. They are divided into categories based on the associated input card number specified in Volume 3, User's Manual. For example, the error categories are 50 and 80 for volumes and junctions, respectively. Each error category allows up to 99 separate errors, so volume related input processing error numbers can range from 5001 to 5099.

The general categories for input processing errors follows.

Error Number	Type of Error
	Problem Control
1000 - 1099	Problem Description Input
1500 - 1599	Initial Power
1800 - 1899	Noncondensable
2000 - 2099	Minor Edits
3000 - 3099	Time-Steps
4000 - 4099	Trips
	Volume
5000 - 5099	Volumes Input
6000 - 6099	Bubble Rise Model
6100 - 6199	Profile Fit Model
7000 - 7099	Time-Dependent Volume Input

Error Number	Type of Error
	Junction
8000 - 8099	Junctions Input
63000 - 63099	Junction Enthalpy Model
8200 - 8299	Two-Region Enthalpy Transport
	Components
9000 - 9099	Centrifugal Pumps
11000 - 11099	Valves
12000 - 12099	Generalized Data Tables
13000 - 13099	Fills
	Kinetics
14000 - 14099	Point Kinetics Model
14100 - 14199	Scram Data
14200 - 14299	Density Reactivity Input
14400 - 14499	Moderator Heating
14600 - 14699	Decay Heat
	Heat Conductor
15100 - 15199	Heat Conduction Driver
15000 - 15099	Heat Conductors
16000 - 16099	Core Conductors
17000 - 17099	Heat Conductor Geometry
18000 - 18099	Material Property
21000 - 21099	Nonconducting Heat Exchangers
22500 - 22599	Dynamic Gap Conductance
	Steady-State Initialization
23000 - 23099	Volume Initial Conditions
23100 - 23199	Junction Initial Condition
23200 - 23299	Steady-State Initialization
23600 - 23699	Off-Rated Power Initialization
	1-D Kinetics
30000 - 30099	Space-Time Kinetics
31000 - 31099	Cross-Section Data
	Iterative Solver
38000 - 38099	Purdue Numerics (T-H)

Error Number	Type of Error
	Auxiliary Models/Components
45200 - 45299	General Transport
50000 - 50099	Turbines
60000 - 60099	Separator
61000 - 61099	Two-Region Nonequilibrium
62000 - 62099	Accumulator Input
63000 - 63099	Method of Characteristics
	3-D Kinetics
67000 - 67099	3-D Kinetics Input
67100 - 67199	Channel to Assembly Mapping
67200 - 67299	Channel Data
67300 - 67399	Cross Flow Data
67400 - 67499	Control Rod
67500 - 67599	Bypass
67600 - 67699	Boron Injection
	Control System
70000 - 70099	Control Input and Blocks
	Miscellaneous
80000 - 80099	DNBR
90000 - 90099	3-D Kinetics, CDI File, Cross Section

When multiple error conditions can be detected in a subroutine, the common information is generally defined once at the beginning of the subroutine. It is then available for use any time an error message may be written by calling subroutine **errlog**. A coding fragment that illustrates this application is shown below.

```
!
!  Setup common input processing parameters in case input errors
!  are encountered
!
      component = 'VOLUME  '
      decknam   = 'INVOL   '
```

The follow coding fragments illustrates the additional set up required prior to calling subroutine **errlog** to write the error.

```
!
!  Check to see if obsolete card numbers (05000x) were used for
!  laminar friction model input - if so, write error message.
!
      istrt = 050001
      iend  = 050009
      incr  = 1
      nold  = ncards(istrt,iend,incr,inp_tbl)
```

```
      if ( nold > 0 ) then
         err_num  = 5015
         card_num = 50001
         err_des3  = 'LAMINAR WALL FRICTION MODEL COEFFICIENTS WERE INPUT
                      USING OBSOLETE 05000X CARDS.'
         err_des4  = 'USE 2332XX CARDS INSTEAD.'
!
         call errlog
      end if
!
```

Second coding fragment example.

```
!
       err_num   = 5027
       card_num  = 0
       write (err_des3, "('TIME DEPENDENT VOLUME TABLE '1i5,' SPECIFIED FOR
                          VOLUMES'1i6,' AND '1i6,'.')" iread(i), olvoln(i),
                          olvoln(j)
       err_des4  = 'A UNIQUE IREAD MUST BE USED FOR EACH TIME DEPENDENT VOLUME.'
!
       call errlog
```

Both error messages set up in the coding fragments above will be written as illustrated in the coding fragment from subroutine **errlog**.

```
!    input processing errors
!
        write(17,10000) err_num, component, decknam, card_num
        write(6,10000)  err_num, component, decknam, card_num
!
        itype = err_num/100
        num   = err_num - 100*itype
!
!       Volume input processing errors
!
        if (itype .eq. 50) then
           if (num>=1 .and. num<=20) then
              write(17,10001) err_des3, err_des4
              write(6,10001) err_des3, err_des4
           else if ……
              ! special case error messages here
           end if
        end if
```

7.2 Memory Allocation Errors

Most memory allocation occurs during input processing. When errors associated with memory allocation occur, subroutine **errlog** is used to write the associated error messages. Memory allocation errors are assign numbers 100, 101, and 102.

The following coding fragments illustrate how the error messages would be set up for two different error conditions that might be encountered when memory is allocated.

```
!
!   Check to insure memory request is positive
!
      if (num_idc <= 0) then
         err_num   = 101
         component = 'CONTROL SYSTEM'
         decknam   = 'ALLOK8_CSUB'
         modulenam = 'CONTROL_SYSTEM'
         err_des3  = 'ERROR ALLOCATING SPACE FOR CONTROL SYSTEM ARRAYS. SUM
                      OF NUMBER OF '
         write (err_des4, "('CONTROL BLOCK INDEXES = '1i6,', MUST BE >
                           0.')" num_adc
!
         call errlog
!
         return
      endif
```

Second example.

```
!
      allocate (i_cidx(siz_icidx), stat=status)
!
!  Check status of allocate - write error if allocate failed
!
      if (status .ne. 0) then
         err_num = 100
         component = 'CONTROL_SYSTEM'
         decknam   = 'ALLOK8_CSUB'
         modulenam = 'CONTROL_SYSTEM'
          write (char1,'(i6)') siz_icsub
          write (char2,'(i6)') status
         err_des3  = 'ERROR ALLOCATING SPACE FOR INTEGER CONTROL SYSTEM
                      INDEXING ARRAYS.'
         err_des4  =  'LENGTH REQUESTED = '//char191:6)//' STATUS FLAG RETURNED
                       FROM ALLOCATE = '//char2
!
         call errlog
     endif
```

7.3 Transient Solution Errors

Transient solution or run-time errors are assigned unique three-digit error numbers. They are divided into eight general categories as follows.

Error Numbers	Type of Error
101 - 199	Numerical Solution
201 - 299	Equation of State (Pressure Solution)
301 - 399	Model Limitation
401 - 499	Heat Transfer
501 - 599	Bubble Rise/Separator Volume/Pressurizer
601 - 699	I/O Handling
701 - 799	Kinetics
801 - 899	Utility Routine or Problem Control

Up to 99 errors can be defined for each category. For more information refer to Appendix C of the User's Manual – Volume 3.

The following code fragment illustrates the use of subroutine **errlog** to write a simple error message from the transient solution. It identifies a path that should not be encountered unless there is a programming error. Note that the volume number is passed to subroutine **errlog** via variable locvec(1).

```
!
!     Code error
!
         else
            component = 'VOLUME  '
            locvec(1) = olvoln(i)
            err_num   = 104
            decknam   = 'PRESUR  '
            err_des1  = 'ILLEGAL PRESURE SEARCH PATH - THIS IS PROBABLY '
            err_des2  = 'DUE TO A CODE ERROR'
!
            call errlog
            go to 100
         end if
```

Error processing for the transient solution is complicated by the fact that the iterative time-step advancement scheme often mitigates errors that are encountered. For example, recovery from a pressure search failure typically occurs by reducing the time-step size and resolving the time step. When the reset eliminates the error, the error message should not be written, but the information related to the failure should be available if the error is not resolved by reducing the time-step size. The following example illustrates how the pertinent information is stored into **ERRDATA** variables so it will be available for later use. Note that subroutine **errlog** is not called in the coding that saves error condition information for later use if needed.

```
!
!     Check for error condition during pressure search
!
         if ( .not.(nogo(1)) ) then
!
!        Error already detected - do nothing more
!
            if (hw(i) .eq. -one) then
```

```
               call fail
            else if ( (hw(i)<=zero) .or. (hw(i)>2000.0D0) .or. (p(i)<plow) .or.
                      (p(i) > phigh) ) then
!
               err_num   = 221
               err_des1  = 'PRESSURE SEARCH CONVERGED TO A PRESSURE OR '
               err_des2  = 'ENTHALPY BEYOND RANGE OF WATER PROPERTIES '
               component = 'VOLUME  '
               locvec(1) = i
            end if
         end if
```

In the event that the iterative solution does not resolve the error, subroutine **errlog** will be called to print the error message.

III

RETRAN PROGRAM FLOW

There are three program options in the RETRAN-3D computer program. Program option RETRAN is used to initialize and execute a transient thermal-hydraulics calculation, while the RESTRT option is used to continue calculations from data saved on a RETRAN-3D restart file. REEDIT is a utility option that produces printed edits and printer plots of data archived on a RETRAN restart file. RETRAN, RESTRT, and REEDIT optionally allow for the generation of a user-specified output data file that can be used to interface with other codes. This section is designed to provide an overview of the program flow in each of these three modes of execution. Subroutine call charts are provided to aid the program flow discussion, although they do not necessarily represent the order in which subroutines are called. The call charts show all major subroutine calls (with the exception of FORTRAN library, and operating system subroutine calls). Some subroutines shown are conditionally called, consistent with the program options exercised.

The main program RMAIN is the driver for the RETRAN code package (see Figure III.1-1). It calls subroutines SET_CONFIGURATION, GETPAR, and RET_INPUT. SET_CONFIGURATION initializes solution constants and dimensions, some of which can be overriden by entries in the RETRAN.cfg configuration file. GETPAR processes parameters that are provided at execution time, e.g., a flag to disable screen writes. Subroutine RET_INPUT initializes the INP free form input processing package by calling subroutine INP. It also reads the problem dimension data card to determine the type of problem that is being run (restart, initial run or reedit). This information is returned to RMAIN which selects the appropriate driver subroutine for one of the three program options, which are described in the following sections.

1.0 INITIAL RUNS

Subroutine RETRAN is called from the main program RMAIN as illustrated by Figure III.1-1 for the RETRAN program module. RETRAN directs the program flow to the major computational blocks. Subroutine INRTRN is called first from RETRAN which in turn calls INTRAN. INTRAN calls the input subroutines that read, check, and edit input data (see Figure III.1-2).

When the multi-dimensional kinetics option is used, subroutine GEOM3D is called from INTRAN to process the appropriate input data. It then calls subroutine ARRINP to perform most of the input and related cross section processing. Figure III.1-3 shows the subroutines called from ARRINP.

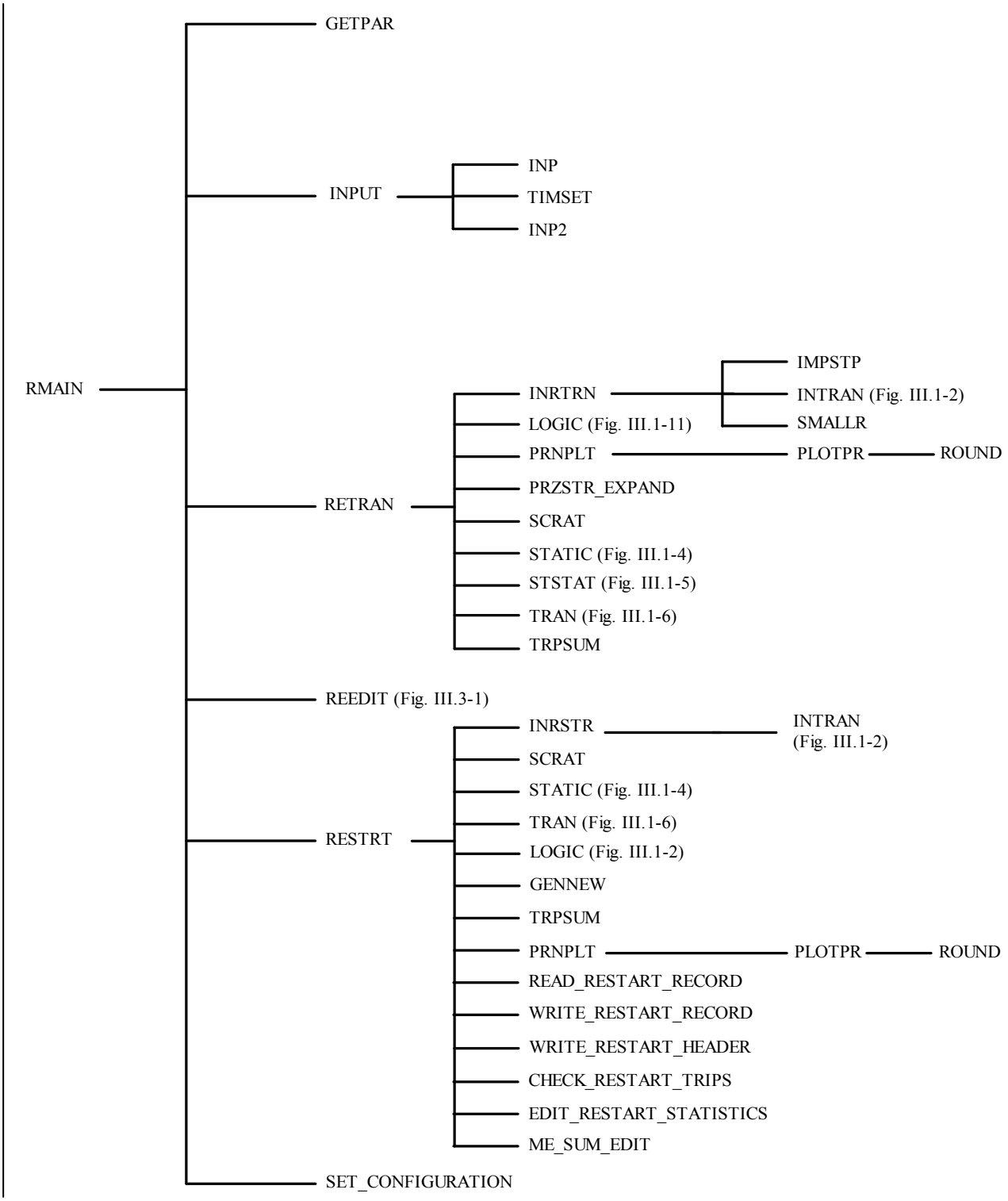


Figure III.1-1. Subroutine Calls from Main Program RMAIN

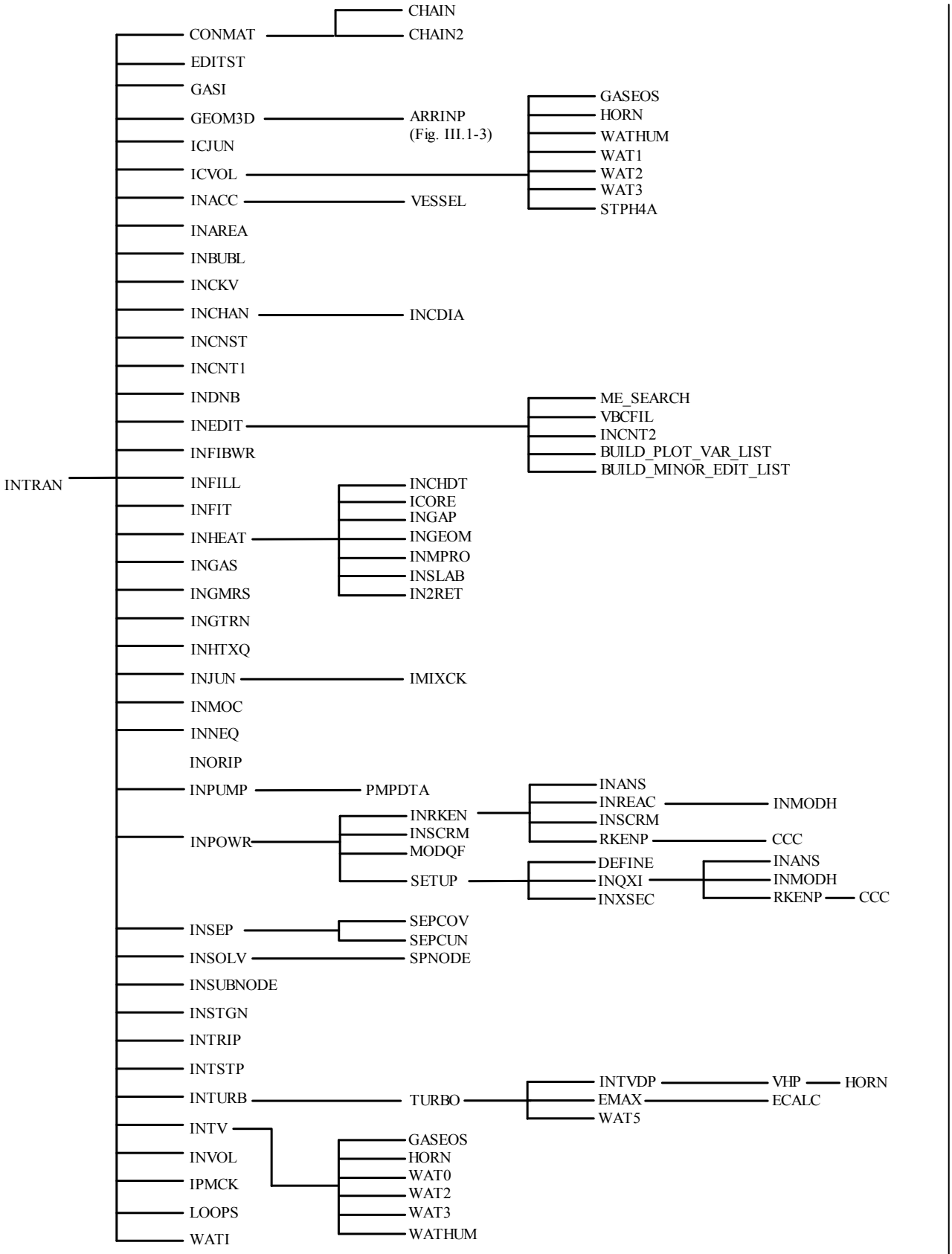


Figure III.1-2. Subroutine Calls from INTRAN

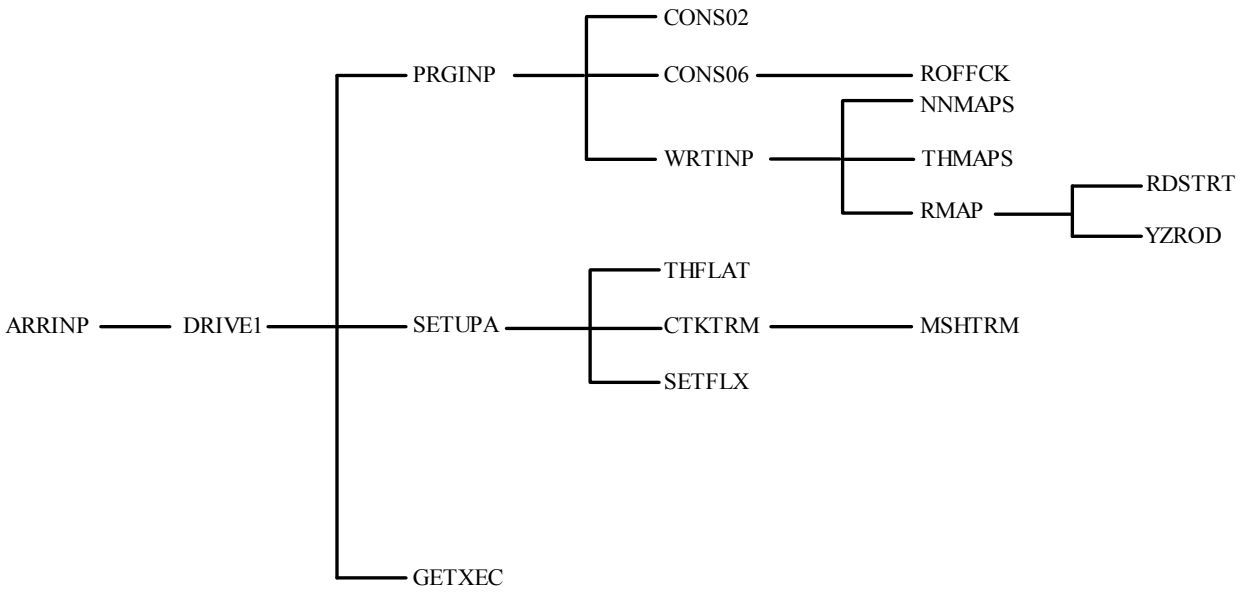


Figure III.1-3. Subroutine Calls from ARRINP

After the input is processed, STATIC is called from RETRAN if the one-dimensional kinetics option is used. STATIC is the driver for one-dimensional kinetics steady-state initialization (see Figure III.1-4). STATIC and subroutines called by STATIC initialize fluxes, the power profile, and the precursor densities. The call to STATIC is bypassed if the one-dimensional kinetics option is not used.

When the stratified pressurizer model is used RETRAN calls PRZSTR_EXPAND to setup the pressurizer and subnodes for steady state. It either expands or contracts the pressurizer volume by merging or extracting subnodes to the pressurizer.

The next subroutine called by RETRAN is STSTAT. STSTAT is the driver for the thermal-hydraulics steady-state initialization (see Figure III.1-5). Initial estimates for control volume state properties and heat source and sink terms are obtained via a call to INITLZ from STSTAT. After returning from INITLZ, an iteration loop in STSTAT calls subroutines PRSORK, JUNPRP, ADVFLO, BUBINT, HAVG or XANDH, STATPH, PRESUR, POSTW, ENERGY, JHOFF, and DERIVS. When the change in initial values for the field equations, constitutive models, and component models from one iteration to the next meets some predefined convergence criteria, STSTAT exits the iteration loop and calls JVEDIT which edits the results of the steady-state solution. Other models are then initialized; the control system by CONTRL, point kinetics by KINITL, the DNB model by DNBM, the one-dimensional space time kinetics model by QX1I, and the iterative numerics option by SAVIMP. When the pressurizer stratification model is used, steady state calls PRZSTR_SETTRAN to setup the pressurizer and subnodes for the transient. It adjusts them according to mixture level and liquid region quality.

TRAN (see Figure III.1-6), the driver for transient calculations, is called by RETRAN. The flow of calculations within TRAN is described in the following discussion. After the converged solution is obtained, and if the pressurizer stratification model is used, TRAN calls PRXSTR_EXPORCONT to expand or contract the pressurizer by merging or removing subnodes. A new time step is selected by TSTP. All trips are strobed to activate coincidence and indirect trips by subroutine TRIP, and the control system blocks are activated by a call to CONTRL.

Subroutine JUNPRP is called to update the junction properties used in the convective terms of the balance equations and subroutine ADVFLO is called to evaluate the terms used to solve the mixture momentum equation. Figure III.1-7 shows the subroutines called from JUNPRP and ADVFLO

The mass transfer rates for the vapor continuity equations are obtained from MDOT. Subroutine SLIP evaluates the terms used to solve the dynamic slip (velocity difference) or algebraic slip equations. Subroutine DERIVS computes the terms needed to solve the volume balance equations. The GENOPT and GENMTx routines are called to set up and solve the coupled set of overall balance equations. The GENMT3 path is used for the three- and four-equation solution options while GENMT4 is used for the five-equation and noncondensable gas flow modeling options.

Subroutine PRESUR (see Figure III.1-9) is then called to obtain new volume state properties given the new volume energy and mass values from the balance equation solution. The fluid thermodynamic and transport properly subroutines used by RETRAN are illustrated in Figure III.1-10. Note that WAT10 is an entry point to subroutine WAT9 and WAT12 is an entry point to subroutine WAT11.

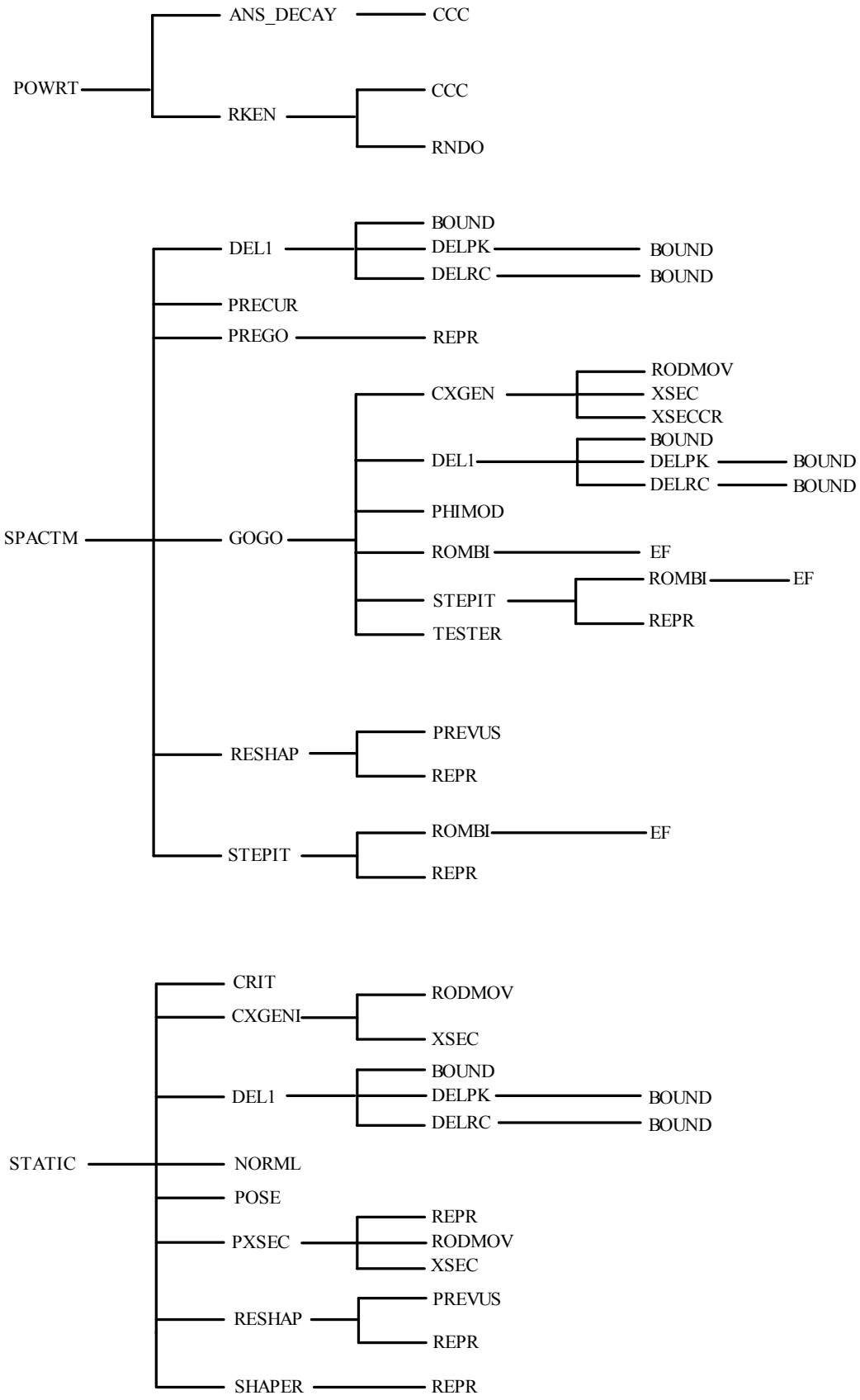


Figure III.1-4. Subroutine Calls from POWRT, SPACTM, and STATIC

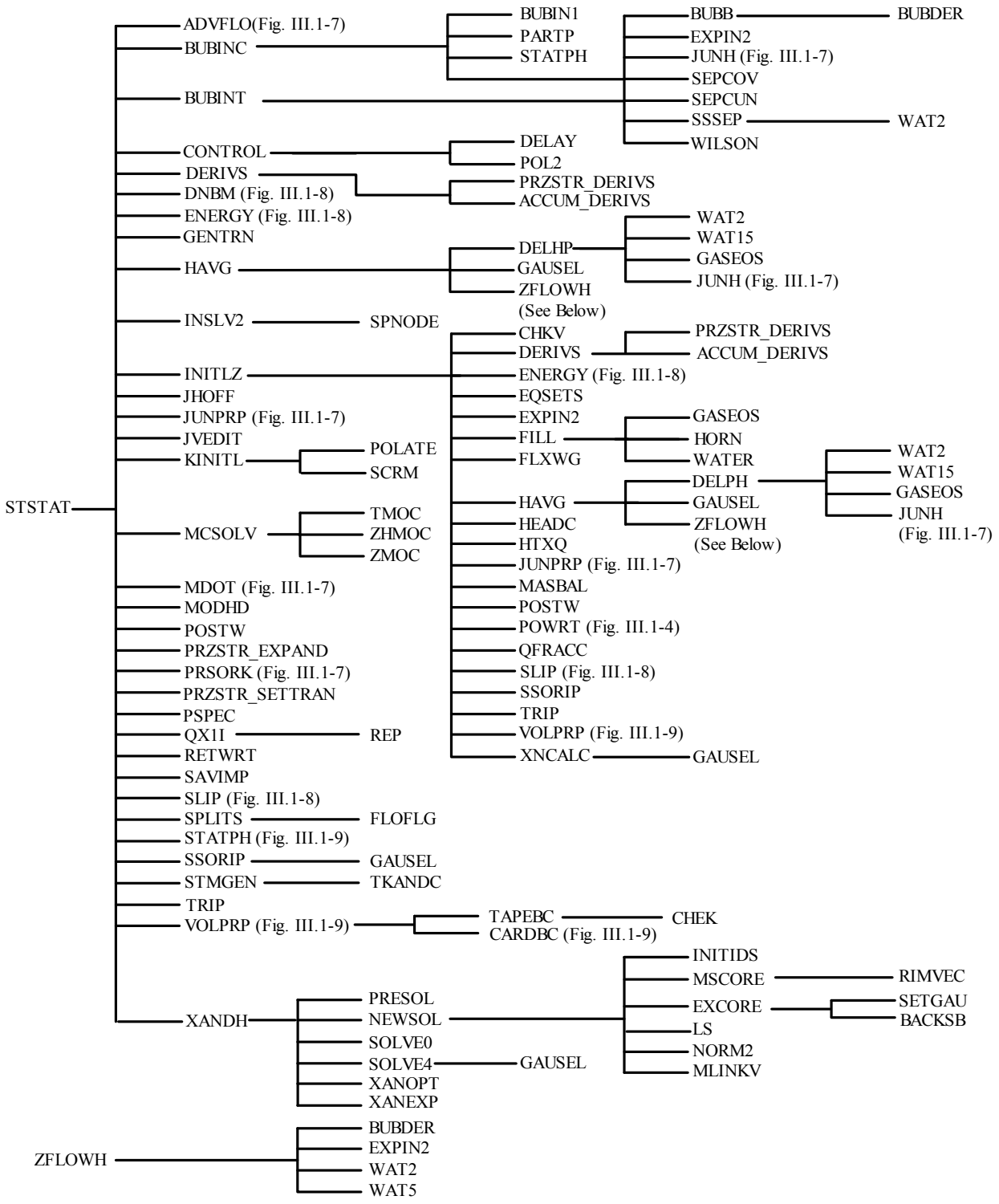


Figure III.1-5. Subroutine Calls from STSTAT and ZFLOWH

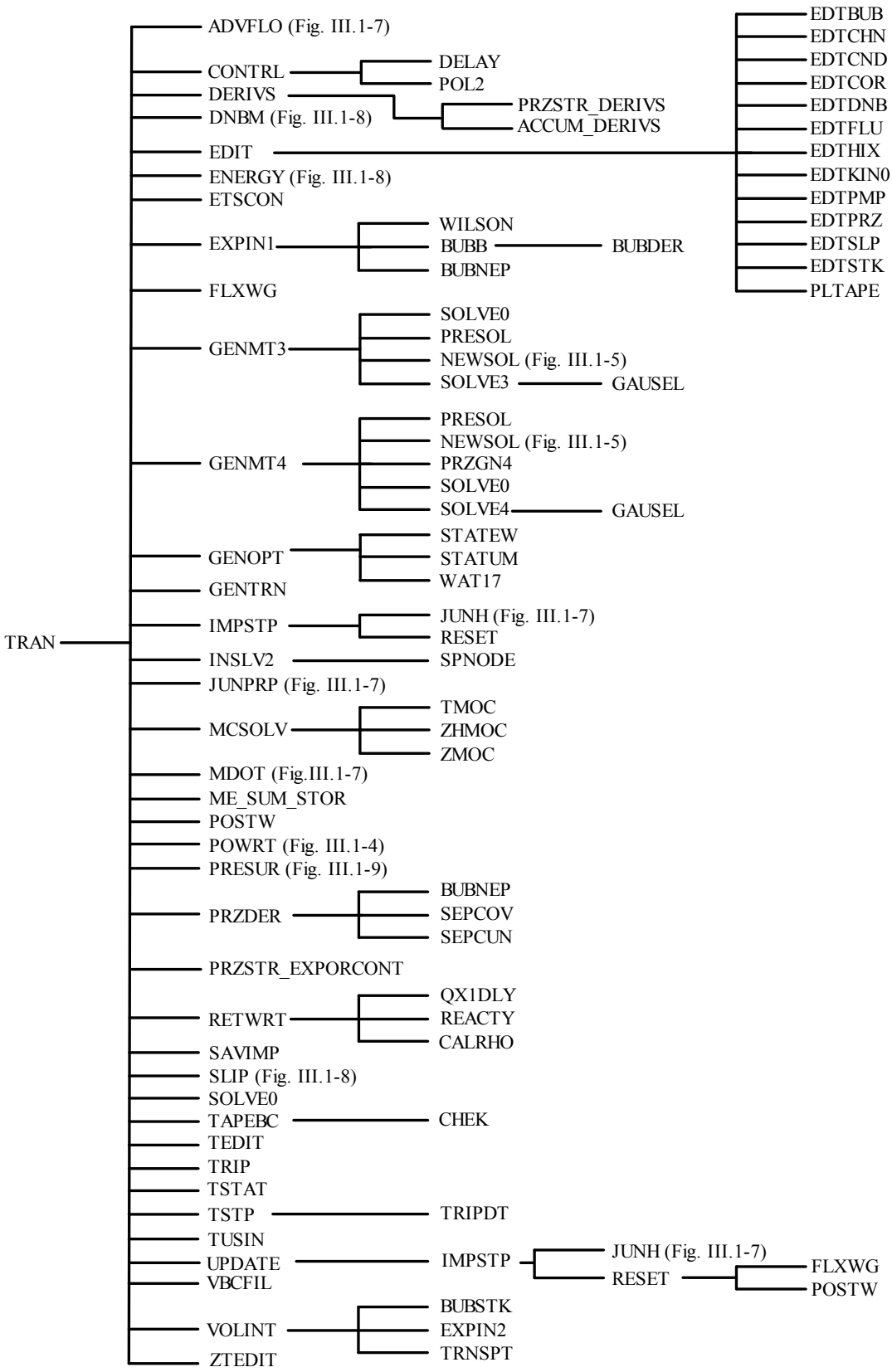


Figure III.1-6. Subroutine Calls from TRAN

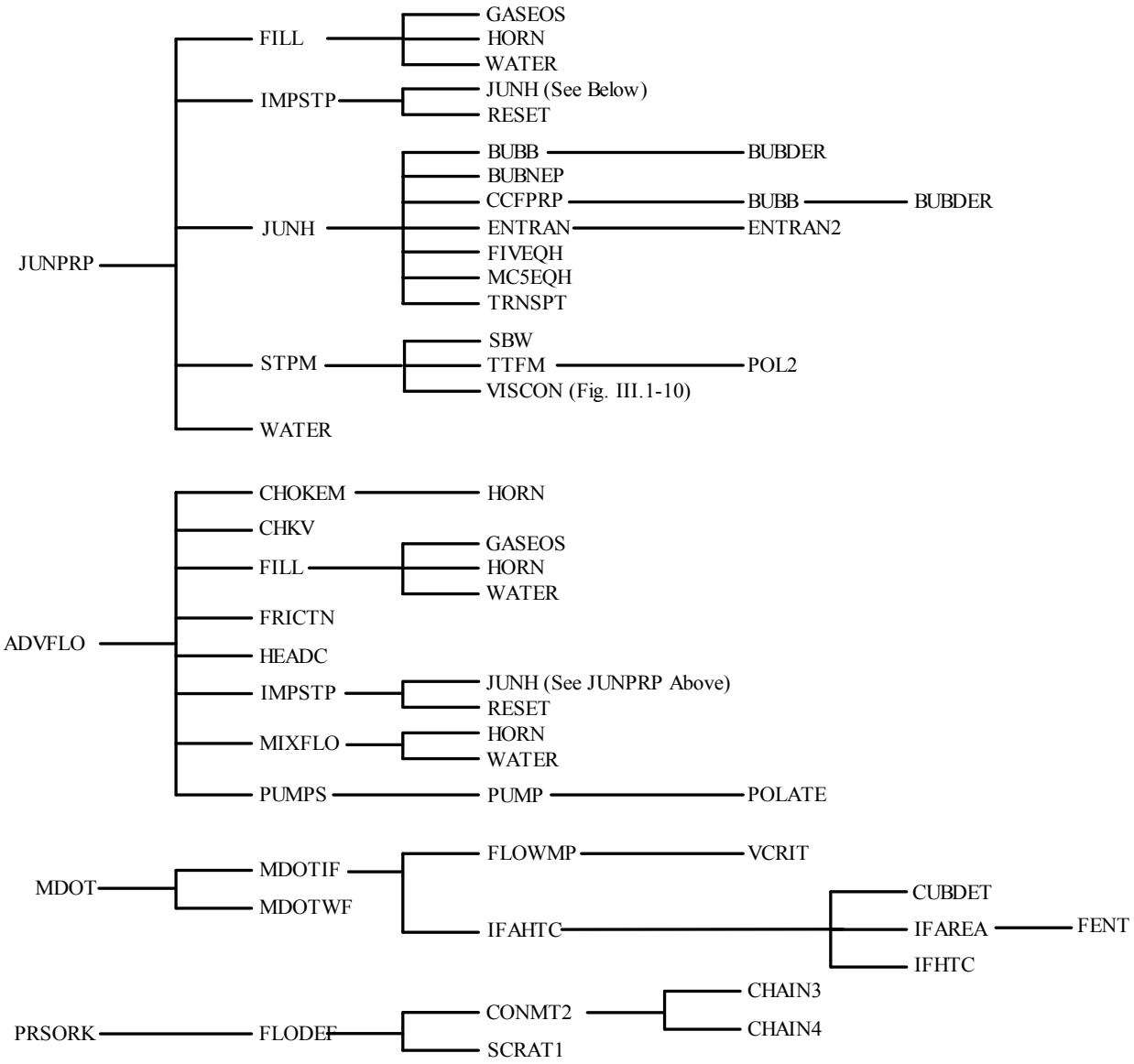


Figure III.1-7. Subroutine Calls from JUNPRP, ADVFLO, MDOT, and PRSORK

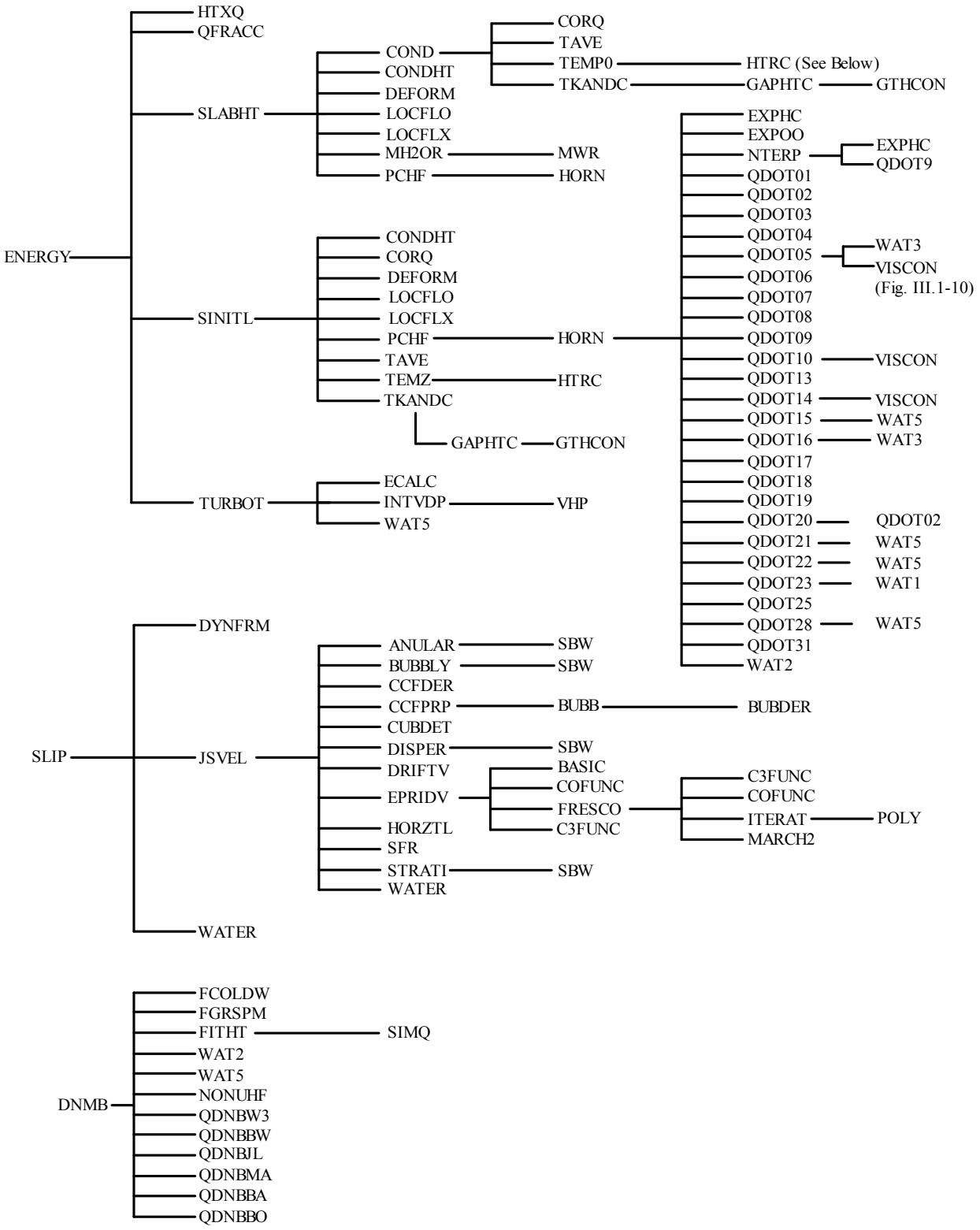


Figure III.1-8. Subroutine Calls from ENERGY, SLIP, and DNBM

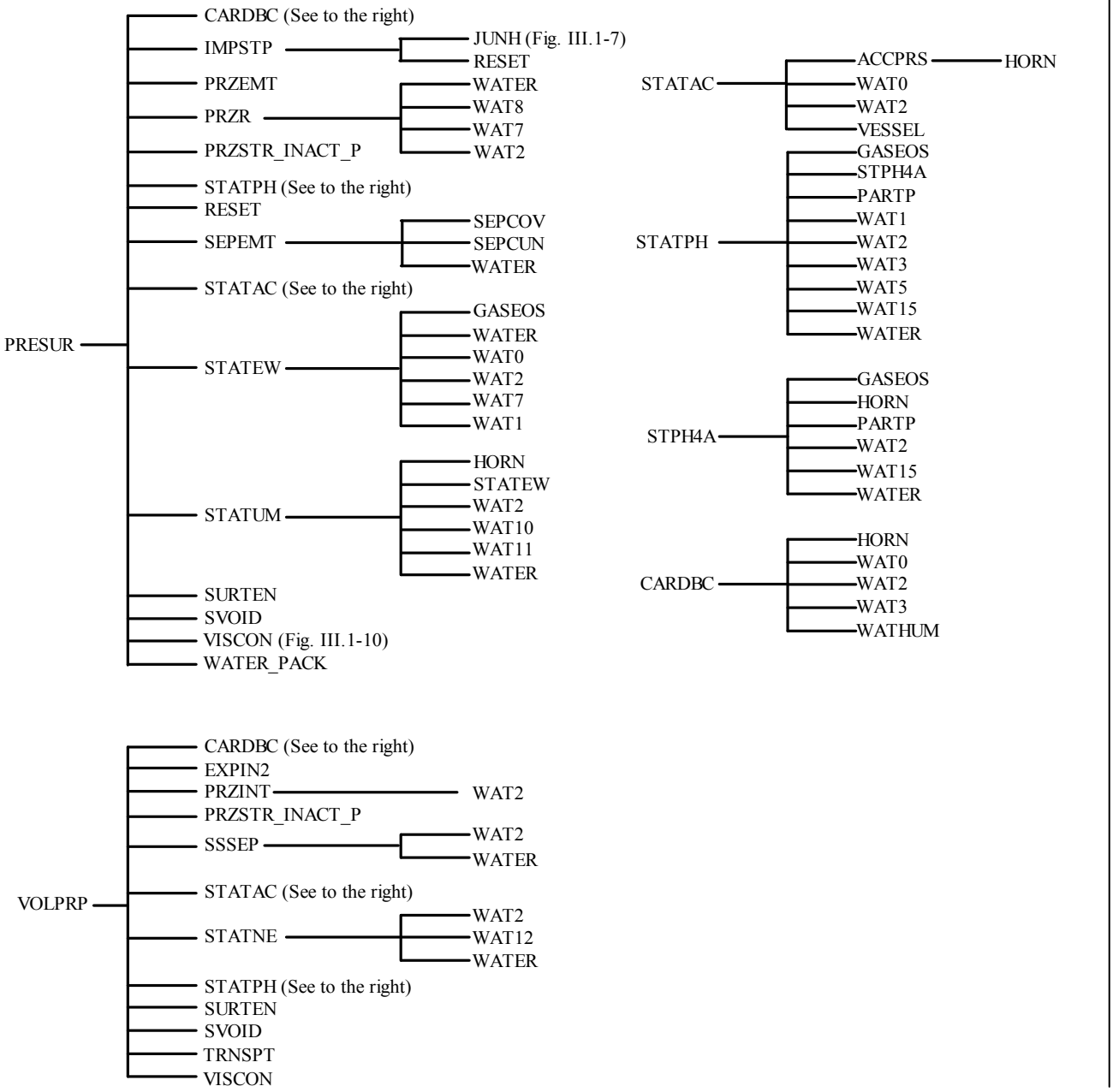


Figure III.1-9. Subroutine Calls from PRESUR, VOLPRP, STATAC, STAPH, STPH4A, and CARDBC

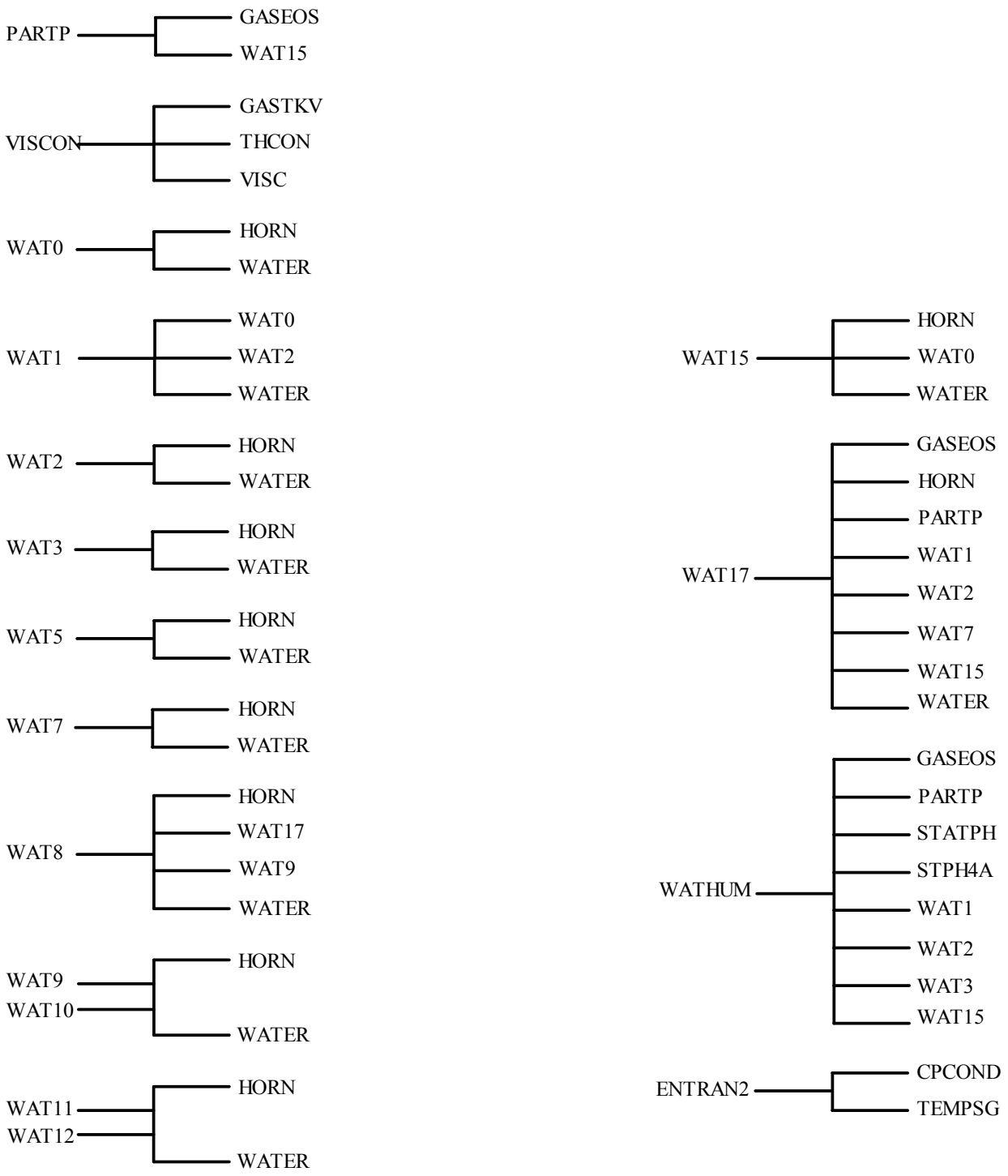


Figure III.1-10. Fluid Property Routine Calls

The new power level is computed in subroutines called from POWRT (see Figure III.1-4). The subroutines called by POWRT depend on the power calculation option used, i.e., tabular power versus time, point kinetics, or one-dimensional kinetics. If the multi-dimensional kinetics option is selected instead of one noted above, subroutine POWRT is not used. Instead, subroutine LOGIC is used. It is illustrated in Figure III.1-11.

Subroutine ENERGY and subroutines called by ENERGY determine the heat transfer rates to or from fluid volumes by heat conduction, nonconducting heat exchangers, or turbines (see Figure III.1-8). If the turbine model option is used, the turbine speed is integrated in subroutine TUSIN. The auxiliary DNB calculation is performed by subroutine DNBM (see Figure III.1-8).

Function TRIP is called to test for the end of problem. Subroutine EDIT is called if it is time to do a major or minor edit. An EDIT option allows the user to write minor edit variables to an auxiliary file (FORTRAN Unit 60) in user-specified format. If a restart file is requested, it is also written from EDIT by making calls to WRITE_RESTART_HEADER, WRITE_RESTART_RECORD, and WRITE_RESTART_TRAILER. EDIT is at the end of the loop at which TRAN transfers back to TSTP to select another time step. Subroutines ZTEDIT and ETSCON are part of the timing package but are not an integral part of the calculations.

At the end of the transient calculations (caused by end trip signal or an error detected), control returns to subroutine RETRAN. Subroutine RETRAN then calls TRPSUM which edits the trip status history of the calculation; then calls PRNPLT which generates printer plots of the minor edit variables if the option is requested.

Control is returned to RMAIN which terminates execution.

2.0 RESTART RUNS

Restarting is defined as continuing a calculation at some point in time from solution results of a previous RETRAN calculation archived on a data file. It is performed by program module RESTRT. Data archived on a RETRAN data file falls into two categories. First, all variables that cannot be computed at restart time and second, all variables that may be required for editing and plotting. All other information that can be recomputed at restart time or variables that do not change (such as geometric information) are obtained by processing the input from the original problem input data.

After calling INPUT, RMAIN calls RESTRT (see Figure III.1-1), which is an entry point in subroutine RETRAN.

RESTRT first calls INRSTR. INRSTR calls CHEK which opens the RETRAN data file from which the calculation is to be restarted and opens another data file for archiving data from the new calculation.

Subroutine INDATA is then called to copy header label, problem description data, and input data from the original data file to the new data file. INDATA also stores the original input data in main memory.

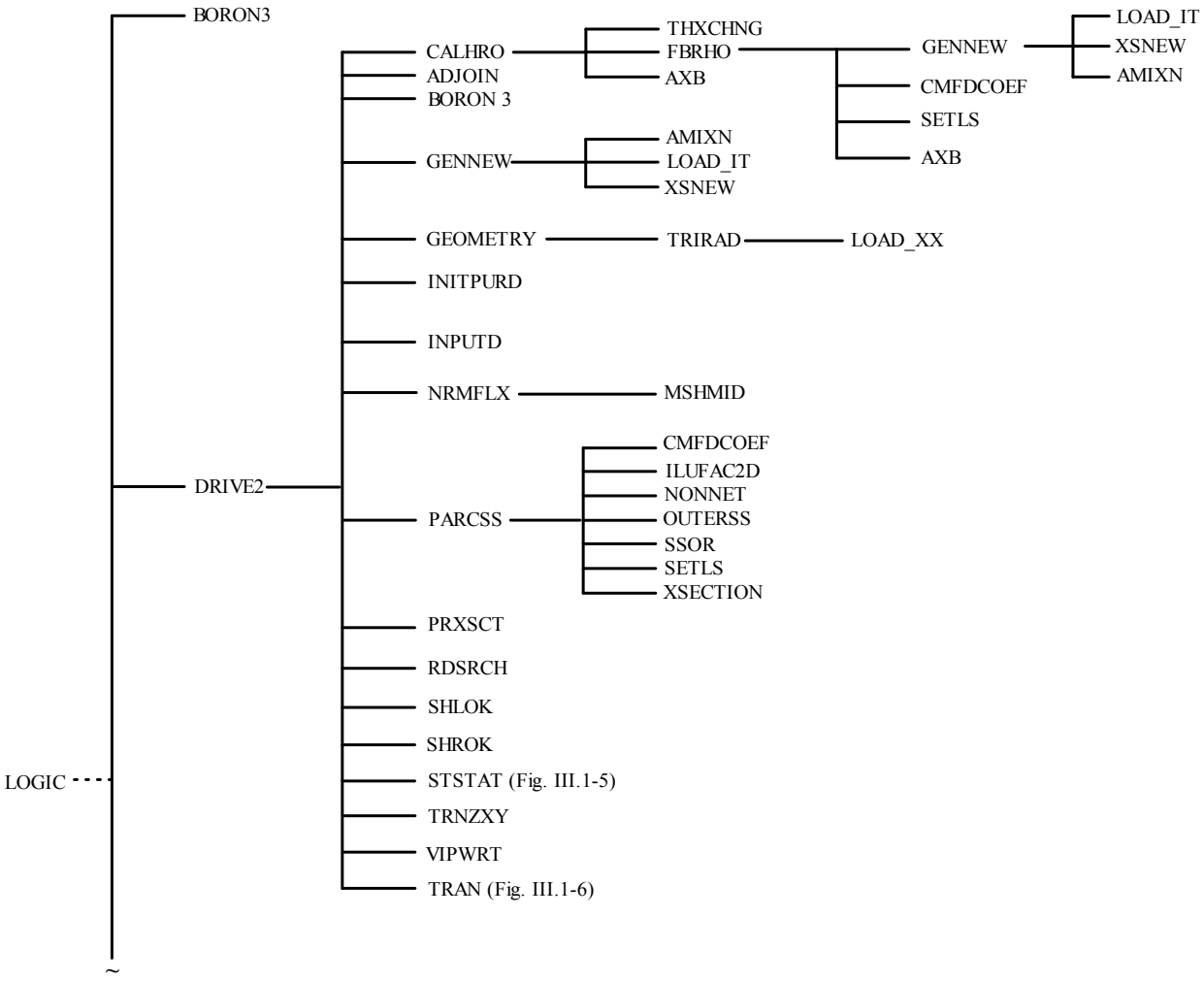


Figure III.1-11. Subroutine Calls from LOGIC

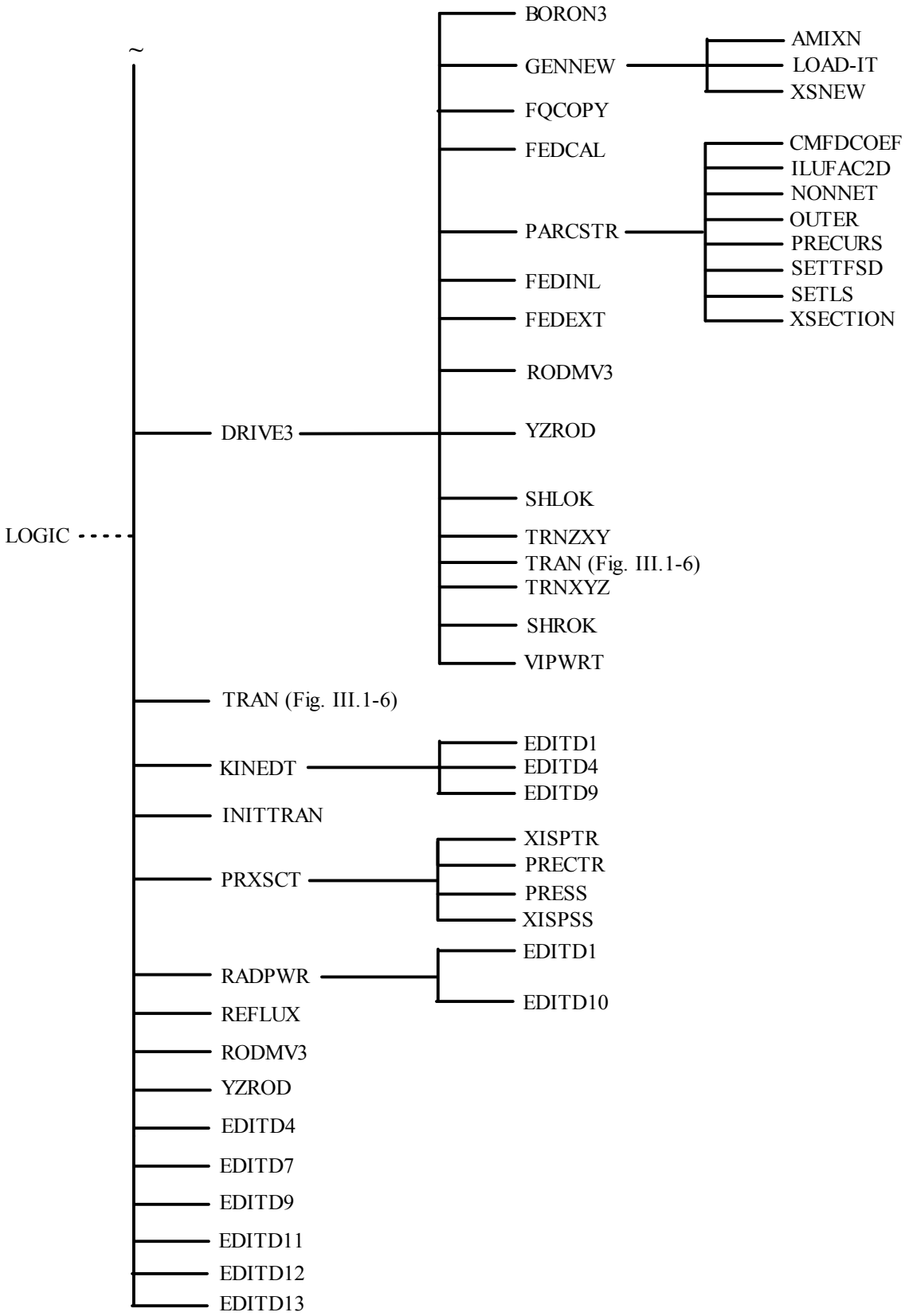


Figure III.1-11. Subroutine Calls from LOGIC (Cont'd)

INRSTR then calls subroutine INTRAN. INTRAN and subroutines called by INTRAN process and store the original input data as if it were an initial run.

Subroutine STATIC is then called by RESTRT to initialize the one-dimensional kinetics. STATIC computes the one-dimensional kinetics quantities not stored on data file and is called only if the one-dimensional kinetics option is used. Subroutine LOGIC is called by RESTART to initialize the three-dimensional kinetics option.

The next subroutine called is READ_RESTART-RECORD, which reads the content of a record into allocated memory blocks. If a new restart file is being written, Subroutine WRITE_RESTART_RECORD is called. This process writes a duplicate restart file and is repeated until the restart record corresponding to the restart time has been read. If a restart file is not created, the calls to READ_RESTART_RECORD skip through the record rather than reading it into memory and no calls are made to WRITE_RESTART_RECORD. Repeated calls are made until the restart point is reached, when the restart record is reached, its data is read into allocated memory.

The program flow proceeds from this point as an initial run, with subroutine TRAN called from RESTRT (see Section III.1 for TRAN description). TRPSUM is called which edits the trip status history and PRNPLT is called to do the optional minor edit variable printer plots at the end of the calculation.

Control is then returned to RMAIN which terminates the execution.

3.0 REEDIT RUNS

The REEDIT program module is used to obtain printed edits (major edits and minor edits) and printer plots of RETRAN problem solutions archived in data files. An option allows the user to write minor edit variables to an auxiliary file (FORTRAN Unit 60) in user-specified format.

Subroutine RET_INPUT is called from RMAIN which initializes the package and reads the problem dimension card to determine the problem type. REEDIT is then called by RMAIN. REEDIT directs the calling of the major computational blocks (see Figure III.3-1). REEDIT first calls INEDTE which is the driver for processing the input and mounting the data file. INEDTE reads and checks the minimal input data supplied for reediting and calls OPEN-RESTART_FILE which mounts and checks the header label of the data file. Subroutine READ_RESTART_RECORD is then called sequentially to find the user-specified data record at which editing is to begin. When the desired record is reached, it moves the data from the restart file allocated memory.

EDITRE is called to direct the retrieval and editing of data from the restart data file. The subroutine calls from EDITRE are done within a loop. Subroutine READ_RESTART_RECORD is called within the loop to position the restart file to the desired data record, It then moves the data record from the restart file to allocated memory. EDIT then performs the printed edit requests. This loop continues until all the data records requested have been edited. Subroutine PRNPLT performs printer plots of the minor edit variables.

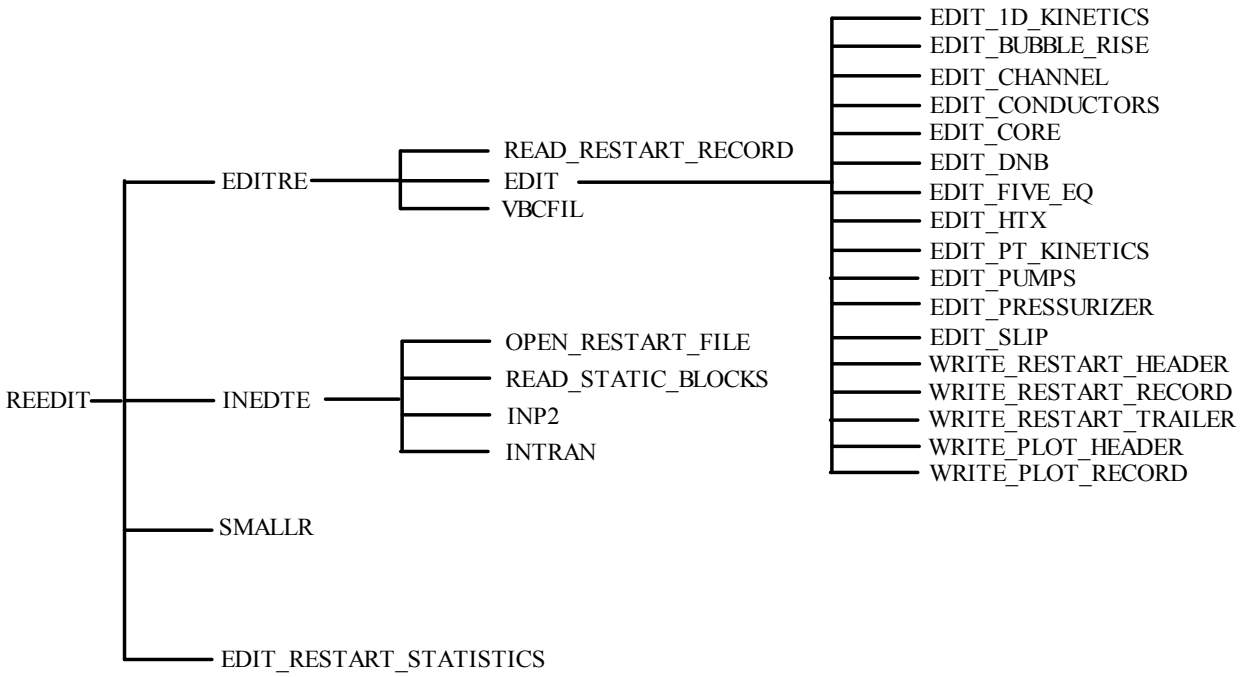


Figure III.3-1. Subroutine Calls from REEDIT

IV

RETRAN INPUT/OUTPUT

The RETRAN-3D computer code uses external files to provide the input model description, supply boundary condition information, to store intermediate results and to archive solution results. Some of these files are program option dependent and are necessary only when a particular program option is used.

The following sections describe all of the files that are used and the particular program option they are used in.

1.0 UNIT DESCRIPTION

The RETRAN-3D computer code only uses standard binary and formatted Fortran reads and writes. A number of temporary and permanent data files are written and read using standard Fortran I/O statements. These data files are associated with logical Fortran unit numbers defined in Table IV.1-1. The Fortran unit number, logical file name, the purpose the unit serves, and subroutine that opens the unit are also given. A logical file name of "default" indicates that the file is a scratch file that will be opened with the default file name for the operating platform.

2.0 DATA FILE DESCRIPTION AND USE

The RETRAN-3D program uses external data files that may consist of program input data, data files written by the program, or data files that provide temporary storage during program execution. This section describes the data files used by RETRAN-3D.

2.1 Restart Data File

A RETRAN-3D restart data file contains the information necessary to continue the problem calculation with the **restrt** option, to provide boundary conditions to a subsequent calculation using either the **retran** or **restrt** options, or to obtain printed edits or printer plots for a problem solution with the **reedit** option. This data file is used by the RETRAN-3D problem modules **restrt, reedit** and the time-dependent volume boundary condition calculation in **retran** and **restrt**.

RETRAN-3D data files are binary and will generally be disk files, but they may also reside on other read/write storage media supported by the platform being used. Restart files are composed of three types of records; a header record which is the first record on all restart data files, data

Table IV.1-1

File Unit Description

FORTRAN Unit No.	Logical File Name	Open Subroutine	Purpose
5	INPUT	**rmain**	Input data for **inp** free form input processing
6	OUTPUT	**rmain**	Standard print output
8	REMARKS	**remark**	Remark log file
3	TAPE3	**trip & trpsum**	Temporary storage for trip action history – **retran** or **restrt**
12	TAPE12	**tapebc**	Time dependent boundary condition data from restart file – **retran** or **restrt**
13	TAPE13	**inrstr**	Restart data file to provide starting conditions – **restrt**
14	TAPE14	**restrt & edit**	Restart data file
17	ERR_LOG	**errlog**	Error file
20	Default	**edit**	Store printer plot data
25	R3D_PLOT	**write_plot_header**	Results for plotting
30	retran.cfg	**set_configuration**	Read RETRAN-3D configuration file parameters
40	TAPE40	**inxsec**	Input 1-D kinetics cross section
60	TAPE60	**edit**	Auxiliary user defined output
68	TAPE68	**geom3d**	3-D kinetics binary cross-section (BXF) file
78	TAPE78	**inchan**	3-D kinetics core data interface (CDI) file

records containing transient problem solution results, and a trailer record written as the terminal record for a RETRAN-3D restart data file. These records are actually logical records since multiple writes may be used to write each logical record. A restart data file is defined as a continuum of solution results from time zero to problem end time so it will include one header record, multiple data records and one trailer label.

2.1.1 Header Record Description

RETRAN-3D restart data file header record consists of problem identification and description data as shown in Table IV.2-1. It also contains the problem dimension specifications included in the **problem_dimension** module and the original problem input data included in the **inp_cards** module. The corresponding block names are **i_dim** and **r_cards** respectively. They are included in the static restart block list discussed in Section II.3 and will be included in the restart record header at a minimum. Other static blocks may also be included but currently are not used for restart.

The horizontal lines in Table IV.2-1 indicate physical record boundaries within the logical header record, which is written by subroutine **write_restart_header,** which resides in the **restart** data module.

The information contained in the header record is used;

- to verify that the desired restart data file has been requested,
- to provide original problem detail for program options that use the restart file, and
- to insure that the archived data is consistent with the requested usage.

Header information is also used to set up pointer associations used to retrieve data from a restart file that used to define boundary conditions, and to provide **restrt** with a physical description of the system being modeled through the original input data.

2.1.2 Data Record Description

Restart file data records contain all of the time-dependent data necessary to continue a problem solution through use of program option **restrt**. They also contain all information required to obtain edits of archived solution results through use of program option **reedit**.

Table IV.2-2 lists the content of each restart data record where each record corresponds to a point in time for the problem solution. The horizontal lines in Table IV.2-1 indicate physical record boundaries within the logical data record, which is written by subroutine **write_restart_record**. It resides in the **restart** data module.

Table IV.2-1

Header Record Description

Word	Data Type[1]	Description of Contents
header_label	C-ckd,len=32	Textual information specifying that the file is a 'RETRAN-3D data tape header label'.
licensed_org	C-ckd,len=32	Licensed Organization – Textual information identifying the licensed organization.
version_header	C-ckd,len=80	Textual information documenting the version of RETRAN-3D used to create the restart file.
problem_title	C-ckd,len=80	The content of the problem title card plus the date the problem was run that created the restart file.
mxi_len	I-ikd	The length of the largest integer data block in the restart file (includes static and dynamic blocks).
mxi2_len	I-ikd	The length of the largest integer 2D data block in the restart file (includes static and dynamic blocks).
mxl_len	L-ikd	The length of the largest logical data block in the restart file (includes static and dynamic blocks).
mxl2_len	L-ikd	The length of the largest logical 2D a block in the restart file (includes static and dynamic blocks).
mxr_len	I-ikd	The length of the largest real data block in the restart file (includes static and dynamic blocks).
mxr2len	I-ikd	The length of the largest real 2D data block in the restart file (includes static and dynamic blocks).
_____	Repeat the following for each static block in the restart block list	
block_name	C-ckd,len=8	The name of the static block written to the header record. Block names begin with r_, i_ or l_ to indicate if they are real, integer or logical data blocks, respectively. Block names beginning with r2_, i2_,or l2 indicate if the blocks are real, integer, or logical 2D blocks, respectively.
len_blk	I-ikd	Length of the block
Appropriate Data Block as Given by the block name (based on 1st character only)		
r_pntr	R-rkd	Real data block
i_pntr	I-ikd	Integer data block
l_pntr	L-lkd	Logical data block
r_2pntr	R-rkd	Real 2D data block
i_2pntr	I-ikd	Integer 2D data block
l_2pntr	L-lkd	Logical 2D data block

[1]C = Character, I = Integer, L = Logical, R = Real – Parameters on right are defined in module kind_specs and indicate the size.

Table IV.2-2

Data Record Description

Word	Data Type[1]	Description of Contents
data_record	C-ckd,len=8	Textual string containing 'data_rec'
record_number	I-ikd	The restart record number.
_____	Repeat the following for each dynamic block in the restart block list	
block_name	C-ckd,len=12	The name of the dynamic block written as part of the restart data record. Block names begin with r_, i_ ,l_, r2_, i2_, or l2_ to indicate if they are real, integer or logical data blocks, respectively.
len_blk	I-ikd	Length of the block
_____		
Appropriate Data Block as given by the block name (based on character preceding "_")		
r_pntr	R-rkd	Real data block
i_pntr	I-ikd	Integer data block
l_pntr	L-lkd	Logical data block
r_2pntr	R-rkd	Real 2D data block
i_2pntr	I-ikd	Integer 2D data block
l_2pntr	L-lkd	Logical 2D data block

[1]C = Character, I = Integer, R = Real – The integer portion indicates the size

The information contained in data records is used;

- to define the complete problem state at the point in time where a restart problem begins,
- to provide boundary condition information for a subsequent problem, or
- to provide solution results for the **reedit** program option.

If the data record structure is changed, RETRAN-3D restart data files generated with previous code versions will not be compatible with the **restrt** or **reedit options** in the modified code version.

2.1.3 Data File Generation

RETRAN-3D restart files are generated by subroutine **edit** for both the **retran** and the **restrt** program options once the transient solution begins. Subroutine **edit** writes the header record at the beginning of new restart files by making a call to subroutine **write_restart_header**. Data records are written at user specified intervals by making a call to subroutine **write_restart_record**. When the problem terminates normally, a trailer label is written to the restart file by making a call to **write_trailer_label**. These calls are made in subroutine EDIT and the three subroutines that perform the writes are included in module restart.

2.1.4 Data File Usage

RETRAN-3D restart data files provide the data interface between the **retran** program and the **restrt** and **reedit** options. Use of a RETRAN-3D restart data file by all program options is facilitated through use of a set of application specific restart data file processing subroutines. A summary of these data processing subroutines and their functions is given in Table IV.2-3. Additional detail on their use is given in Section II.3.1.

The three program options use RETRAN-3D data tapes for various purposes. Consequently, several different Fortran unit numbers are used by the program modules. A summary of the Fortran unit numbers and the corresponding restart data file requirements for the three program options is given in Table IV.2-4.

Table IV.2-3

Data Tape Processing Subroutine Descriptions

Subroutine	Description
restart module	The following subroutines are included in the **restart** module.
open_restart_file	Used to open a file and to ensure that the requested file is a RETRAN-3D restart file and that the first 16 characters of the original and current problem titles match.
get_static_blocks	Used by program module **restrt** to retrieve the original problem dimensions and input data from the header record of a RETRAN-3D restart file.
read_restart_record	Used to read a given data record into memory. Dynamic data block in the restart record are read into the corresponding data block in memory. Also has an option to skip the data blocks rather than reading them.
write_header_record	Used to write the header record to a new restart file.
write_restart_record	Used to write a new restart record to a restart file.
write_trailer_label	Used to write a trailer label on a restart file at the end of a problem solution.
bc_file module	The following subroutines are included in the **bc_file** module.
bc_file_initialization	Used to read the original problem dimension information from the header record of a restart file previously opened using a call to **open_restart_file**. This subroutine maps the pointer arrays for the boundary condition quantities to the appropriate integer, logical or real buffer used to read the data blocks.
get_bc_data	Used to read data record from a restart file and then extract the required boundary condition data.

Table IV.2-4

RETRAN Data Tape FORTRAN Unit Number Cross Reference[1]

Program Module	Unit Number	Description
retran	2	A temporary date file used to store the input data deck in the **inp** free form input internal table format.
	3	A temporary data file used to store the history of trip actuations that are edited at the end of a problem.
	12	Read a previously generated RETRAN-3D restart file to obtain time-dependent boundary conditions for a volume(s) and/or to supply the power history. (READ Only)
	14	Unit used to write a RETRAN-3D restart file when requested by input data. (WRITE Only)
	17	File used to accumulate error messages and related diagnostic information.
	20	Temporary file used to store minor edit data used to generate printer plots.
	40	1-D kinetics cross section data file. (READ Only)
	60	Optional user-specified file containing the list of requested minor edit variables. The file can be binary or formatted according to the user's specifications. (WRITE only)
	68	3-D kinetics binary cross-section data file (BXF). (READ only)
	78	Three-dimensional kinetics ASCII Core Data Interface (CDI) file. (READ only)
restrt	12	Same as for **retran** program module above.
	13	Read a previously generated RETRAN-3D restart file. The restart file contains the information required to restart a RETRAN-3D problem. (READ Only)

Table IV.2-4 (Cont'd)

Program Module	Unit Number	Description
	14	Restart file read on unit 13 is copied to unit 14 up to (but not including) the data record containing the requested restart information. From the restart point on, data records are written as requested through input data. (WRITE Only)
	17	Same as for **retran** program module above.
	40	Same as for **retran** program module above.
	60	Same as for **retran** program module above.
reedit	13	RETRAN-3D restart file containing information for which printed edits and/or printer plots are created. (READ Only)
	20	Temporary file used to store minor edit data that is used to generate printer plots.
	60	Same as for **retran** program module above.

(1) Logical file names are given in Tables IV.1-1 and IV.1-2.

2.2 One-Dimensional Space-Time Kinetics Data File

The one-dimensional space-time kinetics data file must be supplied when NODEL = 2 (on data card 01000Y) is specified in the input model. The file is read from Fortran unit 40 and must be specified using the appropriate job control language. The data file must be in character mode with each record written with a Fortran-formatted **write** or equivalent. Thus, a record contains the information transmitted with each such **write** command. The expected formats and descriptions of the file structure are presented in the following sections.

2.2.1 Multiple Control State Control Rod Model

The multiple control state control rod model used by RETRAN-3D contains four types of information:

- nuclear parameter data for the one-dimensional kinetics equation,
- the control rod fraction data for the control rod model,
- one-dimensional neutron flux and power values corresponding to the nuclear parameter data and, as an option,
- nuclear parameter upper and lower limit values.

The nuclear parameter data records must contain the initial values and coefficients needed to calculate the following variables as a function of the feedback quantities and control rod position:[IV.2-1]

β	=	delayed neutron fraction (dimensionless),
Σ_a	=	absorption cross section (1/cm),
B^2	=	buckling ($1/cm^2$),
D	=	diffusion coefficient,
Σ_s	=	down scatter or removal cross section,
$\kappa\Sigma_f$	=	Kappa*Fission cross section,
$\upsilon\Sigma_f$	=	Nu*Fission cross section, and
V	=	neutron speed.

The control fraction data following the nuclear parameter data records are a one-dimensional representation of the rod distribution used to generate the cross- section set normalized as a fraction between minimum (0.0) and maximum (1.0) control.

The nuclear parameter and control fraction data must be included on the space-time kinetics data file. The third set of data consisting of point values for the group fluxes and one-dimensional averaged data from a multi-dimensional nodal code may be replaced with blank records. The fourth set of data is present on the file only if ILIM = 1 (Word 13 on the space-time kinetics data card 30000Y).

2.2.2 Cross-Section Data Record Structure

Data records in the cross-section file are organized by control state with the first set assumed to be the base or initial case. If present, control states having less control than the base case are included next in order of decreasing control and then any states having more control than the base case are used only in rod withdrawal sequences and may be omitted if rod insertion is the only allowed motion. The first record in the data file specifies the number of control states in each category (having less control and more control than the base case). Also included in the first record are the number of neutronic regions for which cross sections are defined and the number of mesh points at which group fluxes are defined. If the number of mesh points is zero, then a blank record must be included instead for each group flux.

Within each control state, the data are ordered progressively by cross-section type, group number, and region number. The record structure for the cross-section data is given in Table IV.2-5. for the base case. Data Records 4 through 17 are repeated for each energy group for Region 1. For the last, i.e., thermal group, the removal cross-section records are omitted. The sequence of records beginning at Record 2 is then repeated for each neutronic region.

If the order of the coefficients N1, N2, and N3 is zero, the next record containing the coefficients should not be supplied. In this case, the value of the cross section will be set to zero in the code.

If the values for a cross section in a controlled state are identical to those for the base case, the values of N1, N2, and N3 may be set to zero. Data from the corresponding base case cross section will be copied into the controlled state. The coefficients for the controlled state should not be supplied.

2.2.3 Control Fraction Data Record Structure

A control fraction must be supplied for each neutronic region in a control state. The control fraction represents the normalized control rod density as a function of position (region number) from the bottom of the core excluding reflectors. The record must be written with a FORTRAN-formatted WRITE or equivalent using 30E16.7 format. This record is a continuation of the cross-section data file and should immediately follow the coefficient data without an end of file separator.

Table IV.2-5

Cross-Section Data Record Structure
Multiple Control State Model

RECORD 1: Dimensional data for cross-section records.

N1 - number of rod insertion control states
N2 - number of rod withdrawal control states
N3 - 0 (reserved)
N4 - number of neutronic regions in cross-section set
N5 - number of mesh points in neutron flux data (5I8 format)

RECORD 2: Order of independent variables for ß coefficients for Region 1.

N1 - moderator density variable
N2 - fuel temperature variable
N3 - moderator temperature variable (3I8 format)

RECORD 3: ß coefficients for Region 1 (30E16.7 format). Number of coefficients in list = N1*N2*N3.

RECORD 4: Order of independent variables for Σ_a coefficients for Group 1, Region 1. Zeros mean next record missing - assume all coefficients are zero.

N1 - moderator density variable
N2 - fuel temperature variable
N3 - moderator temperature variable (3I8 format)

RECORD 5: Σ_a coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

RECORD 6: Order of variables for B^2 coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3.

RECORD 7: B^2 coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

RECORD 8: Order of variables for D coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3.

RECORD 9: D coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

Table IV.2-5 (Cont'd)

RECORD 10: Order of variables for Σ_s coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3 (Group 1 only).

RECORD 11: Σ_s coefficients for Group 1, Region 1 (30E16.7 format) (Group 1 only). Number of coefficients = N1*N2*N3.

NOTE: For the thermal group, i.e., last group, Records 10 and 11 are omitted.

RECORD 12: Order of variables for $\kappa\Sigma_f$ coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3.

RECORD 13: $\kappa\sum_f$ coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

RECORD 14: Order of variables for $\upsilon\Sigma_f$ coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3.

RECORD 15: $\upsilon\Sigma_f$ coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

RECORD 16: Order of variables for V coefficients for Group 1, Region 1 (3I8 format) N1, N2, and N3.

RECORD 17: V coefficients for Group 1, Region 1 (30E16.7 format). Number of coefficients = N1*N2*N3.

2.2.4 Neutron Flux and Nodal Code Data Record Structure

Neutron fluxes may be supplied with the base case cross-section data to be used as an initial estimate for the static one-dimensional solution in RETRAN-3D. If present, the number of mesh points must match the RETRAN-3D one-dimensional geometry specification. Group 1 flux values are given first for all points, then Group 2 values. Records are written with a FORTRAN-formulated WRITE or equivalent using a 30E16.7 format.

If group fluxes are not present in the file, a single blank record must be included for each group. Blank records should be included in place of fluxes for all controlled states.

Additional records (if present) allows averaged one-dimensional data from the upstream nodal code (such as SIMULATE-E) to be passed on to RETRAN-3D. These data are used for comparison with similar data from RETRAN-3D. There are four additional records (a value for each region), containing averaged axial density, averaged power, GROUP1 and GROUP2 fluxes. Comparisons are currently edited only for the base case. If the nodal code data are not available, four blank records immediately following the fine mesh fluxes should be included in the cross-section data file. In this case no comparisons with nodal data can be made, but the RETRAN-3D averaged data will be printed.

If the fine mesh fluxes are not present in the file, RETRAN assumes that there are no nodal data supplied. A total of six blank records should be inserted to provide the correct spacing for the missing records. The first two blank records are for the missing GROUP1 and GROUP2 fine mesh fluxes discussed above. The remaining four blank records are for the course mesh (nodal) values of density, power, GROUP1, and GROUP2 course mesh fluxes.

2.2.5 Cross-Section Limit Values

Upper and lower limits for each neutronic parameter can be supplied on the cross- section data file. The limits are optional and are requested by setting the variable ILIM = 1 (Word 13 on Card 30000Y of the RETRAN input deck). If this option is used, an upper and lower limit must be supplied for each neutronic parameter for each region and energy group. The cross-section limit data are a continuation of the cross-section data file following the data for all control states. The order in which the limits are supplied is described in Table IV.2-6.

2.3 Three-Dimensional Space-Time Kinetics Data Files

The multidimensional kinetics option requires two auxiliary files to support the model. One is an ASCII file that contains geometries and core layout information used by the upstream physics codes, and the other file contains the cross-section and physics data required by the multidimensional kinetics model. These two files are described in this section.

Table IV.2-6

Cross-Section Limit Data Format
Multiple Control State Model

RECORD N+K: Maximum value of ß for Region K. Minimum value of ß for Region K.

Format: (2E16.7)

RECORD N+K+1: Maximum and minimum values in Region K, Group 1 for Σ_a, B^2, D, Σ_s, $\kappa\sum_f$, $\upsilon\Sigma_f$, V.

Format: (14E16.7)

RECORD N+K+2: Maximum and minimum values in Region K, Group 2 for Σ_a, B^2, D, $\kappa\Sigma_f$, $\upsilon\Sigma_f$, V.

Format: (12E16.7)

.
.
.

where N represents the total number of records used to describe the nuclear parameters.

Repeat for K=1 to number of regions.

2.3.1 The CDI File

The ability to design and implement multidimensional kinetics models is highly dependent upon the coupling of complex computer codes that process neutronics cross sections and core geometric data. The amount of information is extensive and a manual or non-automated method of processing the input lends itself to transcription errors or modeling inconsistencies. In RETRAN-3D, the channel model provides tighter coupling with upstream physics codes to provide easier, less redundant input methods and reduce sources for input error. The channel model is described in Section 28.0 of the User's Manual. The data file to support the channel model is described here.

A key feature of the channel model is an ASCII data file that links the neutronic and thermal-hydraulic specifications from the CORETRAN[N.2-1] model to the channel model input routines in RETRAN-3D. It is referred to as the CORETRAN Data Interface (CDI) file and is created by CORETRAN or an alternate method if CORETRAN is not the upstream core simulator code. Figure IV.2-1 shows the data flow for the channel model.

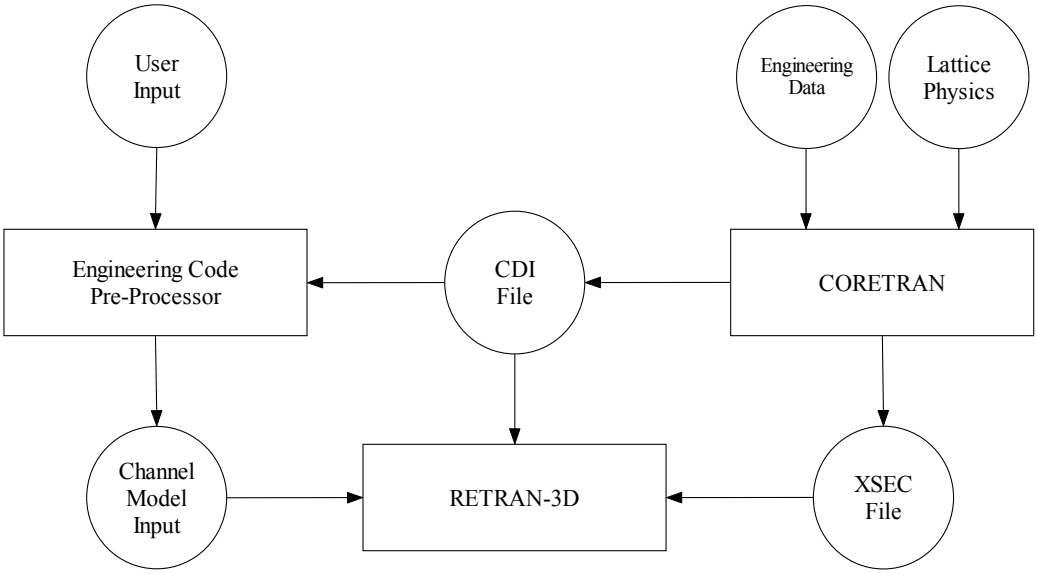


Figure IV.2-1. RETRAN-3D Channel Model Data Flow

The CDI file (FORTRAN Unit 78) augments the RETRAN-3D channel model. It is an ASCII file containing character, integer, and floating point data that characterize the core model in terms of geometry, boundary conditions, and thermal-hydraulic definition. The two aspects of the CDI file are the physics data and the thermal-hydraulic data. The CDI file contains all of the assembly layout and geometric information required for RETRAN-3D to internally generate volume, junction, and conductor input for the core region in a specified layout. The CDI file contains heated lengths, flow areas, wetted perimeters, conductor volumes, and loss coefficients. By making assumptions about channel connections to the upper and lower plenums, the axial number of axial nodes, and user-defined core-wide options such enthalpy transport, two-phase flow model as examples, the new routine GEOM3D generates all necessary volume and junction data internal to the code.

Table IV.2-7 lists the structure of the CDI file. It is comprised of a number of distinct blocks that define the multidimensional kinetics and thermal-hydraulics models, flowed by records containing the block related data. The block header (bold text under block heading) is a text field that precedes the information that comprises the block.

2.3.2 The Three-Dimensional Kinetics Cross-Section File

The primary source for RETRAN-3D cross sections is the CORETRAN code.[IV.2-1] Recent versions of CORETRAN have implemented a new cross-section formalism in order to address reaction types and spectrum issues of interest to fuel management groups. In general, the lattice information (from CPM-3[IV.2-3] or CASMO[IV.2-4] calculations) is processed by CORETRAN and at given state points (i.e., point in the burn cycle) a RETRAN-3D cross-section file is written

Table IV.2-7

CDI File Structure and Content

Block	Format	Content		
TITLE	16A4	Title Field		
DIMN	A4	Dimension Block Header		
	10(1X,I5)	W1-I	ID	Number of columns in the assembly mesh
		W2-I	JD	Number of rows in the assembly mesh
		W3-I	KD	Number of planes in the assembly mesh
		W4-I	ISYM	Symmetry flag for core expansion 0 = solved as given 1 = quarter core to half core with reflection 2 = quarter core to half core with rotation 3 = half core to full core with reflection 4 = half core to full core with rotation 5 = quarter core to full core with reflection 6 = quarter core to full core with rotation
		W5-I	NOFT	Number of assembly types
		W6-I	NCOMP	Number of fuel composition types

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W7-I	ICORE	Core type 1 = PWR with top inserted RCC rods 2 = BWR with bottom inserted cruciform rods 3 = PWR with top inserted cruciform rods 4 = BWR with bottom inserted homogenized rods
		W8-I	NDNEUT	Number of delayed neutron groups
		W9-I	NGPS	Number of control rod groups
		W10-I	IXBCL	Left X-direction boundary condition 0 = zero flux at cell edge 1 = zero current at cell edge 2 = no return flux at cell edge
		W11-I	IXBCR	Right X-direction boundary condition
		W12-I	IYBCL	Left Y-direction boundary condition
		W13-I	IYBCR	Right Y-direction boundary condition
		W14-I	IZBCL	Left Z-direction boundary condition
		W15-I	IZBCR	Right Z-direction boundary condition
		W16-I	IXSYM	Centerline (X = 0.0) assembly specification 0, if the assembly at X = 0.0 is a full assembly 1, if the assembly at X = 0.0 is a half assembly

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W17-I	IYSYM	Centerline (Y = 0.0) assembly specification 0, if the assembly at Y = 0.0 is a full assembly 1, if the assembly at Y = 0.0 is a half assembly
PARM	A4	Parameter Block Header		
	10(1X,I5)	W1-I	NLEAKS	Leakage approximation to use during eigenvalue calculation 1 = truncated leakage approximation 2 = BALUNS implicit leakage treatment 3 = Gauss-Seidel explicit treatment (default)
		W2-I	NLEAKT	Leakage approximation to use during transient calculation 1 = truncated leakage approximation 2 = implicit before flux iterations 3 = explicit before flux iterations (default) 4 = implicit before flux iterations plus one update after NCNCRT-iterations 5 = explicit before flux iterations plus one update after NCNCRT-iterations 6 = implicit before flux iterations and after every other inner iteration pass 7 = explicit before flux iterations and after every other inner iteration pass
		W3-I	NOTERS	Maximum outer iterations per reactor iteration (5)

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W4-I	MAXOUT	Maximum fission source outer iterations for eigenvalue calculation (200)
		W5-I	MAXRIT	Maximum nonlinear iterations (50)
		W6-I	NCNCRS	Number of inner iterations per outer in eigenvalue calculation (4)
		W7-I	NCNCRT	Number of inner iterations per outer in transient calculation (2)
		W8-I	N1CHEB	First outer iteration to apply Chebyshev acceleration (5)
		W9-I	N2CHEB	Initial order of Chebyshev polynomial (3)
		W10-I	N3CHEB	Maximum order of Chebyshev polynomial (10)
		W11-I	NED1	Input editing (No/Yes ~ 0/1) (1)
		W12-I	NED2	Rod position editing (No/Every edit internal/Every time step ~ 0/1/2) (1)
		W13-I	NED3	Average radial and axial power distribution editing (No/Every edit interval/Every time step ~ 0/1/2) (1)
		W14-I	NED4	Flux distribution editing (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W15-I	NED5	Feedback variable editing (No/Every edit interval/Every time step ~ 0/1/2) (0)

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W16-I	NED6	All T/H variable editing (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W17-I	NED7	Precursor concentration or XISP number of density editing (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W18-I	NED8	Nodal cross section and K-infinitives (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W19-I	NED9	Nodal relative power distribution (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W20-I	NED11	3-D nodal ADFs (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W21-I	NED14	Auxiliary editing (No/Every edit interval/Every time step ~ 0/1/2) (0)
		W22-I	ISPECK	Truncated leakage treatment for nodes in radial zero flux boundary corners. This helps to achieve a positive flux at reactor edge. $<$ 0, skip $\geq$ 0, activate (default)
		W23-I	NFEDEX	Feedback variable extrapolation option 0 = no extrapolation (default) 1 = linear 2 = geometry 3 = quadratic

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W24-I	NADJNT	Option to calculate the adjoint flux (No/Yes ~ 0/1) (0)
		W25-I	IXTRP	Option to allow extrapolation on moderator density when calculating nodal cross section (No/Yes ~ 0/1)
PAR1	A4	Parameter Block 1 Header		
	10(1X,1P,E12.5)	W1-R	EIGUES	Initial guess for eigenvalues (1.0)
		W2-R	POWRIN	Total problem power (MWth) at initial condition (1.0)
		W3-R	THETAF	Flux equation time-differencing parameter (1.0)
		W4-R	THETAP	Precursor equation time-differencing parameter (0.5)
		W5-R	CONEIG	Eigenvalue convergence criterion based on consecutive change (1.0E-5)
		W6-R	CONDEG	Eigenvalue derivative test criterion (1.0E-5)
		W7-R	CONSRS	Pointwise convergence criterion for S.S. fission source distribution (1.0E-3)
		W8-R	CONSRT	Pointwise convergence criterion for transient neutronics calculation (1.0E-3)
		W9-R	CONPOW	Pointwise convergence criterion for reactor iteration for the thermal-hydraulic calculations. CONPOW > CONSRT is required. (0.01)

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W10-R	ACCSIG	Input value to use for sigma-bar in Chebyshev acceleration. If not supplied the code will estimate adaptively as often as necessary.
		W11~W16		Nonstandard value for the "al-ratio" at the ends of axes. The "al-ratio" is the ration of the transverse leakage in the node adjacent to the outside boundary over the transverse leakage in the last interior node.
		W11-R	ALRXL	At the left end of X-axis (+1.0 for IXBCL = 1; -1.0 for IXBCL = 0)
		W12-R	ALRXR	At the right end of X-axis (+1.0 for IXBCR = 1; -1.0 for IXBCR = 0)
		W13-R	ALRYL	At the left end of Y-axis (+1.0 for IYBCL = 1; -1.0 for IYBCL = 0)
		W14-R	ALRYR	At the right end of Y-axis (+1.0 for IYBCR = 1; -1.0 for IYBCR = 0)
		W15-R	ALRZL	At the left end of Z-axis (+1.0 for IZBCL = 1; -1.0 for IZBCL = 0)
		W16-R	ALRZR	At the right end of Z-axis (+1.0 for IZBCR = 1; -1.0 for IZBCR = 0)
		W17-R	CONDIF	Convergence criterion in watts/cc for the reactor iteration scheme. An input value of 0.0 will turn off this option. (0.5)

Table IV.2-7 (Cont'd)

Block	Format	Content		
PAR2	A4	Parameter Block 2 Header		
	10(1X,I5)	W1-I	NCOL	Number of nodes in X-direction in neutronics mesh
		W2-I	NROW	Number of nodes in Y-direction in neutronics mesh
		W3-I	NPLN	Number of nodes in Z-direction in neutronics mesh
		W4-I	IDIAG	Diagonal symmetry indicator for the X-Y plane (0 = not symmetric; 1 = symmetric)
		W5-I	IFADF	Flag for the existence of assembly discontinuity factors (0 = no; 1 = yes)
		W6-I	IFROD	Flag for the existence of control rods (0 = no; 1 = yes)
LIMT	A4	Limit Block Header		
	10(1X,I5)	W1-I	NXLIM1	Minimum power producing node in X-direction ($\geq$ 1)
		W2-I	NXLIM2	Maximum power producing node in X-direction ($\leq$ NCOL)
		W3-I	NYLIM1	Minimum power producing node in Y-direction ($\geq$ 1)
		W4-I	NYLIM2	Maximum power producing node in Y-direction ($\leq$ NROW)
		W5-I	NZLIM1	Minimum power producing node in Z-direction ($\geq$ 1)

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W6-I	NZLIM2	Maximum power producing node in Z-direction (≤ NPLN)
		W7-I	KQCORE	Denotes quarter, half, or full core problem. 0 = quarter core 1 = half core, full X-dimension 2 = half core, full Y-dimension 3 = full core
RODL	A4	Rod Layout Block Header		
	2(1X,I5)	W1-I	NGPS	Number of control rod groups
		W2-I	LL	Length of control rod group data in this block (including all contents)
	10(1X,I5)	W1-I	NRODGP(1)	Number of control rods in group 1
		.	.	.
		.	.	.
		.	.	.
		WNGPS-I	NRODGP(NGPS)	Number of control rods in group NUPS (cm)
	10(1X,1P,E12.5)	W1-R	TIPGP(1)	Tip position of the control rods in group 1 (cm)
		.	.	.
		.	.	.
		.	.	.
		W(NGPS)-R	TIPGP(NGPS)	Tip position or control rods in group NUPS (cm)
	Tip position is measured from bottom of neutronics mesh.			
	10(1X,I5)	W1-I	X(1,k)	X-coordinate of the first control rod in group k
		W2-I	Y(1,k)	Y-coordinate of the first control rod in group k

Table IV.2-7 (Cont'd)

Block	Format	Content		
		W3-I	X(2,k)	X-coordinate of the second control rod in group k
		W4-I	Y(2,k)	Y-coordinate of the second control rod in group k
		.	.	.
		.	.	.
		.	.	.
		W(2*NRODGP(k))-I		Y-coordinate of the last control rod in group k (k = 1 to NGPS)
XMSH	A4	X Mesh Block Header		
	10(1X,1P,E12.5)	W1-R	XMSH(1)	Size of the first neutronics mesh in X-direction (cm)
		.	.	.
		.	.	.
		.	.	.
		W(NCOL)-R	XMSH(NCOL)	Size of the last neutronics mesh in X-direction (cm)
YMSH	A4	Y Mesh Block Header		
	10(1X,1P,E12.5)	W1-R	YMSH(1)	Size of the first neutronics mesh in Y-direction (cm)
		.	.	.
		.	.	.
		.	.	.
		W(NROW)-R	YMSH(NROW)	Size of the last neutronics mesh in Y-direction (cm)
ZMSH	A4	Z Mesh Block Header		
	10(1X,1P,E12.5)	W1-R	ZMSH(1)	Size of the first neutronics mesh in Z-direction (cm)
		.	.	.
		.	.	.
		.	.	.
		W(NPLN)-R	ZMSH(NPLN)	Size of the last neutronics mesh in Z-direction (cm)

Table IV.2-7 (Cont'd)

Block	Format	Content		
CTCM	A4	CORETRAN Channel Layout Block Header		
	50(1X,I5)	W1-I . . . W(ID)-I	CTCM(1,j) . . . CTCM(ID, j)	CORETRAN channel layout. Each powered assembly has a unique channel number. Non-powered assemblies are defined so that their channel number is 0. ID is the number of columns.
		Repeated for the JD rows (j = 1,JD)		

The following are thermal-hydraulic geometry data groups

Problem Dimension Data The data are written as one item per record.

(/////,2X,20A4,/	TH Problem	Title
/,2X,I5,	W1-I NCHANV	Number of Active (heated) Channels
/,2X,I5,	W1-I NBYP	Number of Bypass Channels (currently limited to 0 or 1)
/,2X,I5,	W1-I NZHEAT	Number of Heated Axial Planes
/,2X,I5,	W1-I NNODEI	Number of Unheated Inlet Planes
/,2X,I5,	W1-I NNODEO	Number of Unheated Outlet Planes
/,2X,I5,	W1-I NGEOMV	Number of Volume/Conductor Geometries
/,2X,I5,	W1-I NGEOMJ	Number of Junction Geometries
/,2X,I5,	W1-I NFUELV	Number of Fuel Geometries
/,2X,I5,	W1-I NMATV	Number of Material Properties (reserved for future use)
/,2X,I5,	W1-I NFIBWR	Number of FIBWR Leakage Definitions
/,2X,I5,	W1-I NWTUBE	Number of Water Tube Geometries
/,2X,I5,	W1-I NSETLS	Number of Grid Loss Coefficient Sets
/,2X,F11.6,//)	W1-R DXHEAT	Total heated length of the core (ft)

Table IV.2-7 (Cont'd)

Block	Format	Content		
Axial Node Lengths The data are written in groups of six per record. Number of nodes is equal to total number of axial planes (NP = NZHEAT + NNODEI + NNODEO). The first node is written in I7 format, the other nodes are I5.				
	(I7	W1-I	NN(1)	Node Number
	F10.6	W2-R	FLCH(1)	Node 1 Heated Length (ft)
	I5	W3-I	NN(2)	Node Number
	F10.6	W4-R	FLCH(2)	Node 2 Heated Length (ft)
		.		
		.		
	I5	W(2NP-1)-I	NN(NP)	Node Number
	F10.6)	W(2NP)-R	FLCH(NP)	Node NP Heated Length (ft)
	(///)	(Skip)		
Channel Summary Data Each record represents a thermal-hydraulic channel. Total number of records is equal to total number of channels represented (NC = NCHANV + NBYP). If a bypass channel is included, it is assumed to be the last channel. Each index defines the specific geometry block to use for each channel.				
	(6(4X,I3),3X,3E15.6)	W1-I	NCHN	Channel Number
		W2-I	IDVOLG	Volume/Conductor Geometry Index
		W3-I	IDJUNG	Junction Geometry Index
		W4-I	IDFUEL	Fuel Geometry Index
		W5-I	IDWATT	Water Tube Geometry Index
		W6-I	IDLLCK	FIBWR Lateral Leakage Path Definition Index
		W7-R	WJUN	Channel Flow, channel inlet flow normalized to total core flow (used for steady state initial guess)
		W8-R	WLCK	Bundle-to-Bypass Leakage Flow, normalized to channel inlet flow (used for steady state initial guess)
		W9-R	WWAT	Water Tube Flow, normalized to channel inlet flow (used for steady state initial guess)
	.			
	.			
	.			
	Repeat this sequence for NC channels of data.			

Table IV.2-7 (Cont'd)

Block	Format	Content	
Volume/Conductor Geometry This data represents the volume and conductor geometry. There is one set for each volume/conductor geometry type up to NGEOMV sets as specified in the problem dimension data. Each set contains data for every axial plane from 1 to NP (NP = NNODEV + NNODEI + NNODEO).			
	(/,26X,I2,////)	W1-I	INDEXV	Volume Geometry Index
	(I7,5E15.6)	W1-I	LEVEL	Level Number
		W2-R	FAREA	Flow Area (ft2)
		W3-R	WETPE	Wetted Perimeter (ft)
		W4-R	HETPE	Heated Perimeter (ft)
		W5-R	CDVOL	Conductor Volume (ft3)
		W6-R	WROUGH	Surface Roughness
	.			
	.			
	Repeat for NP axial levels			
This group is repeated for NGEOMV sets.				
Junction Geometry This data represents the junction (or cell boundary) geometry. There is one set for each junction geometry type up to NGEOMJ sets as specified in the problem dimension data. Each set contains data for every axial junction level from 1 to NJ (NJ = NNODEV + NNODEI + NNODEO + 1).				
	(/,28X,I2,////)	W1-I	INDEXJ	Junction Geometry Index
	(I7,4E15.6)	W1-I	LEVEL	Level Number
		W2-R	FAREA	Flow Area (ft2)
		W3-R	WETPE	Wetted Perimeter (ft)
		W4-R	FWLOSS	Forward Loss Coefficient
		W5-R	BWLOSS	Reverse Loss Coefficient (0.0 defaults to forward loss coefficient) (Negative forward coefficients are used to activate the grid loss model. INT (ABS (FWLOSS)) defines the grid loss coefficient set to use.)
	.			
	.			
	Repeat for NJ axial levels			
This group is repeated for NGEOMJ sets.				

Table IV.2-7 (Cont'd)

Block	Format	Content

Fuel Pin Geometry This data provides the geometric data for NFUELV geometries.

(////,2I7,2E15.6,///)	W1-I	Fuel Geometry Number
	W2-I	Number of Material Regions. Typically three regions are used to represent fuel, gap, and cladding.
	W3-R	Radius to Left Surface (0.0 for solid fuel)
	W4-R	Gap Conductance (reserved for future use)
		Multiple region descriptions can be used to define the fuel or cladding regions. This is useful if different noding or material properties are needed.

The following data are used to provide geometric and power information for the material regions (i.e., fuel, gap, and cladding) for the RETRAN-3D conduction solution.

(3(4X,I3),2E15.6)	W1-I	Gap Indicator (0 - no, 1- yes)
	W2-I	Material Property table index (material properties are defined in the base RETRAN-3D input)
	W3-I	Number of regions for conduction solution (typically 6-fuel, 1-gap, and 2-cladding)
	W4-R	Region Width (ft)
	W5-R	Region Power fraction, where 1.0 indicates 100% of the pin power is generated in this region. If multiple regions are defined for the fuel region, the power frictions must sum to 1.0.

.
.
Repeat until all regions are described.

This group is repeated for NFUELV sets.

FIBWR Leakage Data This data set is required if FIBWR leakage paths are being modeled (NFIBWR > 0 in problem dimensions data). There are four sub-groups of data.

FIBWR Dimension Data. FIBWR dimensions and constants.

(//,2X,I5,	W1-I	NPBND	Number of Lateral Bundle-to-Bypass Leakage Paths (maximum of 9)
/,2X,I5,	W1-I	NPCOM	Number of Leakage Paths across the Core Support Plate
/,2X,F11.5,	W1-R	FCROD	Control Rod Drive Coolant Flow Rate (lbm/hr)
/,2X,F11.5)	W1-R	RHOREF	Reference Coolant Density (lbm/ft3)
(///)	(Skip)		

Table IV.2-7 (Cont'd)

Block	Format	Content		
	FIBWR Coefficients for Bundle-to-Bypass Paths. One set for each leakage path			
	(I7,1P,6E15.6)	W1-I	IPATH	Path Number
		W2-R	C1B	Coefficient C1
		W3-R	C2B	Coefficient C2
		W4-R	C3B	Coefficient C3
		W5-R	C4B	Coefficient C4
		W6-R	BPPER	Multiplication Factor (number paths of this type)
		W7-R	BELEV	Path Elevation (referenced to the bottom of channel)
	Repeat until NPBND sets are defined.			
	(//)	(Skip)		
	FIBWR Coefficients for Core Support Plate Paths. One set for each leakage path			
	(I7,1P,5E15.6)	W1-I	IPATH	Path Number
		W2-R	C1C	Coefficient C1
		W3-R	C2C	Coefficient C2
		W4-R	C3C	Coefficient C3
		W5-R	C4C	Coefficient C4
		W6-R	CPPER	Multiplication Factor (number of paths of this type)
	Repeat until NPCOM sets are defined.			
	(///)	(Skip)		
	Lateral Leakage Path Channel Assignment. This table defines the lateral (bundle-to-bypass) leakage paths that are used for a channel type. Each record defines a definition that can be referenced from the channel summary data block			
	(I8,5X,I3,4X,9I5)	W1-I	Index Number (referenced by IDLLCK in channel summary data)	
		W2-I	Number of Lateral Leakage Path for this definition	
		W3-I	Lateral Leakage Path 1 (0-do not use, 1-use)	
		W4-I	Lateral Leakage Path 2 (0-do not use, 1-use)	
		.	.	
		.	.	
		.	.	
		W11-I	Lateral Leakage Path 9 (0-do not use, 1-use)	

Table IV.2-7 (Cont'd)

Block	Format		Content
Water Tube Geometry Data This data defines the volume and junction geometry for each water tube type. There is data representing the water tube channel, the inlet junction, and the outlet junction. Each set is repeated for the Number of Water Tube Geometries, NWTUBE, specified in the problem dimensions data.			
	(/,28X,I3,////)	W1-I	IWAT	Water Tube Geometry Type
	(2E15.6,	W1-R	WTAREA	Water Tube Flow Area (ft2)
		W2-R	WTWTPE	Water Tube Wetted Perimeter (ft)
	/,4E15.6,	W1-R	FAREAI	Inlet Flow Area (ft2)
		W2-R	WETPEI	Inlet Wetted Perimeter (ft)
		W3-R	FLOSSI	Inlet Loss Coefficient
		W4-R	ELEVNI	Inlet Elevation (ft) (referenced to bottom of channel)
	/,4E15.6)	W1-R	FAREAO	Outlet Flow Area (ft2)
		W2-R	WETPEO	Outlet Wetted Perimeter (ft)
		W3-R	FLOSSO	Outlet Loss Coefficient
		W4-R	ELEVNO	Outlet Elevation (ft) (referenced to bottom of channel)
This group is repeated for NWTUBE geometry types.				
	(//)			Skip
Grid Loss Model Data The data defines the coefficients for the grid loss model. NSETLS sets of coefficients must be provided in the CDI file.				
	(4X,I3,	W1-I	LSET	Coefficient set number |LSET| > 0, Use Eq. IV.31-1 in Volume 3 < 0, Use Eq. IV.31-2 in Volume 3
	1P,4E15.6)	W2-R	COEFA	Coefficient A
		W3-R	COEFB	Coefficient B
		W4-R	COEFC	Coefficient C
		W5-R	COEFD	Coefficient D

that contains instantaneous cross dependancies. That is, the cross-section dependancies for that are instantaneous. No intermediate processing codes are needed.

The RETRAN-3D cross-section file (FORTRAN Unit 68) is written at the end of a CORTRAN "burn" calculation. The RETRAN-3D model is identical in format and in functionality with the one used by CORETRAN, but the historical dependancies have been removed. That is, the RETRAN-3D model only contains the "instantaneous" dependancies, such as fuel temperature, moderator density, control fraction, or boron concentration. This reduces the size of the file as well as the cross-section calculation time for the transient analysis cases.

The cross-section functionality that forms the basis for the CORETRAN model is given by the following expression:

$$\sum = \sum (EXP, TFH, DMH, CTH, BPH, ..., TF, DM, CT, B, ...) \quad \text{(IV.2-1)}$$

The group given by W1 through W5 is repeated for NSETLS coefficient sets.

where the independent variables can be either historic or instantaneous as defined in Table IV.2-8. As a way of accounting for the contribution of each individual physical effect, the cross-section functionality is expressed by the following general sum of components:

$$\begin{aligned}\sum &= \sum_{base} (EXP, DMH) + \sum a_i \, \Delta\sum_i (EXP, TFH, DMH, CTH, BPH, ..., TF, DM, CT, B, ...) \\ &= \sum_{base} (EXP, DMH) + a1 \, \Delta\sum (EXP, DMH, CTH) + a2 \, \Delta\sum (EXP, DMH, BPH) \\ &\quad + a_j \, \Delta\sum (EXP, DMH, TF) + a_l \, \Delta\sum (EXP, DMH, DM) \\ &\quad + a_m \, \Delta\sum (EXP, DMH, CT) + a_n \, \Delta\sum (EXP, DMH, B) + ... \quad \text{(IV.2-2)}\end{aligned}$$

where a_i , $\Delta\sum_i$ represent the several models involved: control rod history contribution, burnable poison history contribution, and Doppler contribution. The advantage of this cross-section treatment is that it can be easily extended to include other phenomenon that users may request to model such as the interaction between the Doppler and moderator density effects.

During a burn calculation for a given depletion point CORETRAN will prepare a table of the contributions of each effect as shown in the table below. Each cross-section set (**S** or **DS**) contains information about: (a) two-group cross sections , (b) kinetics parameters, (c) assembly discontinuity factors (ADF), and (d) lattice pin power distributions.

The cross-section file for RETRAN-3D includes the core dimensions; core fuel loading information; nodal atom densities of Xenon, Iodine, Samarium, and Promethium; and a three-dimensional nodal cross-section database. The fuel loading and depletion data must be obtained from a multidimensional core simulation code such as CORETRAN[IV.2-1] or SIMULATE-3,[IV.2-5] and the nodal cross-section database must be generated based on the cross-section data from lattice analysis codes such as CPM-3[IV.2-3] or CASMO-4.[IV.2-4] It contains two energy groups for diffusion theory constants and kinetics data.

Table IV.2-8

Cross-Section Model Independent Variables

Type	Variable	Description
Historic	EXP	Exposure
	TFH	Fuel temperature history
	DMH	Moderator density history
	CTH	Control rod history
	BPH	Burnable poison history
Instantaneous	TF	Instantaneous fuel temperature
	DM	Instantaneous moderator density
	CT	Actual control state
	B	Soluble boron concentration

The cross-section file used by the multidimensional kinetics option is binary. Example FORTRAN statements that show how the file is written follow.

```
WRITE(21) BXFTTL
WRITE(21) NFUEL, NDEPL, NBRNCH, NXSEC, NINDP, NREC, NSTICK,
     &          NCOL, NROW, NPLN, LAXMX, IDIM10, IDIM11
WRITE(21) ((PROPR(I,J),I=1,10),J=1,NFUEL)
WRITE(21) ((IPROPR(I,J),I=1,10),J=1,NFUEL)
WRITE(21) ((DIMNS(I,J),I=1,NINDP),J=1,NFUEL)
WRITE(21) (((COORD(I,J,K),I=1,NDEPL),J=1,NINDP),K=1,NFUEL)
WRITE(21) (((ICOMP(1,I,J,K),I=1,NCOL),J=1,NROW),K=1,NPLN)
WRITE(21) ((IAMAP(1,I,J),I=1,NCOL),J=1,NROW)
WRITE(21) ((IQMAP(1,I,J),I=1,NCOL),J=1,NROW)
WRITE(21) ((IASMAP(1,I,J),I=1,NCOL),J=1,NROW)
WRITE(21) (AXMX(L),L=1,LAXMX)
WRITE(21) ((((TABLE(I,J,K,L),I=1,NXSEC),J=1,NREC),K=1,NSTICK),L=1,NPLN)
IF(IDIM10 .EQ. 2) THEN
   WRITE(21) ((IASMAP(1,I,J),I=1,NCOL),J=1,NROW)
   WRITE(21) ((IASMAP(1,I,J),I=1,NCOL),J=1,NROW)
ENDIF
IF(NINDP .LT. 0) THEN
   DO L = 1, NPLN
    DO K = 1, NSTICK
     DO J = 1, NROW
         WRITE(21)(TABLE(I,J,K,L),I=1,NXSEC)
     ENDDO
    ENDDO
   ENDDO
ELSE
   WRITE(21) ((((TABLE(I,J,K,L),I=1,NXSEC),J=1,NREC),K=1,NSTICK),L=1,NPLN)
ENDIF
WRITE(21) ((((AXISP(L,I,J,K),L=1,4),I=1,NCOL),J=1,NROW),K=1,NPLN)
```

As shown above, the cross-section file is written as a binary file. Subroutine DRIVE1 reads the title and dimensions and reserves the required memory space in FTB. Subroutine GETXSEC is used to read the rest of the file. Table IV.2-9 describes the content of the file.

Table IV.2-9

Multidimensional Kinetics Cross-Section File Structure

Title

BXFTTL Title and CORETRAN version and creation date (128 Bytes).

Variables

NFUEL	Number of fuel composition types.
NDEPL	Number of depletion steps.
NBRNCH	Number of branch points for an independent variable.
NXSEC	Number of variables in a set of nodal cross-section data.
NINDP	Number of independent variables.
NREC	Total number of branch records for a composition type.
NSTICK	Number of distinct assembly types.
NCOL	Number of columns in problem geometry.
NROW	Number of rows in problem geometry.
NPLN	Number of axial planes in problem geometry.
LAXMX	Length of the AXMX array
IDIM10	Flag for assembly radial nodalization (1:1 node/assembly; 2:2x2 nodes/assembly)
IDIM11	Indicator of quadrant dependency for 2x2 nodes/assembly

Note: NINDP < 0 is used as a flag to indicate the new BXF format: |NINDP| is used in RETRAN-3D. If NINDP < 0 then the table array is written as multiple records as shown on Page IV-38.

Arrays

PROPR(I,J) FUEL LOADING INFORMATION

I=1	FLOAD	Initial Fuel Loading (KgU/cm)
I=2	PRESS	System Pressure (psia)
I=3	DMREF	Reference Moderator Density or Void Fraction
I=4	TFREF	Reference Fuel Temperature
I=5	BORREF	Reference Soluble Boron Concentration
I=6	PBPREF	Reference Burnable Poison
I=7	TCOLD	Cold Condition
I=8		Reserved for future use
I=9		Reserved for future use
I=10		Reserved for future use

J=1 to NFUEL

Note: Only I=1, 3, 4, and 5 are used by the current RETRAN-3D cross-section model.

Table IV.2-9 (Cont'd)

FLOAD = -1000.0 is used as a flag to indicate data converted from an ARROTTA style cross-section file (TAPE67).

IPROPR(I,J) FUEL LOADING INFORMATION

I=1	NFTYP	Fuel Composition Type
I=2	LTYPE	0 if BWR, 1 if PWR Fuel
I=3	IREF	0 if Fuel, 1 if Reflector
I=4	NROWS	Number of rows of fuel pins in lattice geometry
I=5	ISYMM	1 if all the pins are represented, 2 if half (lower left half), 4 if quarter (SE quadrant), 8 if eighth (lower left diagonal of SE quadrant)
I=6	NTOTAL	Total set size for lattice data file
I=7	NREV	Not used
I=8	IQUAD	Quadrant dependencies for 2x2 FA/node
I=9	NXP1	Not used
I=10		Reserved for future use

J=1 to NFUEL

Note: 7, 9, and 10 are not used by the current RETRAN-3D cross-section model.

DIMNS(I,J) Number of branches in the table for the i-th independent variable and the j-th composition type.
I=1 to NINDP
J=1 to NFUEL

COORD(I,J,K) The i-th branch point value of the j-th independent variable for the k-th composition type.
I=1 to NBRANCH
J=1 to NINDP
K=1 to NFUEL

ICOMP(I,J,K) Integer value containing the composition type correspondence for each neutronics node.
I=1 to NCOL
J=1 to NROW
K=1 to NPLN

IAMAP(I,J) Assembly Discontinuity Factor Orientation.
I=1 to NCOL
J=1 to NROW

Table IV.2-9 (Cont'd)

IQMAP(I,J)	Index pointing to the location (i,j) in the 2-D assembly map. (The maximum magnitude of IDQMAP(I,J) is NSTICK). I=1 to NCOL J=1 to NROW
IASMAP(I,J)	Integer value containing the assembly type. Here the assemblies are distinguished based on the axial fuel compositions only. I=1 to NCOL J=1 to NROW
AXMX(LAXMX)	Array containing axial mixing information. AXMX consists of NOFT blocks (NOFT: Number of assembly types - according to IASMAP). Entries in each block: 1 Assenbly type 2*K The absolute value of which is the composition type number for the K-th axial node >0, Mixing with the node above <0, Mixing with the node below 2*K+1 Mixing factor for the K-th axial node
ISMAP1(I,J)	Quadrant Position if 2x2 nodes/assembly (IDIM10=2). I=1 to NCOL J=1 to NROW
ISMAP2(I,J)	Quadrant ADF Direction Position if 2x2 nodes/assembly (IDIM10=2). I=1 to NCOL J=1 to NROW
TABLE(I,J,K,L)	Nodal cross-section and kinetics data in the table. I=1 to NXSEC J=1 to NREC K=1 to NSTICK L=1 to NPLN
NXSEC	Number of variables in a set of nodal cross-section data.
AXISP(L)	Array of nodal XISP (Xenon, Iodine, Samarium, Promethium) concentrations. L=1 to LNXISP
LNXISP	Length of AXISP array (4 x NCOL x NROW x NPLN).

Currently 53 different data types (or variables) are included in the TABLE array (the I index). Table IV.2-10 provides a description of these data types.

2.4 Optional Output Data Files

Users have the option of creating several output data files. They include the auxiliary data file and the VIPRE-01 boundary condition (VBC) file. They are discussed separately below.

2.4.1 Auxiliary Data File

The RETRAN, RESTRT, and REEDIT program modules provide users with the option to write the minor edit parameters to an auxiliary file. Each record in the auxiliary file will consist of the parameters requested as minor edits (time is not automatically included and must be requested) and are written in the order requested. The file will contain only the records written during the current job, so files generated during restart problems will begin at the restart time.

The auxiliary data file is written to FORTRAN Unit 60 and can be either a formatted or binary file. Users supply the format as input data when the formatted option is used. For both formatted and binary files, the record format is variable length.

2.4.2 VIPRE-01 Boundary Conditions (VBC) Data File

The VBC file generated by RETRAN-3D contains data to provide transient boundary conditions for a subsequent VIPRE-01 calculation. The file structure includes a header section followed by data records representing each discrete time. The record frequency is controlled by variable TVBC on the time-step data cards, 03XXX0. The contents of the VBC file is partially controlled by the input provided on VIPRE-01 boundary conditions card, 026000Y. The user specifies the volume, junction, and control blocks for which data is to be written. Pressure and enthalpy is written for each volume, flow and flowing enthalpy for each junction, and control block output for each control block specified. The VBC also contains the RETRAN-3D code version and installation date, the problem title, and the time and date of the run that created the VBC file. Data records also include the transient time, normalized power, and direct moderator heating fraction. Core section elevations, power fractions, and core heat conductor heat fluxes are automatically included in the file.

The file structure includes a header section followed by a data record for each discrete time. The format of the VBC file is defined in Table IV.2-11 and a sample VBC file is shown in Table IV.2-12. Note that the data record section of sample VBC file (Table IV.2-12) is shown as four separate blocks, although, the file uses a single line for each data record.

Table IV.2-10

Multidimensional Cross-Section File TABLE Array Data Types

Index I	Data Type Description
01	Fast diffusion coefficient
02	Fast macroscopic absorption x-section
03	Fast macroscopic removal x-section
04	Fast macroscopic nu*fission x-section
05	Fast macroscopic kappa*fission x-section
06	Thermal diffusion coefficient
07	Thermal macroscopic absorption x-section
08	Thermal macroscopic nu*fission x-section
09	Thermal macroscopic kappa*fission x-section
10	Fast nu
11	Thermal nu
12	Xenon fast microscopic absorption x-section
13	Xenon thermal microscopic absorption x-section
14	Xenon decay constant
15	Xenon fission yield
16	Iodine fast microscopic absorption x-section
17	Iodine thermal microscopic absorption x-section
18	Iodine decay constant
19	Iodine fission yield
20	Samarium fast microscopic absorption x-section
21	Samarium thermal microscopic absorption x-section
22	Samarium decay constant
23	Samarium fission yield
24	Promethium fast microscopic absorption x-section
25	Promethium thermal microscopic absorption x-section
26	Promethium decay constant
27	Promethium fission yield
28	Inverse fast-group velocity
29	Inverse thermal-group velocity
30	Precursor group 1 neutron fraction
31	Precursor group 1 decay constant
32	Precursor group 2 neutron fraction
33	Precursor group 2 decay constant
34	Precursor group 3 neutron fraction
35	Precursor group 3 decay constant
36	Precursor group 4 neutron fraction
37	Precursor group 4 decay constant
38	Precursor group 5 neutron fraction

Table IV.2-10 (Cont'd)

Index I	Data Type Description
39	Precursor group 5 decay constant
40	Precursor group 6 neutron fraction
41	Precursor group 6 decay constant
42	East-side fast ADF
43	East-side thermal ADF
44	South-side fast ADF
45	South-side thermal ADF
46	West-side fast ADF
47	West-side thermal ADF
48	North-side fast ADF
49	North-side thermal ADF
50	Top-side fast ADF
51	Top-side thermal ADF
52	Bottom-side fast ADF
53	Bottom-side thermal ADF

Table IV.2-11

VBC File Format

Line #	Description	Format
Header Section		
1	RETRAN Title	A80
2	Date/Time/Version	A8, 2X, A8, 2X, A32
3	Number of Volumes Included (NVOLBC)	I14
4	Volume Number(s)	NVOLBC * I14
5	Number of Junctions Included (NJUNBC)	I14
6	Junction Number(s)	NJUNBC * I14
7	Number of Control Blocks Included (NOUTBC)	I14
8	Control Block Identifier(s)	NOUTBC * I14
9	Number of Core Sections (NCORE) In RETRAN-3D Problem	I14
10	Core Section Number(s)	NCORE * I14
11	Comment Line (elevations)	NA
12	Core Section Elevation(s)	NCORE * E14.6
13	Comment Line (record headings)	NA

Table IV.2-11 (Cont'd)

Heading	Description	Units
Data Record Section (one line per record, format = E14.6)		
TIMX	RETRAN-3D Transient Time	(sec)
PNRM	Normalized Core Power	(-)
DMHF	Direct Moderator Heating Fraction	(-)
PRES	Volume Pressure	(psia)
HW**	Volume Specific Enthalpy	(Btu/lbm)
…		
…	(PRES and HW** repeated for NVOL volumes)	
Note:	The junction and control block entries shown below may be absent if NJUNBC or NOUTBC are zero.	
WP**	Junction Mass Flow Rate	(lbm/sec)
HFLO	Junction Flowing Enthalpy	(Btu/lbm)
…		
…	(WP** and HFLO repeated for NJUN junctions)	
COUT	Control Block Output	(-)
…	(COUT repeated for NOUT control blocks)	
QFRA	Core Section Power Factor	(-)
…	(QFRA repeated for NCORE Core Sections)	
PHIR	Core Section Heat Flux	(Btu/hr-ft2)
…	(PHIR repeated for NCORE Core Sections)	

Total words per record = 3 + 2 * (NVOLBC + NJUNBC + NCORE) + NOUTBC

Table IV.2-12

Sample VBC File

```
UNCONTROLLED ROD WITHDRAWAL  --  RETRAN SAMPLE PROBLEM
30/10/03  07:37:58  RETRAN-3D/MOD003.1 28/08/01 EPRI
             2 VOLUMES(S)
             1            12
             2 JUNCTION(S)
            13            12
             3 CONTROL BLOCK(S)
           -10             3          -19
             3 CORE SECTION(S)
             1             2            3
               CORE SETION ELEVATIONS
  2.000000E+00  6.000000E+00  1.000000E+01
  TIMX    0     PNRM    0     DMHF    0     PRES    1     HW**    1
  0.000000E+00  1.000000E+00  1.214575E-02  2.241659E+03  6.228420E+02
  5.000000E-02  1.003753E+00  1.214575E-02  2.241661E+03  6.228420E+02
  1.000000E-01  1.007793E+00  1.214574E-02  2.241666E+03  6.228420E+02
  2.000000E-01  1.016144E+00  1.214569E-02  2.241695E+03  6.228423E+02
  3.000000E-01  1.024901E+00  1.214562E-02  2.241767E+03  6.228433E+02
  4.000000E-01  1.033928E+00  1.214542E-02  2.241878E+03  6.228454E+02
  5.000000E-01  1.043269E+00  1.214509E-02  2.242025E+03  6.228491E+02
  1.000000E+00  1.094458E+00  1.214123E-02  2.243182E+03  6.229154E+02
  1.500000E+00  1.152870E+00  1.213358E-02  2.245726E+03  6.231325E+02
  2.000000E+00  1.218422E+00  1.212245E-02  2.250533E+03  6.235799E+02
  2.500000E+00  1.194409E+00  1.210824E-02  2.257881E+03  6.243081E+02
  3.000000E+00  8.100428E-01  1.210418E-02  2.263136E+03  6.251982E+02

  PRES   12     HW**   12     WP**   13     HFLO   13     WP**   12
  2.273441E+03  5.473849E+02  2.733800E+04  5.473849E+02  8.560000E+02
  2.273443E+03  5.473849E+02  2.733794E+04  5.473849E+02  8.559999E+02
  2.273448E+03  5.473850E+02  2.733782E+04  5.473849E+02  8.560004E+02
  2.273477E+03  5.473851E+02  2.733737E+04  5.473850E+02  8.560035E+02
  2.273550E+03  5.473853E+02  2.733670E+04  5.473852E+02  8.560109E+02
  2.273661E+03  5.473858E+02  2.733595E+04  5.473855E+02  8.560249E+02
  2.273809E+03  5.473863E+02  2.733515E+04  5.473860E+02  8.560452E+02
  2.274974E+03  5.473908E+02  2.733049E+04  5.473902E+02  8.562013E+02
  2.277527E+03  5.474007E+02  2.732296E+04  5.473993E+02  8.563933E+02
  2.282345E+03  5.474194E+02  2.731441E+04  5.474170E+02  8.566802E+02
  2.289707E+03  5.474479E+02  2.730854E+04  5.474448E+02  8.570225E+02
  2.294948E+03  5.474689E+02  2.733498E+04  5.474683E+02  8.567491E+02

  HFLO   12     COUT  -10     COUT    3     COUT  -19     QFRA    1
  5.473849E+02  0.000000E+00  2.220000E+03  7.886000E-04  2.820000E-01
  5.473849E+02  3.985206E-05  2.220002E+03  7.886000E-04  2.820000E-01
  5.473849E+02  1.292729E-04  2.220005E+03  7.886000E-04  2.820000E-01
  5.473850E+02  4.440807E-04  2.220011E+03  7.886000E-04  2.820000E-01
  5.473852E+02  9.744696E-04  2.220022E+03  7.886000E-04  2.820000E-01
  5.473855E+02  1.591250E-03  2.220045E+03  7.886000E-04  2.820000E-01
  5.473860E+02  2.389306E-03  2.220089E+03  7.886000E-04  2.820000E-01
  5.473902E+02  1.166860E-02  2.220959E+03  7.886000E-04  2.820000E-01
  5.473993E+02  3.524597E-02  2.223322E+03  7.886000E-04  2.820000E-01
  5.474170E+02  6.972424E-02  2.227753E+03  7.886000E-04  2.820000E-01
  5.474448E+02  1.092893E-01  2.234943E+03  7.886000E-04  2.820000E-01
  5.474683E+02  1.150211E-01  2.243655E+03  7.886000E-04  2.820000E-01
```

Table IV.2-12 (Cont'd)

QFRA 2	QFRA 3	PHIR 1	PHIR 2	PHIR 3
4.360000E-01	2.820000E-01	1.509185E+05	2.330990E+05	1.506133E+05
4.360000E-01	2.820000E-01	1.509198E+05	2.331018E+05	1.506148E+05
4.360000E-01	2.820000E-01	1.509272E+05	2.331166E+05	1.506224E+05
4.360000E-01	2.820000E-01	1.509689E+05	2.331968E+05	1.506639E+05
4.360000E-01	2.820000E-01	1.510531E+05	2.333517E+05	1.507472E+05
4.360000E-01	2.820000E-01	1.511713E+05	2.335689E+05	1.508601E+05
4.360000E-01	2.820000E-01	1.513239E+05	2.338456E+05	1.510035E+05
4.360000E-01	2.820000E-01	1.525678E+05	2.360376E+05	1.521403E+05
4.360000E-01	2.820000E-01	1.545804E+05	2.395140E+05	1.539795E+05
4.360000E-01	2.820000E-01	1.573591E+05	2.442662E+05	1.565474E+05
4.360000E-01	2.820000E-01	1.605313E+05	2.495943E+05	1.594745E+05
4.360000E-01	2.820000E-01	1.586427E+05	2.454178E+05	1.576621E+05

2.5 Temporary Files

Temporary files are used to store input data and intermediate results that are accumulated and processed at some later stage of the job. Only formatted and unformatted Fortran I/O methods are used for the temporary files. The temporary files are discussed below.

2.5.1 Trip Summary File

A temporary file is written during RETRAN-3D execution to store the trip actuation history. It is written to Fortran Unit 3 using a fixed record length binary write. Information describing each trip actuation event is written to Unit 3 and upon completion of the job, the file is rewound and the history is edited on the standard output file.

2.5.2 Printer Plot Data Summary File

A variable record length binary file is used for intermediate data storage required for the printer plot option. When printer plots are requested during a run, the minor edit parameters are written to Fortran unit 20 each minor edit time interval. The file is rewound at problem completion and the printer plots are then generated from the summary file.

2.6 Input Data File

Input data used to describe the RETRAN-3D problems is supplied in 80-column card image form. The data cards are processed by the INP free form input package (see Section V) which reads them from Fortran Unit 5. No I/O references to Fortran unit 5 can be made except those from INP.

2.7 Output Print File

Hardcopy output of RETRAN-3D solution results is written to Fortran Unit 6. This output is formatted with standard FORTRAN printer carriage control characters and has a 132-character record length.

2.8 Plot Data File

RETRAN-3D has an option that will allow a data file to be created for subsequent use by other applications to create X-Y plots or other graphical representations of problem solutions. T he plot file is similar to the auxiliary file (with header) in that it contains problem-specific documentation of the run date and time, problem name, and code version. It also contains the minor edit descriptions and units that can be used for default plot labels and minor edit request flag and region number pairs that can be used to locate the data for a given minor edit. Following the header

information, separate records are written for each time point for which a minor edit is written. The record contains the values for each plot variable.

Two optional built-in lists of minor edit variables are available for determining the content of a plot file. They are the short and long forms and are comprised of fixed lists of minor edits that are identified in Table IV.4-13. The short built-in list of minor edit variables consists of only important minor edit variables, e.g., volumes, junctions, system parameters. The long built-in list contains additional parameters that may be required for detailed analysis using the plot file information. Other minor edit variables can be added to the short or long list built-in minor edit variables using the optional plot file supplemental variable list (data card 027XXX in Section IV-4.1.3 of Volume 3) to specify additional minor edit variables to be included.

Table IV.2-13

RETRAN-3D Plot File Built-in Minor Edit Lists

Built-in List	Minor Edits
Short (1)	TIMX, PNRM, POWR, PRES, TEMP, HW**, AVEX, VOIV, LIQL, WP**, GPM*, HP**, HFLO, SLPR, SPED, MIXL, PHIL, PHIR, WQCL, WQCR, IHTL, IHTR, TAVG, PERD, REAC, RC**, RW**, RV**, RF**, RD**, PPOW, DPOW, RO**, RDRE, DPRE, VDRE, RHOA, CTRA, DOPA, MODA, DENA, BORA, XISA, RESA, NELT, NELX, NEVT, NEVX, COUT, TRIP
Long (2)	TIMX, TIMM, TIMH, PNRM, POWR, DCAY, PRPT, DTAV, NSTP, NTTS, CPUS, QLOS, PTHR, DTOL, PRES, TEMP, SATT, HW**, AVEX, FMAS, AVED, WQ**, VOIV, GASH, LIQH, STVF, STVG, STHF, STHG, WVBR, NVOI, LIQV, VL**, VS**, LIQL, CONV, MDOT, TEML, MDDT, WP**, GPM*, XP**, HP**, AVDJ, HFLO, VOIJ, WLJ*, VGJ*, VLJ*, SLPR, DELF, SPVJ, PJUN, AJNT, VLSJ, SPED, PMPW, VBUB, MIXL, BUBM, MIXQ, SE**, PHIL, PHIR, WQCL, WQCR, IHTL, IHTR, TL**, TR**, FCHL, FCHR, HTCL, HTCR, GR**, CORQ, TAVG, MODQ, QFRA, FENT, PERD, REAC, RC**, RW**, RV**, RF**, RD**, PPOW, DPOW, RO**, RDRE, DPRE, VDRE, A1**, A2**, RHOA, CTRA, DOPA, MODA, DENA, BORA, XISA, RESA, NELH, NELT, NELX, NEVH, NEVT, NEVX, COUT, TRIP

In addition to the minor edit variables included in the RETRAN-3D plot file as described above, user-specified minor edits requested on the 0200XX data are also included in the plot file. Only minor edit variables that are not already included in the plot file are added as supplements to the plot file content.

The structure of the plot file header records and data records are shown in Table IV.2-14. Records 1 through 8 comprise the header, which provides the descriptive information needed to identify plot variables and the associated data. It also contains descriptive text fields.

Table IV.2-14

RETRAN-3D Plot File Structure`

Record	Variable	Type	Format	Length	Description
1	**plot_file_hdr**	C-32	A32	1	RETRAN-3D Plot File
2	**problem_title**	C-80	A80	1	Problem Title
3	**run_date**	C-8	A8	1	Date Problem was Run
	waltim	C-8	A8	1	Time of Day Problem was Run
	version	C-32	A32	1	Code Version
	plt_si	I-4	I8	1	SI Units Flag - =0 For Engr., =1 for SI
	num_plot_vars	I-4	I8	1	Number of Plot Variables in Each Record
4	**flg_reg**	C-8	20A8	**num_plot_vars**	Merged Minor Edit Flag and Region (4 char each)
5	**blk_title**	C-8	20A8	**num_plot_vars**	Minor Edit Block Title
6	**desc**	C-12	20A12	**num_plot_vars**	Variable Description
7	**units**	C-8	20A8	**num_plot_vars**	Units String
8	**units_id**	C-4	20I8	**num_plot_vars**	Units ID
9-n	**plot_var**	R-4	20E12.5	**num_plot_vars**	Plot Variable Values

The plot file will contain records of plot variable results at various times during the simulation. The intervals at which data records are written are determined by the minor edit intervals given on the time-step control data cards (see User's Manual, Volume 3, Section IV.5.0).

The plot file name is **R3D_PLOT**. It can be either an ASCII or binary file. ASCII files can be examined using a text exiting program, but binary files cannot be easily examined. The size of a plot file is affected by the number of volumes, junctions, and optional components as well as the number of nodes. Binary plot files will be approximately one third the size of the corresponding ASCII file. Since plot files may be large, using binary plot files will save a significant amount of space. An ASCII plot file can be three to five times larger than the associated output file.

By default, no plot file is written. If a plot file is desired, one can be requested and its type (binary or ASCII) can be defined through the auxiliary file description data card (025000).

Subroutine **build_plot_var_list** is called from subroutine **inedit** to build the list of plot variables that will be included in the plot file. The plot file header records are written by subroutine **write_plot_header** and the time-dependent plot records are written by subroutine **write_plot_record**. Both subroutines are called by subroutine **edit**. Subroutines **build_plot_var_list**, **write_plot_header,** and **write_plot_record** are contained in module **R3D_plot_file**, which is found in source file **m_R3D_plot_file.f90**.

A simple search of the **flg_reg** record (after it has been read into memory) for a merged minor edit flag and region (eight character word – four characters for flag and four for the region), will identify the index into the **plot_var** array for the given minor edit. Time points are in subsequent records. Appendix E discusses the use of a utility program get_R3D_plot_vars, which can be used to extract data for selected plot variables from a RETRAN-3D plot file.

2.9 Remark Log

Entries are made to a remark log at various stages of input processing and steady-state initialization. Each entry (record) includes the wall time and a character message. The log is written to file REMARKS on Fortran unit 8 and has a record length of 90 characters.

2.10 Error Log

Error messages for most error conditions encountered during a RETRAN-3D solution are written to file ERR_LOG on Fortran unit 17. It is a text file containing time, date, time-step number, and time-step size information for the time when the error occurred. It also contains code version information along with textual information describing the error condition. Some diagnostic information may also be included.

2.11 RETRAN-3D Configuration File

In some instances it is necessary to define the length of work arrays where the length is not known before the fact or to define text strings or other data. Typically these lengths or data would be hardwired and a code modification and re-installation would be required to change any of them. For example, licensed organizations would have to recompile the code to change the organization name to their organization for it to be included in the documentation header (PC distributions are the exception since the licensed organization is included in the license file and is retrieved as part of the license validation). A new feature is included in RETRAN-3D that allows these types of information to be defined as default values, which can be overridden by providing new values in a RETRAN-3D configuration file.

Use of a RETRAN-3D configuration file is optional. If one is provided it must be named **retran.cfg** (must be lower case on Linux platforms) and must reside in the same directory as the RETRAN-3D input file. Table IV.2-15 lists the variables that can be changed using the configuration file. The default values are used if a configuration file is not provided.

A configuration file is a text file with a maximum record length of 132 characters, which is read from Fortran Unit 30. Records within the file are comprised of two fields, one is a keyword used to identify the content of the record and the other is the associated field value. The variable name in Table IV.2-14 is used as the keyword and the redefined value is the other. Both the keyword and value are bounded by square brackets. An example record might look like

```
[inp_data]  [1000]   ! change length of vector used to read inp data
```

A configuration file may contain one record or many keyword records. They can be in any order. If a keyword is supplied more than once, the final instance will be the one used.

The RETRAN-3D configuration file provides the ability to activate steady-state and transient solution matrix debug edits. This feature typically will be used by programmers and for debugging. For steady state, XANDH debugs can be turned on and off by supplying 'xandh_istart' and 'xandh_iend' values. For transient solution matrix debugs, the user has more flexibility. Various parts of the matrix can be printed as required. 'genmt_ntcoeb' and 'genmt_ntba' specifies the actual time-step number at which coefficient matrix and right-hand side solution matrix are printed. To turn off all debugs, set 'genmt_ntlast' to the actual time-step number at which debugs will stop printing and the code will also stop running.

The configuration file can be used to activate writing of a plot file by changing the value of **plot_type**. It also determines if the file will be written as a binary or ASCII file.

Table IV.2-15

RETRAN-3D Configuration File Variables

Variable	Data Type	Definition	Default
inp_data	I-ikd	Length of the r_data real(kind=8) vector used by the inp routines (e.g. inp2) to return data as it is read.	500
genmt_ntcoef	I-ikd	Time-step number at which transient solution matrix coefficients will be printed by genmt3 or genmt4	1000000
genmt_ntba	I-ikd	Time-step number at which transient solution matrix right-side will be printed by genmt3 or genmt4	1000000
genmt_ntlast	I-ikd	Time-step number at which genmt3 or genmt4 debugs will stop printing and stop the current run.	1000000
len_rblk	I-ikd	Length of vector used by restart routines, used during restart run.	50
organization	C-ckd,len=32	Name of the organization licensed to use RETRAN-3D	' ** Unlicensed Trial Version ** '
rstrt_blks	I-ikd	Maximum number of blocks saved in restart block list.	200
xandh_istart	I-ikd	Steady-state iteration number at which XANDH matrix debugs will begin priniting	10000
xandh_iend	I-ikd	Steady-state iteration number at which XANDH matrix debugs will stop printing	10000

[1]C = Character, I = Integer, L = Logical, R = Real – Parameters on right are defined in module kind_specs and indicate the size.

V

inp FREE FORM INPUT

The **inp** subroutines constitute a convenient free form data input package for use with Fortran programs. The **inp** package is based on a similar subroutine package in the INEL Environmental library.[V.1-1] To the user, the package offers: free form input; card numbers to identify data cards; automatic removal of cards containing duplicate card numbers; arbitrary ordering of input cards except for duplicate, continuation, and terminator cards; arbitrary use of comment cards and comments on data cards; ease of preparing cases in which only moderate changes are made from case to case; and a listing of the card data. For the programmer, the package is a convenient method for implementing a highly user-oriented data input scheme and includes: extensive checking of the data mode (integer, floating point, or alphanumeric) and number of data items; automatic expansion of sequential and overlay type of input data; deletion of superfluous cards; checking whether extraneous input has been entered; and the ability to detect errors during input checking.

1.0 USER ASPECTS OF inp PACKAGE

This section describes the **inp** package as seen by the program user. It discusses the use of card numbers, field delimiters, and deck separators.

1.1 Data Deck Organization

The data deck contains input for one or more problem sets. No relationship is assumed between problem sets. Each problem set consists of one or more cases in which the input data for cases other than the first consist of the data from the previous case plus modification cards entered for the present case. Input data for cases are separated by slash cards; the final case is terminated by a period card instead of a slash card. The period card also serves as the separator between problem sets. A slash card has a (/) as the first nonblank character on a card; a period card has a (.) as the first nonblank character. Comments may follow the slash and period on slash and period cards.

A list containing a card sequence number and the card image of each card is printed at the beginning of printed output for each case. The card sequence number starts at one for each case. The first line of the list contains "Listing of input data for case n", where n is the case number.

1.2 Title Card

A title card is designated by an equal sign (=) as the first nonblank character on a card. The remainder of the card can have any alphanumeric characters. The information on the title card and the current date are printed at the top of every page following the input data listing. One title card should be entered for each case. If more than one title card is entered in a case, the contents of the last title card are used for the page heading. The heading contains only the date if no title card is entered for a case.

1.3 Comment Cards

An asterisk (*) or a dollar sign ($) appearing as the first nonblank character identifies the card as a comment card. Any information may be entered on the remainder of the card. Blank cards are treated as comment cards. There is no processing of comment cards other than listing them in the card list.

1.4 Data Cards

All cards other than title cards, comment cards, slash cards, or period cards are considered data cards. The data cards contain a varying number of fields which may be decimal integer, decimal floating point, alphanumeric, octal, or hex. The rules for specifying fields are as follows.

Blanks preceding and following fields are ignored. A decimal field is started by either a digit (0 through 9), a sign (+ or -), or a decimal point (.). A comma or a blank (with one exception noted below) terminates the decimal field. The decimal field has a number part and, optionally, an exponent part. A decimal field without a decimal point or an exponent is a decimal integer field. A field with either a decimal point or an exponent or both is a decimal floating point field. A decimal floating point field without a decimal point is assumed to have a decimal point immediately in front of the first digit. The exponent denotes the power of ten to be applied to the number part of the field. The exponent part is a sign, an E or D, or an E or D and a sign followed by a number giving the power of ten. Rules for decimal floating point numbers are identical to those for entering data in Fortran E or F formatted fields except that no blanks (one exception) are allowed between characters. Formatted floating point data written by Fortran programs can be read. To permit this, a blank following an E or D denoting an exponent is treated as a plus sign. Acceptable ways of entering floating point numbers are illustrated by the following seven fields all containing the quantity 12.45,

```
12.45
+12.45
1245+2
1.245+1
1.245E1
1.245E+1
1.245E 1
```

When entering a zero for either an integer or floating point quantity, it can be written in either form. Thus, a floating point zero can be entered simply as 0 without a decimal point. A field starting with a nonblank character other than a digit, sign, comma, period or decimal point, asterisk, dollar sign, or slash is considered a default alphanumeric field. The field is terminated by a comma or the end of the card. All characters except commas can be imbedded in the field. Imbedded blanks are considered part of the alphanumeric field and do not terminate the field. Blanks extending from the last nonblank character of an alphanumeric field to the end of the card are not considered part of the field. An alphanumeric field can also be delimited by enclosing the field within apostrophes (') or quotes ("). Either is acceptable but they must be used in pairs. A blank or comma must follow the terminating delimiter. Subroutine **cvi** is used by the **inp** package to process free format data on cards as discussed in Section V.3.1.

Data on a card may be continued on a continuation card by entering a plus sign as the first nonblank character on the continuation card. A field starting on a card must be completed on that card and may not continue to the next card. The plus sign indicating the continuation card is not considered part of the first data field on the continuation card and may be placed alone or adjacent to the first data field. Continuation cards themselves may be continued. In subsequent processing, data on continuation cards are treated as if the data were all entered on one card.

Comment information may follow the data fields on any data card (including cards that are continued) by preceding the comments with an asterisk or dollar sign. A default alphanumeric field preceding a comment must be terminated by a comma or the comment information is considered part of the alphanumeric field.
When card format errors are detected, lines containing a $ located under the character causing the error and a comment giving the card column of the error are printed. A field containing an error is converted as an alphanumeric field of $$$$$$$. An error flag is set and input processing continues, but the job can be aborted at the end of input processing. Usually another error is produced by a routine attempting to process the erroneous data.

The first field on a data card is treated as a card number which must be a positive decimal integer number. If the first field has an error or is not a positive decimal integer, the card number is replaced by the current card sequence number, an error statement is printed, and the error flag is set. Data on the card is not used and the card will be identified by the card sequence number if the list of unused data cards is printed. Continuation cards do not have card numbers since they are considered an extension of the first card. After each card number and the accompanying data are converted, the card number is compared to previously entered card numbers. If a matching card number is found, the data entered on the previous card is replaced by the data of the current card. If the card being processed contains only a card number, the card number and the data entered on the previous card are deleted. If a card causes replacement or deletion of data, a statement is printed indicating that the card is a replacement card.

The list of card numbers and associated data used in a case can be passed to the next case. Cards entered for the next case are added to the passed list or act as replacement cards depending on the card number. The resulting input to cases following the first case is the same as if all previous slash cards were removed from the input to the problem set.

2.0 PROGRAMMING USE OF THE inp PACKAGE

The **inp** package contains a collection of subroutines or entry points which can be used by a programmer to implement free form input processing in an application. One call is issued to subroutine **inp** for each case in order to read and convert the data for the case, to replace or remove duplicate cards, and to form a sorted table of card numbers cross referenced to a list. The list contains data words obtained from the cards and mode words generated during input conversion. Because of the sorted table, the order of cards in the data deck is not important.

The **link** subroutine accesses the table and can locate one card at a time. Subroutine **moder** is used to check the appropriate mode of the data against a specified list, also one card at a time. The **inp2** subroutine is used to check data and to move data from the list to a specified array by using calls to **link** and **moder**. It can be used to process data from a single card or from a set of cards numbered within a specified range and to check for minimum and maximum numbers of items and appropriate mode. The **inp4** subroutine executes repeated calls to **inp2** and modifies the call parameters to **inp2** by specified amounts. Function **inp8** can be used to determine whether there are cards in the list that have not been referenced by **link** and, thus, also **inp2**. The function **inp9** deletes cards from the table and list which have been referenced by **link** and **inp2**. The function **inp10** deletes selected cards from the table and list.

The following sections describe programming requirements for using the **inp** package. In the following descriptions, calling parameters are named for their integer or floating point format; that is, integer quantities have I, J, K, L, M, or N as their first character and names beginning with any other character are floating point quantities. Symbols appearing in the calling sequences are unique and when the same symbol appears in two or more calling sequences, the symbol has the same definition in each appearance. The symbol is usually completely described only in its first appearance, but the definitions are summarized in the Array and Variable Summaries (Section V.4.2 and V.4.3). Calling parameters that are marked with an asterisk both convey information to the subroutine and return information from the subroutine and, thus, the parameter can have a different value on exit than it had on entry. The error messages referenced in the descriptions are listed in the Error Message Summary, Section V.4.4. Programming errors such as improper calling parameters cause an abnormal termination.

Input data cards can have a mixture of integer, floating point, and alphanumeric data and the **inp** subroutine converts and stores all data into the list as 64-bit words. Consecutive integers on an input card will reside in consecutive 64-bit words. Since integers only occupy 32-bits, 32-bits of padding is associated with each. This padding is accomplished by use of two-dimensional integer arrays. The **inp2** subroutine, after checking the data against a specified list for the correct mode, moves the data to a specified output array if the move flag is set. Accessing data from this array containing mixed integer and floating point data is easily accomplished by equivalencing integer and floating point names to the array containing the mixed data. The integer array is dimensioned (2,n).

In RETRAN-3D, the **inp_tbl** array is used to store the **inp** package card data list and the **rdata** array is used to return card data from **inp** package subroutines. Both arrays are allocated in subroutine **ret_input** via a call to subroutine **allok8_inp_card_arrays**. The length of array **rdata**

is defined by the **inp_data** variable, which can be revised using the **retran.cfg** file (see Section IV.2.10) without making a source code revision. As noted above, all card data whether real, integer or character is returned to the calling subroutine via the **rdata** array. Real data can be accessed directly. For example, following a call to subroutine **inp2** to located the card data for a given control volume, the real data is obtained as illustrated in the following code segment

```
p(i5)     = rdata(3)
temp(i5)  = rdata(4)
hum(i5)   = rdata(5)
v(i5)     = rdata(6)
zvol(i5)  = rdata(7)
flowl(i5) = rdata(8)
flowa(i5) = rdata(9)
diamv(i5) = rdata(10)
elev(i5)  = rdata(11)
ew(i5)    = rdata(12)
```

The integer data for the volume is accessed as shown in the following code segment.

```
ibub(i5)  = idata(1)
iread(i5) = idata(2)
```

Note that **idata()** is not an integer array, but rather an integer function that returns the 32-bit integer that is stored in a given element of array **rdata**. In the example above, **idata(1)** returns the 32-bit integer that is stored in **rdata(1)**.

As a warning, programmers should not apply generic functions to **rdata** (or **inp_tbl**) elements directly, e.g. don't use **abs (rdata(i))**. Use them after the data element has been moved into a RETRAN-3D data or pointer element. If the **abs** function is applied to an integer element stored in **rdata**, the actual value of the integer can be changed instead of simply changing the sign (this depends on the whether the platform uses big or little endian architecture).

Character data included on an **inp** data card is also returned to the application using the real **rdata** array. A character string less than or equal to 8 characters in length is returned in one **rdata** element. Strings longer than 8 characters are returned in consecutive **rdata** elements. The number of elements required to return a string is given by num elem = (len + 7)/8 where len is the number of characters in the string.

The **inp** package was originally designed and implemented when the Fortran language used either integer of real variables to store character data. As such, the data type sets the number of characters that can be stored in a given word (typically 4 for integers and 8 for 8-byte real variables). For this reason, subroutine **inp** uses the **inp_tbl** real array to store character strings in the card list and the **rdata** real array to return character strings. For this reason, character data returned by the **inp** package may be stored in real variables for latter use and in some instances they are converted and stored as character data to avoid data type conflicts. The following code segment illustrates converting a character string returned in the **rdata** array to a 4-character variable **flag**.

```
write (flag,'(a4)') rdata(k)
```

2.1 call inp (xl1, nl1, title*, ncase*, ndata*, isw*)

One call to **inp** reads all the input cards for the next case. That is, each call to **inp** reads cards from the current position on Fortran Unit 5 until either a slash card or a period card is read or the end of the data set is encountered.

The quantity **ncase** is incremented by one. **ncase** should be set to zero before the first call to **inp**. If a period card terminates the case, the sign of **ncase** is set to minus. The calling program can test the sign of **ncase** to determine whether this case is the last case of problem set or another case follows. If **ncase** is negative indicating the end of a problem set, **ncase** should be reset to zero before calling **inp** for the first case of the next problem set.

As input cards are read, they are printed without modification. Before printing the first card, the first heading line is printed at the top of a page and followed by "Listing of input data for case n", where n is **ncase** after it has been incremented by one. Subsequent pages of the input data listing have only the first heading line at the top of an output page. As title cards are read, the title information with the equal sign removed overlays the title storage array **title**, which was initialized to blanks. **title** should be at least 12 real 64-bit words. The title printing noted above is performed if **isw** $\neq 0$ upon entry to **inp**. If **isw** = 0, no printing is performed.

The real array **xl1** is a **nl1** words in length and is used to store the list containing card data and the card number table plus two control words. The control words are stored in **xl1**(1) and **xl1**(2); the list is stored starting at **xl1**(3) and extends upward in the array; the table is stored starting at **xl1**(**nl1**) and extends downward in the array. The space between the list and table is used for temporary working space. Data card information is converted to binary form through calls to the **cvi** subroutine. Binary information from a data card that is not a continuation card is stored beginning at the middle of the temporary work space. The binary information and mode indicators for continuation cards are stored following the information and mode indicators for the proceeding card. As each individual data card is processed, there must be 40 words between the end of the list or the last converted binary quantity and the beginning of the mode indicators. This space is necessary to prevent the binary quantities from over storing the mode indicators and the mode indicators from over storing the table; 40 words are necessary since that is the largest number of quantities that can be entered on an 80-column card. After the data card and any continuation cards have been converted the binary data and mode indicators are stored as if the data were entered on one card and subsequent processing can proceed as if only one card was entered. Three table entries are then constructed for each card. They consist of the card number or a zero if the card number is illegal, a pointer to the table location where the binary information is stored, and the number of words on the card excluding the card number. The entry for the number of words is also used as a card use indicator. Card numbers are initially positive, but are set negative when the card is processed by a call to subroutine **link**. If there are data other than the card number on the card, the binary information is moved downward one word eliminating the card number from the list, and the corresponding mode indicators returned from subroutine **cvi** are converted to two-bit indicators and stored in a packed form, 32 indicators per word, following the binary information (done in subroutine **inp**). The table words, three for each card, constitute the table and the list is made up of the converted binary information and the mode indicators.

As data cards are processed, new card numbers are compared against the card numbers stored in the table. If a duplicate is found, the table word containing the card number is replaced by the new table word. Space occupied by the replaced list data is retrieved by shifting the data downward over the replaced data when the number of words on the replaced card and the new card are different or simply by over storing the replaced list data with the new list data when the number of words is the same. Table pointers are updated when list data are moved. When a replacement card contains only a card number, the card acts as a deletion card; the table word is deleted and the list space is retrieved by shifting the list over the deleted data. A message "This card is a replacement card", is printed to the right of any card that replaces or deletes a data card. A card containing only a card number that is not a duplicate card number has no effect on the table and list and no message is printed.

A normal return from the **inp** subroutine is made if a period or slash card is read or an end of data set is encountered after at least one input card was read. Before a normal return, the table is sorted by card number and is moved adjacent to the list and the number of words in the list is stored in **xl1**(1) and the number of words in the table is stored in **xl1**(2). Upon normal exit, the absolute value of **ndata** is set equal to the number of words needed in **xl1** to hold the control words, list, and table. The sign of **ndata** is set minus if no data cards (cards other than title, comments, slash, or period cards) are entered for a case, and in normal usage this indicates that no input data were entered and that a succeeding case may have been input identical to the preceding case.

On entry to **inp**, **ndata** indicates whether the array **xl1** contains data from a previous case. If **ndata** is equal to or less than zero, **xl1** contains no data from a previous case and the table and list are assumed empty. If **ndata** is greater than zero, **xl1** is assumed to contain data from a previous problem in the same format as that upon exit from **inp**. That is, **xl1**(1) contains a control word containing the number of words in the list and **xl1**(2) contains the number of words in the table and these control words are followed by the list and table. The table is moved to the end of **xl1** and the card use indicators in the table are cleared by replacing the card number entry with its absolute value. The input cards for the current case are then processed as described above.

The parameter **isw** controls the list edit and indicates the return status. If **isw** is zero when **inp** is called, the card list edit is suppressed. The card list edit is provided if **isw** is nonzero. If **isw** is zero on return, a normal return was made and no errors were detected during the processing of the input cards. If **isw** is one, the end of data set was encountered when trying to read the first data card of a case and **inp** returned immediately. This is the normal exit path if **ncase** is not zero. If **isw** is two, a normal return was made, but card format errors were detected and the list of input cards contains one or more of Error Messages 3 through 6. The usual practice in this case is to continue checking the input data for additional errors, but execution is terminated after input checking is completed. If **isw** is three, the array **xl1** is not large enough to process the input data as indicated by Error Message 1 or 2 and **inp** returned immediately.

2.2 call inp2 (xl1, xl2, l3)

The array **xl1** contains the list and table data generated by subroutine **inp**. The array **xl2** is the array into which the data specified in the call is to be moved. The array **l3** contains specifications as follows:

l3(1)	**ic1**, first card number.
l3(2)	±**ic2**, last card number. **ic1** and **ic2** specify the set of card numbers of the data that are to be moved into **xl2**. IF **ic2** is zero, only the card with card number **ic1** is specified. If **ic2** is nonzero, cards with card number c, **ic1** $\leq$ c $\leq$ |**ic2**|, are requested. Not all the cards in the range of c need be present. If **ic2** is positive, the card numbers that are present within the range must be sequential and are processed in sequential order beginning with **ic1**. If card number **ic1**, **ic1**+1, **ic1**+2 . . ., **ic1**+a are present where **ic1**+a is the last sequential card number, a card with number c_x, **ic1**+a + 2 $\leq c_x \leq$ **ic2**, is an error and causes Error Message 8 to be printed. If **ic2** is negative, the cards need not be sequential and are processed in increasing order.
l3(3)	**min**, the minimum number of items to be processed. Error Message 9 is printed if fewer items are processed.
l3(4)	**max**, the maximum number of items to be processed; ignored if zero. Error Message 10 is printed if more items are processed.
l3(5)	**nj**, the number of words to skip between items in **xl2**; usually zero.
l3(6)	±**j***. **j**, if positive, is the starting location in **xl2**; input item n is stored in **xl2**(**j**+(**nj**+1)*(n-1)). If **j** is negative, no data is moved, but all checking is performed.
l3(7) . **l3**(n)	An array defining the integer, floating point, or alphanumeric format expected on the cards. Format information is defined in **moder** description. Errors cause Error Messages 7, 11, or 12 to be printed.

Subroutine **link** is used to locate each card and subroutine **moder** is used to check the format.

On exit from **inp2**, **l3**(6) or **j** is set to **nmove**, the number of items moved, if no errors were found and **j** was positive on entry. **l3**(6) or **j** can be zero indicating no cards with the specified card numbers are present if **l3**(3) or **min** is set to zero. If any card requested contained a card format error (as detected by **inp**) or if any of the tests specified in the **l3** array failed, **l3**(6) or **j** is set to -1 on exit. If **j** was negative on entry, **l3**(6) or **j** is set to -**nmove** on exit if no errors were found. Only the first error in the specified set of cards is found and Error Messages 13 and 14 can be printed by **inp**2 in addition to those noted in the definition of **l3**.

2.3 call inp4 (ic1, ±ic2, min, max, nj, j*, ic3, ntimes, newj, xl1, xl2, l5)

Subroutine **inp4** makes **ntimes** calls to **inp2**. An abnormal termination occurs if **ntimes** is zero or negative. For the first call to subroutine **inp2**, **ic1**, **ic2**, **min**, **max**, **nj**, and **j** are as described for **inp2** and **l5** is the format array for checking the mode of the data. For the following calls, **ic1 and ic2** are increased by **ic3**, and **j**, if positive, is increased by **newj. j** is changed upon exit as in the **inp2** description. **ic1** and **ic2** on exit have the same value as on entry. Maximum size of the **l5** array is 40.

2.4 call inp5 (ic1, ±ic2, ic3, ±n1, ±nmin, ±nmax, ±nstore, ntimes, newj, j*, xl1, xl2, l5, xl6, nl6)

The subroutine **inp5** is similar to **inp4** in that it makes **ntimes** calls on **inp2**, but is more powerful in that it accepts self-expanding data of the sequential or overlay type. The basic Unit of input data is a vector, $\underline{S}$, of |**n1**| components where N1 is defined as an input argument. If N1 is positive, the data is of sequential form where ($\underline{S}_k$,n_k), k = 1, . . . K means that the vector $\underline{S}_k$ is to be repeated n_k-n_{k-1} times, i.e., expanded into vectors n_{k-1} + 1 through n_k. The variable k is the number of vectors on the data cards with n_o = |**nmin**|, $n_k > n_{k-1}$, and $n_k \leq$ |**nmax**|.

The expanded vectors, $\underline{S}_i$, form a two-dimensional array with elements, $s_{\ell,i}$ (the ℓ^{th} component of $\underline{S}_i$), where $1 \leq \ell \leq$ |**n1**|, |**nmin**| + $1 \leq i \leq$ |**nmax**|. This array can be stored into **xl2** in two different modes where one mode is the transpose of the other mode. If **nstore** is positive, $s_{\ell,i}$ is stored in **xl2**(**j**+(ℓ-1) + **nstore***(i-1)) and **nstore** should be greater than or equal to |**n1**|. If **nstore** is negative, $s_{\ell,i}$ is stored in **xl2**(**j**+(i-1) + |**nstore**|*(ℓ-1)) and the proper size of **nstore** depends on **nmin** and **nmax**. This is equivalent to having a two-dimensional array defined as: **dimension** SS(**nstore**,n) and **equivalence** (**xl2**(**j**), SS(1,1)). If **nstore** is positive, $s_{\ell,i}$ is stored in SS(L, I), and if **nstore** is negative, $s_{\ell,i}$ is stored in SS(I, L).
As an example, let

```
n1       =   2
nmin     =   0
nmax     =  10
nstore   =   2
ntimes   =   1
newj     =   0
```

and let the vectors $\underline{S}_k$ be given by

$\underline{S}_1$ = (1.0, 10.0), $\underline{S}_2$ = (2.0, 20.0), $\underline{S}_3$ = (3.0, 30.0) ,

and n_k be given as n_1 = 2, n_2 = 4, n_3 = 10. These data on a card could appear as follows:

```
xxxxxx   1.0,10.0,2   2.0,20.0,4   3.0,30.0,10
```

where xxxxxx is the card number. The expanded data would be stored in core as a two-dimensional matrix SS(K, I) and would appear as:

```
 1.0   1.0   2.0   2.0   3.0   3.0   3.0   3.0   3.0   3.0
10.0  10.0  20.0  20.0  30.0  30.0  30.0  30.0  30.0  30.0  .
```

Now, let

```
n1       =    2
nmin     =    0
nmax     =   10
nstore   =   -2
ntimes   =    1
newj     =    0
```

and use the same data as above. Since NSTORE is negative the expanded matrix is stored as the transpose, i.e., SS(I, K) and would appear in core as

```
1.0  10.0
1.0  10.0
2.0  20.0
2.0  20.0
3.0  30.0
3.0  30.0
3.0  30.0
3.0  30.0
3.0  30.0
3.0  30.0
```

If **n1** is negative, the data is of the overlay form (m_k,$\underline{S}_k$, n_k), k = 1, . . .K where $\underline{S}_k$ is overlaid on an initial set of vectors beginning at the m_k^{th} vector and extending through the n_k^{th} vector, with $m_k \leq n_k$, min $m_k \geq$ **nmin**, and max $n_k \leq$ |**nmax**|. For either type there results a sequence of expanded input data of the form $\underline{S}_i$, $i_o \leq i \leq$ |**nmax**|, where certain of the $\underline{S}_i$ may be missing in case of overlay expansion. Here i_o = |**nmin**| + 1 or |**nmin**| for sequential and overlay types, respectively. For overlay data, a positive **nmin** requires that the lower limit be included while a negative **nmin** can be used for negative indexing. For both types a positive **nmax** is used to require that the upper limit be included while a negative **nmax** only specifies an upper bound. The initial vectors being overlaid may be null since the complete matrix can be overlaid initially.

As an example of the overlay feature, let

```
n1       =  -2
nmin     =   1
nmax     =  10
nstore   =   2
ntimes   =   1
newj     =   0
```

and let the vectors $\underline{S}_k$ be given by

$\underline{S}_1$ = (1.0, -1.0), $\underline{S}_2$ = (2.0, -2.0), $\underline{S}_3$ = (3.0, -3.0) .

Assume a data card is given by

```
xxxxxx 1,1.0,-1.0,10  4,2.0,-2.0,8  5,3.0,-3.0,7  .
```

The first set overlays all ten vectors in SS (K, I) which then appears as

```
 1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0   1.0
-1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  -1.0  .
```

The second set of data on the card overlay vectors four through eight with the new vector $\underline{S}_2$. The matrix SS (K, I) is now given as

```
 1.0   1.0.   1.0   2.0   2.0   2.0   2.0   2.0   1.0   1.0
-1.0  -1.0   -1.0  -2.0  -2.0  -2.0  -2.0  -2.0  -1.0  -1.0  .
```

The third set of data overlays vectors five through seven in SS(K,I) which then becomes

```
 1.0   1.0   1.0   2.0   3.0   3.0   3.0   2.0   1.0   1.0
-1.0  -1.0  -1.0  -2.0  -3.0  -3.0  -3.0  -2.0  -1.0  -1.0   .
```

If **inp5** was called as in the third example but with **nstore** = -10, the result would be the transpose of the last matrix above. Note that the value of **nstore** and N1 must be consistent with the matrix ordering.

ic1, **ic2**, **ic3**, **ntimes**, **j**, **newj**, and **l5** are handled as in **inp4**. J on exit contains not the amount of expanded data, but the amount of data on the cards. The array XL**6** of length NL**6** is used for temporary storage and must be large enough to hold the unexpanded input data for one card set **ic1** $\leq$ c $\leq$ **ic2**. Additional error checks are made because of the form of the input, and Error Messages 15 through 20 can be printed. Subroutine **inp6** is called for error processing. If the parameters **n1**, **nstore**, or **ntimes** are less than or equal to zero, an abnormal termination occurs.

2.5 call inp6 (ic1, ic2, n1, icard, item, xl1)

Subroutine **inp6** is called from subroutine **inp5** when item **item** on a set of cards **ic1** $\leq$ **c** $\leq$ **ic2** is found to me in error. On exit, **icard** is the card number and **item** is the number of the field on the card containing the error. **n1** is the number of items to find on the set of cards identified above.

2.6 call inp7 (icard, item)

Subroutine **inp7** simply prints Error Message 22 stating that item **item** on card **icard** is in error. This subroutine can be used to print error information obtained from **inp6**. This subroutine is not called by other **inp** package routines.

2.7 call link (ic, ix, n3, n4, xl1)

Subroutine **link** searches the table and list array, **xl1**, for Card |**ic**|. The subroutine exits with **ix** equal to the card number in the table next larger than |**ic**| unless such a card does not exist and then **ix** equals -1. On exit, if **n4** equals 0, Card |**ic**| is not in the table; if **n4** < 0, a format error was detected by **inp** on Card |**ic**|; and if **n4**>0, there are **n4** data fields (excluding the card number) on Card |**ic**| and the data are stored sequentially beginning at **xl1**(**n3**). The use flag is set on the table entry for |**ic**| if it is found and **ic** > 0, i.e., the use flag is not set for negative values of **ic**. **link** issues no error messages.

2.8 call moder (xl1, l3, n3, n4, n5, n6)

Subroutine **moder** checks **n4** items of data stored sequentially beginning at XL1(**n3**) for appropriate format. The format specification begins at **l3**(7). The specification starting at 7 is

consistent with the format specification for **inp2**. The format entries are -1 for alphanumeric, 0 for integer, and 1 for floating point. Cyclic repetition for two or more entries can be condensed by prefixing the repeated format by ±N, where the magnitude is the number of items repeated, and the sign is positive if the cycle is to be reset for each entry, and the sign is negative if it is to pick up at the stopping point of the previous entry. To allow starting within the format specification, **n6** is the number of items previously checked with the current format. On exit, **n5** = 0 if no errors were found; 0 < **n5** < 10000 if item **n5** should have been an integer but was not; **n5** < 0 if item **-n5** should have been a floating point number, but was not; and **n5** > 10000 if item **n5** - 10000 should have been alphanumeric, but was not. A decimal zero is considered either integer or floating point as required to satisfy mode tests. **moder** issues no error message.

2.9 function inp8 (nprint, xl1)

Function **inp8** returns the number of cards that have not been processed by **link**. This is done by counting the number of table entries in **xl1** that have positive values for the card number. If **nprint** is 1, the card numbers of the unprocessed cards are listed under Error Message 21, while if **nprint** is 0, no output is printed.

2.10 function inp9 (xl1) and function inp10 (xl1, ic1, ic2)

Functions **inp9** and **inp10** delete table entries and associated data and mode information from the array **xl1**. **inp9** deletes the cards that have been referenced at least once by **link**; that is, table entries with the use bit set on. Function **inp10** deletes all cards, **ic1** $\leq$ c $\leq$ **ic2**. When cards are deleted, all holes created by deletion are squeezed out, the table entries are adjusted accordingly, and the control word is updated. Both functions return the new length of the table, list, and control word. When all cards have been deleted, the length required in the **xl1** array is two, the length required for the control words.

When a card is deleted, a hole in the table and list is created, and the remaining tables and data must be moved to regain the storage made available. In order to move the remaining table and list only once and not use storage outside the array **xl1**, the holes are marked with a bit pattern that has all bits set except the highest. After all cards are deleted, holes are located by testing for the special bit pattern. Thus, the bit pattern used for marking holes must not be allowed as a data item. The bit pattern corresponds to a floating point number that should never be used. Subroutine **inp** checks for this bit pattern as input data is processed. If the bit pattern is found, a warning message is written and the lowest bit is set to zero.

2.11 function ncards (start, end, incr, cards)

Function **ncards** uses subroutine **link** to search the **inp** card table located in **real** array **cards** for unique card numbers that are within the range of card numbers given by **start** and **end**, where both are integer values. On exit, **ncards** returns the number of unique card numbers found.

incr is an integer that can be used to eliminate a portion of the card number from the search by applying the following

cardno = **incr***(**cardno**/**incr**) for **incr** > 0

to card numbers located in the table. Function **ncards** only counts unique occurrences of **cardno**. **incr** should be 1, 10, 100, etc.

Whereas calls to most **inpx** subroutines result in the card use flag being set when a card is found and processed , the use flags are not set when cards are processed by function **ncards**.

As an example use of function **ncards**, consider that the application program that uses the **inp** card data series given by 05XXXY, where XXX is a unique component number and Y is a sequence number that allows use of several cards to define the input for a given component XXX, e.g., different values of Y. If the programmer wants to determine the number or unique components defined in such an input deck, function **ncards** can be used by setting the following values,

start = 050010
end = 059999
incr = 10

With this input, function **ncards** will only count the first occurrence of a **cardno** . If the card deck included the following cards

050031 ‘data follows'
050032 ‘data follows'
051000 ‘data follows'
051501 ‘data follows'
051503 ‘data follows'

function **ncards** will return a value of 3 for **cardno**s 050030, 051000, and 051500. If **incr** is defined as 0 in the example, function **ncards** will return a value of 5 since it will count each of the 05XXXY cards in the card deck.

2.12 function nitems (start, end, cards)

Function **nitems** is part of the **inp** free form input processing package. It uses subroutine **link** to search through **real** array **cards** within the range of card numbers given by **start** and **end** where both are integer values, and counts number of data items on those cards. On exit, **nitems** returns total number of data items.

3.0 LOW LEVEL inp SUBROUTINES

The subroutines described in the following sections are not called directly by users, but rather perform specific functions for the other **inp** subroutines.

3.1 call cvi (char, binary, cond, num, ipos)

This subroutine converts a data card from character format to the appropriate binary form. The arguments are defined by

char Character array containing card to be converted in A80 format

binary Array containing **num** converted items

cond Array containing code for converted items
0 a zero result (integer or floating)
1 integer conversion
2 floating conversion
3 character conversion

num Number of converted items

ipos 0 means no error
>0 indicates column position of error

3.2 call conv (a, xnum, type, lstart, lend, err)

Subroutine **conv** converts numeric data in character format to integer or floating point format. **a** is a character array containing the character data to be converted. **xnum** contains the converted floating point or integer number on output. **type** is a real variable which is set by **conv** to 0 if **xnum** is an integer or floating point 0, 1 if **xnum** is integer, or 2 if **xnum** is floating point. **a**(**lstart**) is the element of **a** where the numeric data begins. **a(lend)** is the element of A where the numeric data ends. **lend** is set by the subroutine **conv**. **err** is a logical error flag. **true** implies that **conv** detected an error, while **false** implies it did not.

3.3 call holstr (a, l1, wrk, cond, nwrds, err)

Subroutine **holstr** converts Hollerith data read from a card to binary form. A is a character array containing the string to be converted. **l1**, on input, points to the beginning of the Hollerith string, on output it points to the end of the string. **wrk** is the real array that the Hollerith data is read into, the format A8 is used. **cond** is the array containing the code for converted items. For every eight characters read in the Hollerith string, one entry in **cond** will be set to three. NWRDS is the number of words in the **wrk** array that the Hollerith string occupies. **err** is a logical flag. If no errors are detected, **err** is set to **false**, otherwise it is set to **true**.

4.0 inp SUMMARY

The following sections contain a summary of the **inp** package subroutine and function calls, a summary of parameters, a list of error messages, and structures of control words, table words, and mode words.

4.1 Summary of inp Package Calls

```
call inp (xl1, nl1, title*, ncase*, ndata*, isw*)
call inp2 (xl1, xl2, xl3)
      with l3(1) = ic1, l3(2) = ±ic2, l3(3) = min, l3(4) = max, l3(5) = nj,
      l3(6) = j*, l3(7) and on equivalent to l5(1) and on.
call inp4 (ic1, ±ic2, min, max, nj, j*, ic3, ntimes, newj, xl1, xl2, l5)
call inp5 (ic1, ±ic2, ic3, ±n1, ±nmax, ±nstore, ntimes, newj, j*, xl1, xl2,
      l5, xl6, nl6)
call link (ic, ix, n3, n4, xl1)
call moder (xl1, l3, n3, n4, n5, n6)
call inp6 (ic1, ic2, n2, icard, item, xl1)
call inp7 (icard, item)
function inp8 (nprint, xl1)
function inp9 (xl1)
function inp10 (xl1, ic1, ic2)
function ncards (start, end, incr, cards)
function nitems (start, end, cards)
call cvi (bcd, bin, icond, num, nch)
call conv (a, xnum, type, lstart, lend, err)
call holstr (a, l1, wrk, cond, nwrds, err)
```

4.2 Array Summary

l3 Array used for specifications to **inp2**.

l5 Equivalence to **l3**(7). Array used to define appropriate mode of data fields on card or set of cards excluding the card numbers. An entry of -1 is for alphanumeric fields, an entry of 0 is for integer fields, and an entry of 1 is for floating point fields. If a format repeats beyond a point, prefix the repeated format in **l5** by ±N, N ≥ 2, where N is the number of items repeated. Use N positive to reset the repeat cycle at the beginning of each card, or use N negative to allow the cycle to overlap cards. Within a cycle, all elements must be 0 or ±1, and only one cycle is allowed. Size of **l5** array is 40. Array for **inp2** starting at **l3**(7) is not limited.

xl1 Array containing control word, converted data from cards, mode indicators, and table entries.

xl2 Array into which data is to be moved.

xl6 Array for temporary storage used in **inp5**.

4.3 Variable Summary

bcd Array containing card to be converted in A80 format.

bin Array containing **num** converted items.

ic Card number desired.

icond Array containing code for converted items
0 a zero result (integer or floating point)
1 integer conversion
2 floating conversion
3 hex conversion
4 octal conversion
-(character count) Hollerith conversion. Uses (|**icond**| + 7)/8 words in **bin**

ic1, ±**ic2** Define card numbers of a set, **ic1** $\leq$ c $\leq$ |**ic2**|. If **ic2** is zero, only one card, **ic1**, is requested. If **ic2** > 0, card numbers must be sequential, c = **ic1**, **ic1** + 1, . . ., **ic1** + a $\leq$ **ic2**, and if **ic1** + a is the last card in sequence, the next larger card c_x must not be in the range **ic1** + a + 2 $\leq$ c $\leq$ **ic2**. If **ic2** is negative, cards need not be sequential and are taken in increasing order. As used for **inp**10, all cards c, **ic1** $\leq$ c $\leq$ **ic2** are deleted.

ic3 Added to **ic1** and **ic2** to define next set for use in **inp4** and **inp5**.

icard Number of card containing error item **n2**.

inp8 Result of function call is the number of cards in table not processed by **link**, **inp2**, **inp4**, or **inp5**.

item Item number on **card** of item **n2** on set of cards in error.

itms List of items to pack (unpack).

ix Card number in table next larger than **ic**. If no such card is present **ix** is returned as -1.

j On entry, if **j** is positive, store data beginning at **xl2**(**j**); if **j** is negative, do not move data into **xl2** but check data. On exit, **j** is set to -1 if an error was found; if positive on entry and no errors were found, it is set to +**nmove**; if negative on entry and no errors were found, it is set to -**nmove**.

max Maximum number of data items in a set of cards.

min Minimum number of data items in a set of cards.

±**n1** **n1** = number of elements in the input vector S; data is sequential type if **n1** is positive and overlay type if **n1** is negative.

n2 The number of the data field in error in call to **inp6**.

n3 On return from **link**, **xl1**(**n3**) contains first data word if card is present.

n4 Number of data words on Card **ic** located by **link** if **n4** is positive; if **n4** is zero, Card **ic** is not in table; if **n4** is negative, format error was found on Card **ic**.

n5 On exit from **moder**, **n5** = 0 if format correct; 0 < **n5** <10000 if item **n5** should have been integer but was not; **n5** < 0 if item -**n5** should have been floating point, but was not; and **n5** > 10000 if item **n5**-10000 should have been alphanumeric but was not.

n6 On entry to **moder**, the number of previously checked items. This is used to located proper starting position in **l5**.

ncase On entry, previous case number which should be non-negative and zero if the first case. On exit, it is the current case number, which is negative if the last case of a problem set.

nch 0 means no error
>0 gives column position of error.

ndata On entry, **ndata** > 0 calls for adding data to the previous list and table in **xl1**, and **ndata** $\leq$ 0 indicates no previous list and table is present and the new data is complete in itself.

newj Added to **j** for subsequent set in **inp4** and **inp**5.

nj Number of words to skip in **xl2** between items. **inp2** stores data item n in **xl2** (**j** + (**nj**+1)*(N-1)). Usually **nj** is zero.

nl1 Size of **xl1** on entry to **inp**.

nl6 Size of XL**6**; must be large enough to hold data from one set in call to **inp5**.

±**nmax** Upper limit for sequential and overlay data; if **nmax** is positive limit must be included.

±**nmin** If sequential type, **nmin** = n_o; if overlay type, min $m_k \geq$ **nmin**, with **nmin** positive requiring that the lower limit be included.

noitms Number of items to pack (unpack).

nprint If 1, list unprocessed card number, if zero, do not list them.

±**nstore** Used to control storage of data in **inp5**. See **inp5** description (Section 2.2.4)

ntimes Number of sets of card to process in call to **inp4** and **inp**5.

num Number of converted items.

word Packed table information.

4.4 Error Message Summary

All error comments are preceded by eight asterisks to facilitate recognition of error comments in the midst of regular program output and are presented in Table V.2-1. The lower case letters represent call parameters or symbols used in the subroutine descriptions and actual values are substituted in the output.

4.5 Control Word Structure

The control words stored in **xl1**(1) and **xl1**(2) consist of two integer words. Integer Word 1, stored in **xl1**(1), is the length of the list containing the binary data and mode indicators; Integer Word 2, stored in **xl1**(2), contains the length of the table. Note that the **xl1** array is real and must be equivalenced to a properly padded integer to retrieve or store the integer contents.

4.6 Table Entries

Three integer words are used to describe each card in the table. Word 1 is the card number, Word 2 is the index in **xl1** of the first data word associated with this table entry, and Word 3 is the number of words on the card, excluding the card number. Word 3 also functions as a card use indicator, the value in Word 3 is set negative when the card is used. Each of these integer words occupies one array entry in the **xl1** array.

Card numbers are limited to 536,870,911 (2^{29}-1) and card numbers greater than this cause Error Message 6 to be printed.

4.7 Mode Indicator Word Structure

For each card (continuation cards are considered as part of the first card), the mode indicator words are stored immediately following the last data word. The mode indicators are 0 for an integer or floating point zero, 1 for a nonzero integer, 2 for a nonzero floating point quantity, and 3 for an alphanumeric quantity.

Table V.2-1

inp User Error Summary

Error Number	Description
1	Insufficient storage allocation for previous data, processing terminated.
2	Insufficient storage for data, processing terminated.
3	$ (placed under column in error) $ points to card error at col. i
4	End of file encountered before end(.) card.
5	Continuation card indicated, but no previous data card. Treated as new data card.
6	Unrecognizable card number
7	Word n5 on card ic should be in alphanumeric format
8	Card c+a+1 missing in sequence
9	Too few numbers on cards ic1 through ic2
10	Too many numbers on cards ic1 through ic2
11	Word n5 on card ic should be in integer format
12	Word n5 on card ic should be in floating point format
13	Cards ic1 through ic2 missing
14	Illegal format on card ic
15	m numbers on card ic1 through ic2 are not a multiple of nl
16	Item m on card ic is less than minimum allowed of nmin
17	Item m on card ic exceeds maximum allowed for nmax
18	Error in limits of the set beginning at item m on card ic
19	Lower limit of nmin not included on cards ic1 through ic2
20	Upper limit of nmax not included on cards ic1 through ic2
21	The following cards were not used
22	Item item on card card in error

inp *Free Form Input*

VI

MAINTENANCE AND INSTALLATION

RETRAN-3D is operational on Personal Computers, PCs, running Windows or on Linux workstations. The HP, IBM, and Sun UNIX workstations are no longer formally supported for RETRAN-3D. However, the existing installation procedures for these platforms are still included in the transmittal. Two different formats are used for transmitting the RETRAN-3D computer program, one that only contains executable code and other files required for the Windows installation. The second transmittal format contains the source code and procedures required to install the program from the source code on Linux platforms. This source code transmittal also includes the Windows installation. These transmittal packages will be referred to as

- the Windows transmittal, and
- the Linux source code transmittal.

Both transmittal packages will include the following:

- instructions and procedures to install RETRAN-3D (either prebuilt executable versions or, versions installed from source code),
- instructions and procedures to execute RETRAN-3D,
- the RETRAN-3D sample problem input files,
- the RETRAN-3D sample problem output listings, and
- baseline date used to validate the code installation using the COMPARE program (see Appendix A), and
- support programs used to validate the code and provide cross section conversion.

Source transmittal packages will also include the RETRAN-3D source code.

The Linux source format transmittal is for the most part generic. An executable program and sample problem output files and compare standard data files are the only platform-specific information included with the transmittal. Operating system and hardware requirements are included in the computer code abstract that is provided with distribution packages. The procedures for installing the code are written with options for the supported platforms. They provide an excellent starting point for installations on other new platforms.

PROPRIETARY INFORMATION - WITHHOLD UNDER 10 CFR 2.390

The Linux source code transmittal can be used to install the code on most Linux and Windows platforms using a Fortran 95 compiler and related linker.

The RETRAN-3D engineering contractor that performs maintenance and development activities for RETRAN-3D uses a version control program to track code modifications. Code users do not need to use the same code maintenance procedures and source code version control program as the engineering contractor. A user organization can either use the transmitted source code with any version control program of their choice, or they can use the source directly.

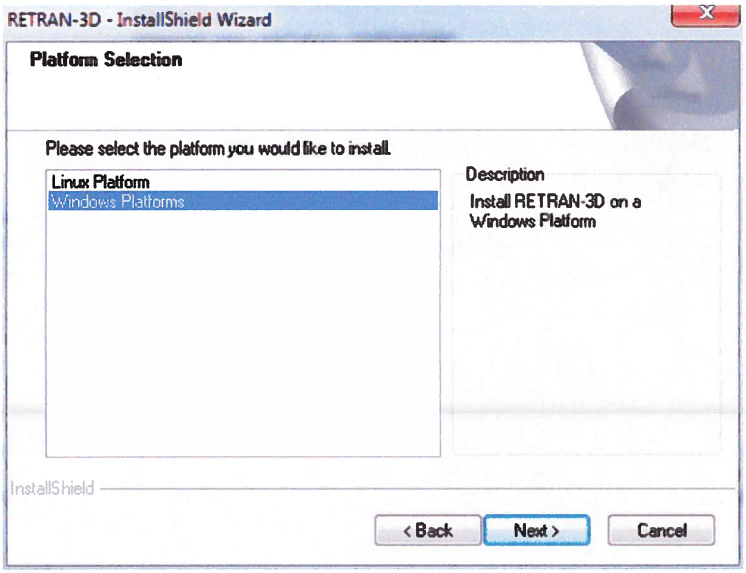
When source code is included in a RETRAN-3D transmittal, the engineering contractor will extracted it from the source code archive using their version control program. The resulting source will contain all code needed to install RETRAN-3D using a Fortran 95 compiler and associated tools. Care has been taken to utilize standard Fortran 95 programming to facilitate transporting the code to new platforms not formally supported.

There are no database or third party software requirements to install RETRAN-3D.

Section 1.0 describes the code installation using the Linux source code transmittal format, while Section 2.0 describes the Windows installation format. Both sections describe the procedures used to test the installation.

1.0 LINUX SOURCE CODE TRANSMITTAL

The Linux source code transmittal includes installation instructions and various output files for supported platforms. It also includes an executable file and necessary support programs for testing and executing some program options, e.g., 3-D kinetics. The transmittal CD-ROM is written using a format compatible with Windows machines. When a transmittal CD-ROM is inserted into a Windows machine, an **autorun** procedure is initiated. It will request the user to review and accept the license agreement and then displays a selection window (shown below) that allows the intallation target platform to be selected.



After a selection is made, the platform-specific InstallShield unload procedure will be executed. It will request the user to select the location where the files are to be unloaded. If these files are not directly accessible to the target platform via a file sharing mechanism, they will need to be moved or copied to the target platform using ftp, ssh, or equivalent utility program. Additional installation instructions will be included on the transmittal in the readme.txt file.

The install.sh script will build the platform-specific executable file from source code. If a source code installation is not desired, the executable unloaded from the transmittal can be used. It should be tested using the procedures described in Section VI.1.2

1.1 Linux Source Code Installation

The discussion of the installation procedures for the RETRAN-3D code are based on the assumption that the RETRAN-3D source program and the installation procedures have been copied from the transmittal media to disk using the appropriate InstallShield unload procedure described above. These files will be used to create a RETRAN-3D executable program module.

Installation of RETRAN-3D on a Linux platform requires approximately 120 Mbytes of free disk space; unloading the CD-ROM requires approximately 75 Mbytes of disk space, building the code requires 25 Mbytes, and running the sample problems an additional 20 Mbytes.

The minimum RAM configuration for executing RETRAN-3D is 128Kbytes, but a minimum of 512Kb is recommended. For large multidimensional kinetics problems, even more RAM (1 to 2 Mbytes) may improve run times since smaller memory configurations may lead to disk swapping, which can increase run times. Minimal swapping should occur if the recommended RAM is available or exceeded. Output file created by RETRAN-3D can be quite large. For this reason it is recommended that users have at least 2GB of free disk space and much more when multiple production analyses are being performed.

The installation and testing procedures for RETRAN-3D on Linux platforms utilizes several Bourne shell scripts. The primary installation script is named **install.sh**. It changes the attributes of the script files and installation directories to read/write/execute. The **install.sh** script then asks for an organization name, up to 32-characters in length, to be entered. This name is used to modify the default organization name in subroutine **set_configuration**, which is contained in module **retran_configuration** (file m_retran_configuration.f90). The **organization** field is included in the page header written to the standard print file and identifies the licensed company or organization name. The default definition is ' ** Unlicensed Trial Version ** '. If the default **organization** name is not revised in the source code, it can be overridden when RETRAN-3D is executed by supplying the **organization** name in the **retran.cfg** file (see Section IV.2.11). Platform-specific executables for the **compare2** and **bxftool** programs are unloaded by the InstallShield unload procedure. If these executables are not copied or are removed prior to running the **install.sh** script, new executables will be created by compiling the associated source code.

Finally, the **install.sh** script executes the **bld** script to compile the source code and link the resulting object files to create an executable file. The **bld** script is shown in Table VI.1-1. It requires three input parameters, one of which defines the target platform, e.g., AIX, HPUX,

Table VI.1-1

Linux Installation and Maintenance Script: bld

```
#=====================================================================
#  bld -- shell script
#=====================================================================
#
#  This script file is used to build an executable file for various
#  platforms.   It defaults to the Fortran 95 compiler for the specified
#  platform.  If the -f parameter is provided, the Fortran 77 compiler
#  (or equivalent) will be used.  Fortran 77 source files must have a .f
#  suffix, while Fortran 95 files must have a .f90 suffix.
#
#  When the -B option is used, a full build is performed using the source
#  files listed in file compile_list.  File compile_list also determines
#  compile dependencies.
#
#  When the -B parameter is not provided, an update is made using all
#  source files in the source  directory are compiled.  The resulting
#  object files replace any corresponding object files in the object
#  file directory which contains object files from a previous build.
#  They are linked to create a new executable.  This feature is used
#  to debug or patch a previous build.
#
#=====================================================================
#
#  DEFINE FILE SPECS
#
exec=progx
lognam=bld.log
PLATFORM=DUMMY
fext=f90
incdir=.
moddir=.
objdir=.
srcdir=.
build=0
nolink=0
#
#
if [ $# = 0 ]
 then
     echo "                                                              "
     echo " bld **** ERROR **** no arguments specified                   "
     echo "                                                              "
     echo " The bld script is use to perform either a complete build or an    "
     echo " update of a program that was built previously.  A complete build   "
     echo " uses the compile_list to provide the path and order that           "
     echo " subroutines will be compiled.  The path will be relative to        "
     echo " either the current directory or the one specified using the -S     "
     echo " option flag.  Update jobs  compile any source routines in the      "
     echo " current or source directory  and links with the object code in the "
     echo " the current or -O directory.                                       "
     echo "                                                                    "
     echo "              -B       = complete build (otherwise update)         "
```

Table VI.1-1 (Cont'd)

```
      echo "               -c       = compile only (no link)                    "
      echo "               -f       = use f77 compiler - otherwise use f95      "
      echo "               -I name  = include/module directory name (read)      "
      echo "               -M name  = module directory name (write)             "
      echo "               -O name  = object directory name  (write/read)       "
      echo "               -P name  = platform name                             "
      echo "                        = AIX                                       "
      echo "                        = HPUX                                      "
      echo "                        = SOLARIS                                   "
      echo "                        = Linux-Ifort      (32-bit)                 "
      echo "                        = Linux-Ifort64    (64-bit)                 "
      echo "                        = Linux-gnu                                 "
      echo "                        = Linux-g95                                 "
      echo "               -S name  = source directory name                     "
      echo "               -x name  = name of new executable                    "
      echo "                                                                    "
      exit 1
#
else
      timdat=`date`
      echo "                                                 "
      echo " Running bld shell script .................   "$timdat
#
      set -- `getopt I:M:O:P:S:x:cBf $*`
#
      if [ $? != 0 ]
       then
           exit 99
      fi
#
      while [ $1 != -- ]
       do
         case $1 in
         -B)
           build=1
           shift
           ;;
         -c)
           nolink=1
           shift
           ;;
         -f)
           fext=f
           shift
           ;;
         -I)
           incdir=$2
           shift; shift
           ;;
         -M)
           moddir=$2
           shift; shift
           ;;
         -O)
```

Table VI.1-1 (Cont'd)

```
          objdir=$2
          shift; shift
          ;;
        -P)
          PLATFORM=$2
          shift; shift
          ;;
        -S)
          srcdir=$2
          shift; shift
          ;;
        -x)
          exec=$2
          shift; shift
          ;;
        esac
     done
     shift
fi
#
rm -f  $lognam
#
     echo " Build executable: "$exec" for "$PLATFORM" platform ....... "
     echo " Build executable: "$exec" for "$PLATFORM" platform ....... "$timdat
>> $lognam
#
if [ $build = 0 ]
 then
     echo " Update using local source files in directory $srcdir/ and object files
in directory $objdir/"
     echo " Update using local source files in directory $srcdir/ and object files
in directory $objdir/" >> $lognam
else
     echo " Full build using source files specified in file compile_list"
     echo " Full build using source files specified in file compile_list" >> $lognam
fi
#
#
#  SET UP FOR SPECIFIC PLATFORM
#
#
if [ $PLATFORM = "AIX" ]
 then
#
      if [ $fext = "f90" ]
       then
         Fort="f90 -C -qsave -I$incdir -M$moddir -bmaxdata:0x80000000 "
         FF=' -c -qmaxmem=10000 '
       else
         Fort="f77 -O -I$incdir -M$moddir -bmaxdata:0x80000000"
         FF=' -c -qmaxmem=3800 -qqcount '
      fi
#
elif [ $PLATFORM = "HPUX" ]
```

Table VI.1-1 (Cont'd)

```
 then
#
#    +save also zeros uninitialized data
#
      if [ $fext = "f90" ]
       then
         Fort="f90 +O2 +save -I$incdir -M$moddir +check=all +fltconst_strict "
         FF='-c '
       else
         Fort="f90 +O2 +save -I$incdir -M$moddir "
         FF='-c '
      fi
#
elif [ $PLATFORM = "SOLARIS" ]
 then
#
#    -C performs run and compile time bounds checking
#    save seems to be the default - cann't option to turn on
#
      if [ $fext = "f90" ]
       then
         Fort="f90 -C -O -I$incdir -M$moddir "
         FF='-c '
       else
         Fort="f90 -f77 -ftrap=%none -O -I$incdir -M$moddir "
         FF='-c '
      fi

#
elif [ $PLATFORM = "Linux-Ifort" ]
 then
      if [ $fext = "f90" ]
       then
         Fort="ifort -save -zero -I $incdir -module $moddir -check bounds  -threads
-traceback -O "
         FF='-c'
       else
         Fort="ifort -save -zero -I $incdir -module $moddir -threads -traceback
-O "
         FF='-c '
      fi
#
elif [ $PLATFORM = "Linux-Ifort64" ]
 then
      if [ $fext = "f90" ]
       then
         Fort="ifort64 -save -zero -I $incdir -module $moddir -check bounds
-threads -traceback -O "
         FF='-c'
       else
         Fort="ifort64 -save -zero -I $incdir -module $moddir -threads -traceback
-O "
         FF='-c '
      fi
```

Table VI.1-1 (Cont'd)

```
#
elif [ $PLATFORM = "Linux-gnu" ]
 then
      if [ $fext = "f90" ]
       then
         Fort="f95 -fno-automatic -fbounds-check  -I$incdir -J$moddir -O "
         FF='-c'
       else
         Fort="f77 -fno-automatic -I$incdir -J$moddir -O "
         FF='-c '
      fi
#
elif [ $PLATFORM = "Linux-g95" ]
 then
      if [ $fext = "f90" ]
       then
         Fort="g95 -fstatic -fbounds-check -fzero -i4 -I$incdir -fmod=$moddir -O
-fsloppy-char -ftrace=full "
         FF='-c'
       else
         Fort="f77 -fstatic -I$incdir -fzero -i4 -fmod=$moddir -O "
         FF='-c '
      fi
else
     echo " "
     echo " **** ERROR ****  invalid platform specified: "$PLATFORM
     echo " "
     exit 12
fi
#
     echo " Compile options used  : "$Fort$FF" " >> $lognam
     echo " Source file directory : "$srcdir"/" >> $lognam
     echo " Include file directory: "$incdir"/" >> $lognam
     echo " Module file directory : "$moddir"/" >> $lognam
     echo " Object file directory : "$objdir"/" >> $lognam
     echo "                       "        >> $lognam
#
#  COMPILE SOURCE
#
#  Note: compile_list is created using Visual Studio build log
#        It is ordered to account for module dependencies
#
echo " Compile source ........................."
ecode=0
#
#  if build = 1 do a complete build from source
#
if [ $objdir != . ]
 then
#
   if [ ! -d $objdir ]
    then
      mkdir $objdir
   fi
```

Table VI.1-1 (Cont'd)

```
fi
#
#
if [ $moddir != . ]
 then
#
   if [ ! -d $moddir ]
    then
      mkdir $moddir
   fi
fi
#
#
if [ $build = 1 ]
  then
    if [ -s "compile_list" ]
      then
        rm -f $objdir/*.o
        rm -f $moddir/*.mod
#
        for rtn in `cat compile_list`
          do
             echo " Compiling subroutine: "$srcdir/$rtn
             echo " Compiling subroutine: "$srcdir/$rtn 1>>$lognam
#
             $Fort $FF $srcdir/$rtn 1>>$lognam 2>>$lognam
             if [ $? -ne 0 ]
              then
                ecode=1
             fi
         done
      else
         echo " "
         echo " **** ERROR ****  file compile_list not in install directory "
         echo " "
         exit 12
    fi
#
#  if build /= 1 do a build using fortran source files and object code in
#  current directory
#
else
        for rtn in `ls -1 *.$fext`
          do
             echo " Compiling subroutine: "./$rtn
             echo " Compiling subroutine: "./$rtn 1>>$lognam
#
             $Fort $FF ./$rtn 1>>$lognam 2>>$lognam
             if [ $? -ne 0 ]
              then
                ecode=1
             fi
         done
fi
```

Table VI.1-1 (Cont'd)

```
#
if [ $objdir != . ]
 then
    mv -f *.o $objdir
fi
#
if [ $ecode = 1 ]
 then
#
     echo " **** ERROR ****  FORTRAN Compilation failed "
     echo " **** ERROR ****  FORTRAN Compilation failed " >> $lognam
     exit 5
fi
#
#  GENERATE EXECUTABLE
#
timdat=`date`
#
if [ $nolink = 0 ]
 then
#
   echo " Link step ................................."
#
   echo " Link options used: "$Fort" -o "$exec" "$objdir"/*.o" >> $lognam
   $Fort -o $exec $objdir/*.o 1>>$lognam 2>>$lognam
   if [ $? != 0 ]
    then
        echo " **** ERROR ****  Link failed "
        exit 9
   fi
 #
   echo " Normal end of build ......................   "$timdat
   echo " Normal end of build ......................   "$timdat >> $lognam
#
 else
   echo " Normal end of compile only ................   "$timdat
   echo " Normal end of compile only ................   "$timdat >> $lognam
#
fi
#
exit 0
```

SOLARIS, Linux-Ifort or Linux-Ifort64. The platform flag is used to define the compiler-specific name and option flags.

Three parameters are required when running the **bld** script to build a RETRAN-3D executable file, they are **–P, –B,** and **–x**. The **–P** parameter is used to identify the platform the build is being run on (also the platform an executable is being created for) and requires that a valid platform identifier be supplied with the parameter. If **bld** is entered without any parameters, an error is triggered and the available options are written to the console without further action. This allows the script to be self-documenting. The text identifiers for UNIX platforms not formally supported include AIX, HPUX, or SOLARIS (must be upper case). Several different compiler options are available for use on Linux platforms. They are Linux-Ifort and Linux-Ifort64 for the Intel 14.0.2 Fortran compiler [V.1-4] for 32-bit and 64-bit applications, respectively, and Linux-gnu or Linux-g95 for the GNU or G95 Fortran compliers.[V.1-5, V.1-6] The Intel complier is used to prepare the Linux executable included with the code distribution. The gnu and G95 options are provided as a starting point for those who may wish to use these compilers, but they have not been fully tested with RETRAN-3D.

The **-B** parameter specifies that a file named **compile_list** will be used to define the order in which subroutines are compiled. The compilation order is significant because of the use of modules, which define constants and data definitions used during compilation of other modules and subroutines that refer to them. The required compilation order is determined using Microsoft Visual Studio (VS) on a Windows PC where the code is formally maintained by the maintenance contractor. VS analyzes the source code and determines the order subroutines must be compiled to satisfy the associated dependencies. During a full VS project build, a build log is created that contains the order that subroutines are compiled. This log file is used to create the **compile_list** file that is used by the **bld** script to define files that are compiled and the order they are compiled. Table VI.1-2 is a fragment of the **compile_list** file that illustrates its content and format. A complete and current **compile_list** will be included on the transmittal media.

The third parameter used with the **bld** script during a full build is **–x**. It is used to provide the name of the executable file that will be created. The name of the executable is supplied following the **–x** parameter. The executable file will be created in the current directory and the object files will also be created in the current or installation directory. An example command line for building a RETRAN-3D executable on a Linux platform using the Intel 14.0.2 Fortran compiler follows.

```
bld –P Linux-Intel –B –x r3dm4p6x
```

An optional parameter **–O**, can be used to specify the name of the directory where the object files will be placed. This will place all of the object files in this directory rather than the installation directory. The executable will still be created in the installation directory.

An optional parameter **-S**, can be used to specify the path name of the directory where the source code files reside if they are not in the installation directory. This is used in conjunction with the **compile_list** file which may contain path information, either relative or absolute. If the **compile_list** file entries contain relative path information, it will be combined with the path name given with the **-S** parameter. If the absolute path is given in the **compile_list** file, the optional **-S** parameter should not be used.

The **–f** parameter is not used with RETRAN-3D because it invokes the Fortran 77 or equivalent compiler options. This option is used with other programs that require use of the Fortran 77 compiler. An option is also available to define the **INCLUDE** directory, **-I**. This option is also used with other code versions since the Fortran 95 version of RETRAN-3D does not use **INCLUDE** files. A similar option is available for module (.mod) files created by the compiler. By default they will reside in the installation directory. The **-M** parameter can be used to identify where the module files will be located.

Table VI.1-2

Example Subroutine Compilation Order File: compile_list

```
./modules/m_kind_specs.f90
./source/dummies_1d3d.f90
./source/nitems.f90
./source/inp7.f90
./source/getpar.f90
./source/ncards.f90
./source/date8.f90
./interfaces/s_isygn.f90
./modules/m_retran_configuration.f90
./interfaces/s_rken.f90
./interfaces/s_condht.f90
./source/inp_spec.f90
./modules/s_l_pntr.f90
./modules/m_ncgasc.f90
./interfaces/s_ghem.f90
./interfaces/s_locf.f90
./modules/m_arrcom.f90
./interfaces/s_zmoc.f90
./interfaces/s_poly.f90
./interfaces/s_chkv.f90
./modules/s_i_pntr.f90
./modules/m_minor_edits.f90
./interfaces/s_thcon.f90
./interfaces/s_delhp.f90
./interfaces/s_inp8.f90
./interfaces/s_water.f90
./interfaces/s_nitems.f90
./interfaces/s_c3func.f90
./modules/m_minor_edit_search.f90
./modules/m_edit_headers.f90
.
.
.
./source/qdot05.f90
./source/pspec.f90
./source/nterp.f90
./source/me_sum_e.f90
./source/locflx.f90
./source/inp2.f90
./source/ingeom.f90
./source/inckv.f90
./source/ghem.f90
./source/flowmp.f90
./source/expin2.f90
./source/edit_steady_state.f90
./source/delpin.f90
./source/contrl.f90
./source/cardbc.f90
```

Depending on the speed of the machine being used for the installation, the wall time required to install RETRAN-3D can range from several minutes to 20 minutes. The build log file named bld.log is created during the installation process and will contain the Linux standard out and error results for the compile and link steps of the installation. It can be viewed during installation to determine the progress. The log file will include the completion status of the various installation steps.

After an executable has been created, the installation should be tested by performing the testing described in the following section.

1.2 Linux Platform Testing

One of the most important aspects of installing RETRAN-3D is testing the installed code to insure that the results generated for a series of test cases agree with those for a baseline installation. This is facilitated by use of the COMPARE2 program discussed in Appendix A. It compares a selected set of output for each sample problem, with baseline data. Baseline data may be included on the transmittal for each supported platform, or for a single reference platform. The COMPARE2 program creates a report that contains all occurrences where the results from the new installation differ from the baseline standard. User supplied tolerances to the COMPARE2 program determine when differences are reported.

The installation script installs the COMPARE2 program. It is run separately following the installation, by running a **checkin.sh** script. This step is discussed in the following sections.

Once an executable file is either installed using the **bld** script or copied from the transmittal media as part of the unload process as described above, it can be used to run RETRAN-3D. A script file, **run.sh**, is shown in Table VI.1-3. It is used to execute RETRAN-3D and is copied to the root installation directory by the unload script. It is used to execute RETRAN-3D and perform the necessary file set up prior to executing RETRAN-3D and upon completion of the run. Running the run script without any arguments will produce a list of the possible options. The **-x** switch defines the executable file to use. Alternately, the variable exec can be set within the run script to the desired executable file. As transmitted, run will use the RETRAN-3D executable included on the transmittal (retran3d.x), unless an executable is generated by the **bld** script (r3dm004p7).

The **run.sh** script can be used to execute RETRAN-3D at this point in the installation. However, it is suggested that the installation first be tested using the **checkin** script shown in Table VI.1-4. It runs the standard sample problems, which are set up to create auxiliary files (TAPE60) containing the minor edit output. These files for the sample problem set are then copied to a single file, TSTDTA, that is then compared against the STDDTA baseline file. The **checkin.sh** script is used to run the sample problem set, create the TSTDTA file, and run the COMPARE2 program. The **checkin.sh** script by default, assumes that all sample problem are to be executed including the three-dimensional kinetics sample problems. If just the non-multidimensional kinetics problems are to be run, the variable r3dkin in the **checkin.sh** script should be set to "no".

The **checkin.sh** script is executed by entering

```
./checkin.sh
```

Table VI.1-3

Linux Execution Script: run.sh

```
#! /bin/sh
#
# HP
# dbgr=gdb
#
# SUN, IBM
# dbgr=dbx
#
dbgr=
rdeck=
tape12=
tp12ed=
tape13=
tp13ed=
xsecin=
xseced=
inpext=
debug=
opts=
nsm=
RTAPE=
XTAPE=
dattim=
xstat=
rm -f _headr
#
#
if [ $# = 0 ]
 then
  echo "                                                    "
  echo " **** ERROR ****  no arguments specified"
  echo "                                                  "
  echo "            -x  name  = executable name"
  echo "            -r  name  = retran input"
  echo "            -X  name  = 1D X-section file (TAPE40)"
  echo "            -3  name  = 3D X-section file (TAPE68)"
  echo "            -B  name  = old RETRAN data file (TAPE12)"
  echo "            -R  name  = old RETRAN restart file (TAPE13)"
  echo "            -C  name  = old CDI file (TAPE78)"
  echo "            -ns       = no screen messages"
  echo "            -d        = run debugger"
  echo "                                                    "
  exit 1
#
elif [ $# = 1 ]
    then
     rdeck=$1
else
#
set -- `getopt x:r:B:C:R:3:X:n:d $*`
#
```

Table VI.1-3 (Cont'd)

```
  if [ $? != 0 ]
   then
      exit 99
  fi
#
    while [ "Q$1" != "Q--" ]
    do
       case $1 in
        -x)
          exec=$2
          shift; shift
          ;;
        -r)
          rdeck=$2
          shift; shift
          ;;
        -B)
          tape12=$2
          tp12ed="with TAPE12: "$2
          shift; shift
          ;;
        -R)
          tape13=$2
          tp13ed="with TAPE13: "$2
          RTAPE="TAPE13"
          shift; shift
          ;;
        -X)
          xsecin=$2
          xseced="with TAPE40: "$2
          XTAPE="TAPE40"
          shift; shift
          ;;
        -3)
          xsecin=$2
          xseced="with TAPE68: "$2
          XTAPE="TAPE68"
          shift; shift
          ;;
        -C)
          cdiin=$2
          cdied="with CDI file : "$2
          CTAPE="TAPE78"
          shift; shift
          ;;
        -n)
          opts=$opts" -ns"
          nsm="no"
          shift; shift
          ;;
        -d)
          debug="yes"
          shift
          ;;
```

Table VI.1-3 (Cont'd)

```
         *)
          echo " Bad Option on Command Line"
          exit 99
          ;;
       esac
    done
shift
fi
#
#  RUN EXECUTABLE
#
#
execc=`type $exec|awk '{print $3}'`
if [ -x "$execc" ]
 then
    if [ -n "$xsecin" ]
     then
       if [ -s "$xsecin" ]
        then
          rm -f $XTAPE
          ln -s $xsecin $XTAPE
       else
          echo " **** ERROR **** Specified "$XTAPE" does not exist :"$xsecin
          exit 40
       fi
    fi
    if [ -n "$cdiin" ]
     then
       if [ -s "$cdiin" ]
        then
          rm -f $CTAPE
          ln -s $cdiin $CTAPE
       else
          echo " **** ERROR **** Specified "$CTAPE" does not exist :"$cdiin
          exit 41
       fi
    fi
    if [ -n "$tcsin" ]
     then
       if [ -s "$tcsin" ]
        then
          rm -f $HTAPE
          ln -s $tcsin $HTAPE
       else
          echo " **** ERROR **** Specified "$HTAPE" does not exist :"$tcsin
          exit 42
       fi
    fi
    if [ -n "$tape12" ]
     then
       if [ -s "$tape12" ]
        then
          rm -f TAPE12
          ln -s $tape12 TAPE12
```

Table VI.1-3 (Cont'd)

```
       else
          echo " **** ERROR **** Specified TAPE12 does not exist :"$xsecin
          exit 12
       fi
    fi
    if [ -n "$tape13" ]
     then
       if [ -s "$tape13" ]
        then
          rm -f $RTAPE
          ln -s $tape13 $RTAPE
       else
          echo " **** ERROR **** Specified "$RTAPE" does not exist :"$tape13
          exit 13
       fi
    fi
    if [ -n "$rdeck" -a -s "$rdeck" ]
       then
          rm -f INPUT
          cp $rdeck INPUT
          outfil=OUTPUT
    else
       echo " **** ERROR **** Specified input deck does not exist :"$rdeck
       exit 11
    fi
#
# header information
#
    echo " OS:            "`uname`     > _headr 2>/dev/null
    echo " User Name:     "`whoami`  >> _headr 2>/dev/null
    dattim=`date "+%m/%d/%y %H:%M %p"`
    echo " Start time:    "$dattim >> _headr 2>/dev/null
#
    echo "" >> _headr  2>/dev/null
    echo " Input File Information:"  >> _headr 2>/dev/null
    echo "       Input File: "$rdeck >> _headr 2>/dev/null
    if [ ! -z "$xseced" ];then echo "       "$xseced >> _headr 2>/dev/null ;fi
    if [ ! -z "$cdied"  ];then echo "       "$cdied >> _headr 2>/dev/null  ;fi
    if [ ! -z "$tcsed"  ];then echo "       "$tcsed >> _headr 2>/dev/null  ;fi
    if [ ! -z "$tp13ed" ];then echo "       "$tp13ed >> _headr 2>/dev/null ;fi
    if [ ! -z "$tp12ed" ];then echo "       "$tp12ed >> _headr 2>/dev/null ;fi
    echo "" >> _headr  2>/dev/null
    echo " Command:  "$execc" "$opts >> _headr  2>/dev/null
    echo "" >> _headr  2>/dev/null
    echo " Program Name:    "`basename $execc`"  ("`dirname $execc`")"  >> _headr
2>/dev/null
    echo " Program Stats:    "`ls -l $execc`   >> _headr 2>/dev/null

#
#
#
    if [ -z "$nsm" ]
     then
       echo " Running  "$execc" "$opts"  with INPUT: "$rdeck
```

Table VI.1-3 (Cont'd)

```
       if [ ! -z "$xseced" ];then echo "                  "$xseced ;fi
       if [ ! -z "$cdied"  ];then echo "                  "$cdied  ;fi
       if [ ! -z "$tcsed"  ];then echo "                  "$tcsed  ;fi
       if [ ! -z "$tp13ed" ];then echo "                  "$tp13ed ;fi
       if [ ! -z "$tp12ed" ];then echo "                  "$tp12ed ;fi
       echo "                   "
    fi
#
       rm -f ftn* fort* ERR_LOG
       if [ ! -z "$debug" ]
        then
          $dbgr $exec $opts
          xstat=$?
        else
          $exec $opts
          xstat=$?
       fi

#
#    continue header information
#
    dattim=`date "+%m/%d/%y %H:%M %p"`
    echo "" >> _headr  2>/dev/null
    echo " Execution ended:  "$dattim >> _headr 2>/dev/null
    echo "" >> _headr  2>/dev/null
    echo " Exit status:  "$xstat >> _headr  2>/dev/null
    echo "" >> _headr  2>/dev/null
    echo " Output File Information:" >> _headr 2>/dev/null

       rm -f INPUT TAPE2 TAPE3 TAPE20
       bdeck=`basename $rdeck`
       if [ -s "$outfil" ]
        then
#        mv $outfil $bdeck.out
         echo "        Output File: "$bdeck.out >> _headr  2>/dev/null
#        if [ -s "REMARKS" ]
#         then
#          cat REMARKS >> $bdeck.out
#          rm REMARKS
#        fi
       fi
       if [ -s "ERR_LOG" ]
        then
           mv  ERR_LOG  $bdeck.ERR_LOG
           echo "          Error Log: "$bdeck.ERR_LOG >> _headr  2>/dev/null
       fi
       if [ -s "TAPE14" ]
        then
         mv TAPE14  $bdeck.tape14
         echo "       Restart File: "$bdeck.tape14 >> _headr  2>/dev/null
       fi
       if [ -s "TAPE60" ]
        then
```

Table VI.1-3 (Cont'd)

```
        mv TAPE60  $bdeck.tape60
        echo "        TAPE60 File: "$bdeck.tape60 >> _headr  2>/dev/null
      fi
      if [ -s "VBCFIL" ]
       then
        mv VBCFIL  $bdeck.vbc
        echo "           VBC File: "$bdeck.vbc >> _headr  2>/dev/null
      fi
      if [ -s "TAPE41" ]
       then
        mv TAPE41  $bdeck.tape41
      fi
      if [ -s "TAPE42" ]
       then
        mv TAPE42  $bdeck.tape42
      fi
      if [ -s "TAPE43" ]
       then
        mv TAPE43  $bdeck.tape43
      fi
      if [ -s "R3D_PLOT" ]
       then
        mv R3D_PLOT $bdeck.plt
        echo "          PLOT File: "$bdeck.plt >> _headr  2>/dev/null
      fi
      if [ -s "fort.2" ]
       then
        rm -f fort.2
      fi
      if [ -s "fort.3" ]
       then
        rm -f fort.3
      fi
      if [ -s "ftn02" ]
       then
        rm -f ftn02
      fi
#
#    finish header information
#
     echo "" >> _headr  2>/dev/null
     cat _headr $outfil REMARKS > $bdeck.out 2>/dev/null
     rm -f _headr $outfil REMARKS
#

else
    echo " **** ERROR **** Specified exec file not found :"$exec
    exit 20
fi
#
exit 0
```

Table VI.1-4

Linux Installation Verification Script: checkin.sh

```
#!/bin/sh
#
#  Script to execute all 16 sample problems.
#  After running the sample problems, the TAPE60 files are concatenated
#  to file TSTDTA which is then used as input to the COMPAR program.
#
#  Select the executable file
#
xxx=retran3d.x
#
if [ -x "r3m004p3x" ]
 then
       xxx=r3m004p3x
fi
#
if [ $# = 1 ]
 then
      xxxc=`type $1|awk '{print $3}'`
      if [ -x "$xxxc" ]
       then
             xxx=$1
      else
          echo " **** ERROR **** Specified exec file not found :"$1
          exit
      fi
fi
#
#
#  Run sample problems
#
./run -r sample/sp1 -x $xxx
./run -r sample/accum -x $xxx
./run -r sample/sp5 -x $xxx
./run -r sample/tlta -x $xxx
./run -r sample/ttwob -x $xxx
./run -r sample/ucrw -x $xxx
./run -r sample/fl2d -x $xxx
./run -r sample/turb -x $xxx
./run -r sample/ttqx1 -X sample/ttqx1.t40 -x $xxx
./run -r sample/atws -x $xxx
./run -r sample/pipe -x $xxx
./run -r sample/wovrs -x $xxx
./run -r sample/lrhr -x $xxx
#
./xfgen.sh A sample/pwr
./run -r sample/pwr -3 pwr.bxf -C sample/pwr.cdi -x $xxx -m 5M
./xfgen.sh A sample/slb
./run -r sample/slb -3 slb.bxf -C sample/slb.cdi -x $xxx -m 5M
./xfgen.sh A sample/bwr
./run -r sample/bwr -3 bwr.bxf -C sample/bwr.cdi -x $xxx -m 5M
mv *.bxf sample
#
#  Concatenate the tape60 files to file TSTDTA
#
cat   sp1.tape60  > compare/TSTDTA
cat accum.tape60 >> compare/TSTDTA
cat   sp5.tape60 >> compare/TSTDTA
cat  tlta.tape60 >> compare/TSTDTA
```

Table VI.1-4 (Cont'd)

```
cat ttwob.tape60 >> compare/TSTDTA
cat  ucrw.tape60 >> compare/TSTDTA
cat  fl2d.tape60 >> compare/TSTDTA
cat  turb.tape60 >> compare/TSTDTA
cat ttqx1.tape60 >> compare/TSTDTA
cat  atws.tape60 >> compare/TSTDTA
cat  pipe.tape60 >> compare/TSTDTA
cat wovrs.tape60 >> compare/TSTDTA
cat  lrhr.tape60 >> compare/TSTDTA
cat   pwr.tape60 >> compare/TSTDTA
cat   slb.tape60 >> compare/TSTDTA
cat   bwr.tape60 >> compare/TSTDTA
#
#  Run COMPARE program
#
cd compare
rm -f INDTA REPORT
cp R3D.TXT INDTA
./compare
```

After first running the sample problems, the **checkin.sh** script verifies the installation by comparing results to those supplied in the ./compare/ directory by executing the COMPARE2 program. A summary of these comparisons is written to the REPORT file in the ./compare/ directory. A text editor can be used to examine the REPORT file to determine if the installation was successful. The REPORT file should indicate that there are 0 differences. The cause of any differences should be determined.

When the multidimensional kinetics sample problems are run by **checkin.sh**, cross-section files are required to execute RETRAN-3D. They must be in binary form, but they are provided in ASCII form on the transmittal media and therefore, must be converted to binary form using BXFTOOL as discussed in Appendix B. This will be done by **checkin.sh** if it is executed. If the multidimensional kinetics sample problems are executed using the **run.sh** script prior to running **checkin.sh**, BXFTOOL will have to be run separately (using the **xfgen.sh** script) to convert each of the three cross-section files to binary format.

1.2.1 COMPARE2 and BXFTOOL Installation

The InstallShield unload procedure will generally copy the platform-specific executables for the BXFTOOL and COMPARE2 programs into the appropriate directories in which case this section can be skipped. However, in the event that RETRAN-3D is being installed on a new unsupported platform or if the supplied executables fail to execute properly, it may be necessary to install them from source.

The source code for these programs is copied to disk by the InstallShield unload procedure. They can be installed by using the Fortran 95 compiler by entering the following commands from the keyboard (starting from the installation directory);

```
cd compare
f95 -c compare_data.f90
f95 -c report.f90 compare2.f90
f95 -0 compare2 *.0
cd ../bxftool
f95 bxftool.f -o bxftool
cd ..
```

After successfully compiling bxftool.f and compar.f, the **checkin.sh** script can be used to test the executable. Note that BXFTOOL is used to convert the cross-section files included on the transmittal media to a form used by the multidimensional kinetics option (see Appendix B).

1.3 Installing on Other Platforms

The Linux transmittal will provide a starting point for installing RETRAN-3D on a platform that is not formally supported. It may be necessary to use the Linux InstallShield unload procedures or to do the unload process manually.

The installation instructions given in the previous sections should provide a good starting point for platforms that are not specifically supported. The general organization of installation steps should follow those given in Table VI.1-1, where the appropriate compiler name and option flags are

substituted for the target platform. The **run.sh** and **checkin.sh** scripts given in Tables VI.1-3 and VI.1-4, respectively, will provide a starting point for the new platform.

The sample problem input decks reside in a separate sample problem directory (sample) and the corresponding sample problem output files are located in the platform specific output directory (output) on the transmittal media.

The BXFTOOL and COMPARE programs will have to be installed as instruction in the previous section. The baseline data (STDDTA) and input data (INDTA) used by compare will have to be copied from the transmittal media.

It is recommended to install the code without optimization initially. Once the code is installed and satisfactory comparisons of the sample problem output with the COMPARE baseline data have been made, the code can be recompiled using higher levels of optimization.

2.0 WINDOWS TRANSMITTAL

RETRAN-3D for Windows platforms is only provided in executable form. Disk space requirements are 50 Mbytes, which provides adequate disk space to install and test the code installation.

The minimum RAM configuration for executing RETRAN-3D is 1Mbyte, but a minimum of 2Mbytes is recommended. For large multidimensional kinetics problems, even more RAM may improve run times since smaller memory configurations may lead to disk swapping, which can increase run times. Minimal swapping should occur if the recommended RAM is available or exceeded. Output file created by RETRAN-3D can be quite large. For this reason it is recommended that users have at least 2GB of free disk space and much more when multiple production analyses are being performed.

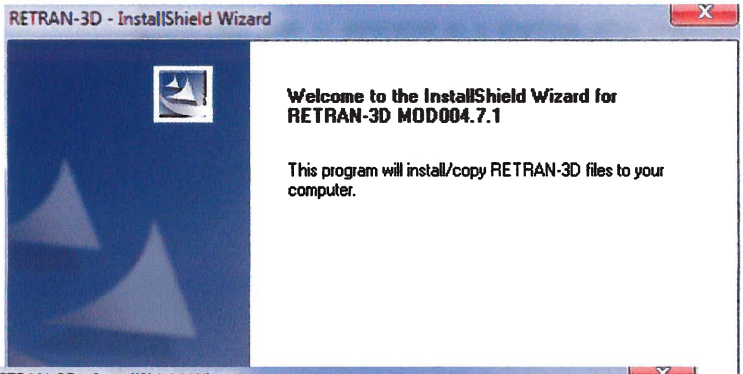
2.1 Windows Installation

The RETRAN-3D installation media contains a typical Windows installation package that will copy all of the needed files to the appropriate locations on your computer system. The RETRAN-3D installation package will also set up the logic that allows you to remove the application (and all of its supporting files) from your computer system, should you choose to do so.

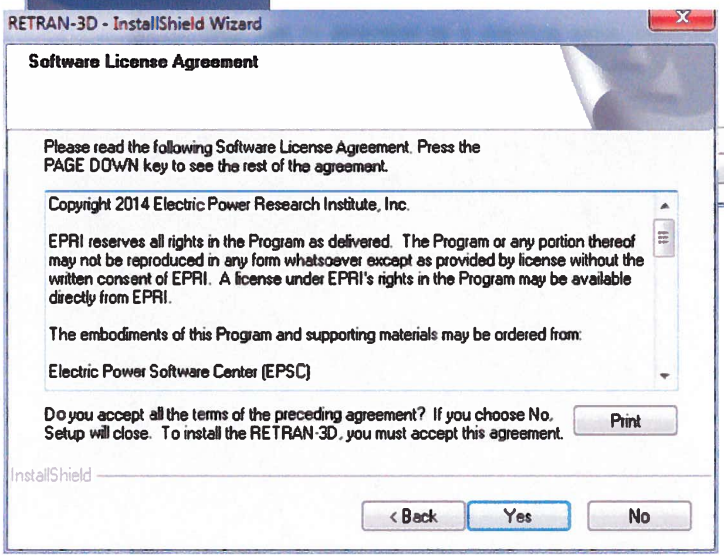
1. Insert the installation media (CD-ROM) in the appropriate drive of your computer system. If the setup does not start automatically, from the **Start** menu select **Run**, then type in the name or browse for the setup program that resides on your CD-Rom drive (**<Drive>:\Platform\Windows\Install\Setup.exe**).

2. After the installation has begun, a **Welcome** window will appear on the screen (as shown below) introducing the installation process. Select the **Next>** button at the bottom of that window to continue the installation process.

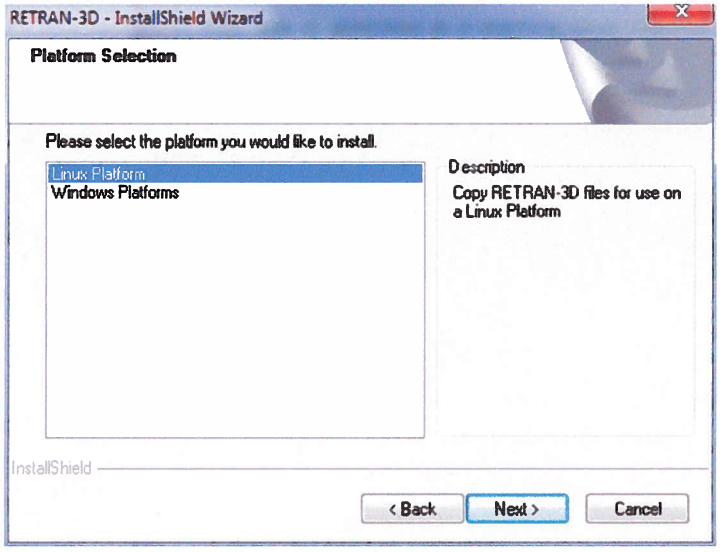
If, at any point, you decide to stop the installation of the RETRAN-3D software, select the **Cancel** button on any of the following windows. If you cancel the installation, no files from the installation will be left on your computer, nor will your computer be changed.



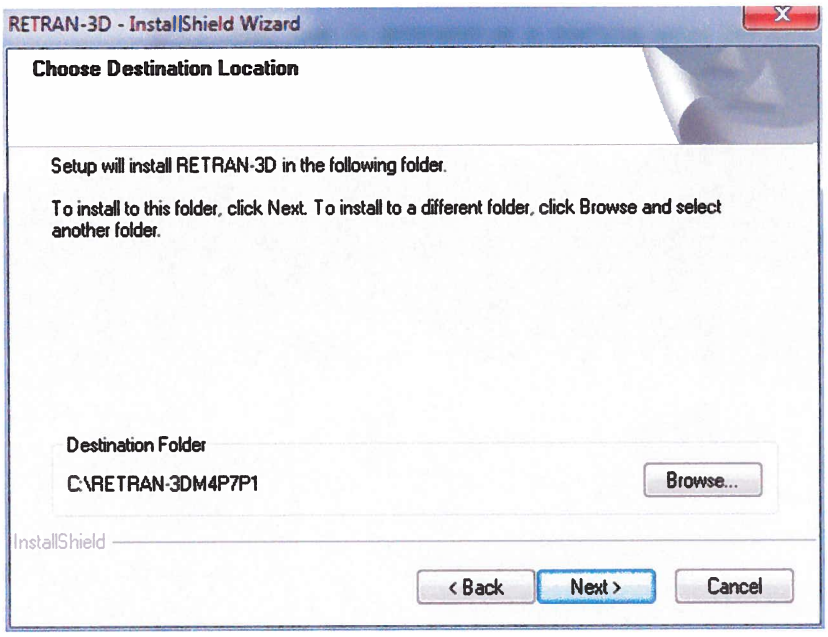
3. The **Software License Agreement** for the RETRAN-3D application will be displayed on the screen (shown below). If, after reading this agreement, you accept the terms of this agreement and wish to continue the installation, select the **Yes** button on that window to continue with the installation process. If you select the **No** button on this window, the installation process will be halted, and the RETRAN-3D software will not be installed on your computer. Note that installation of the RETRAN-3D software implies acceptance of this agreement.



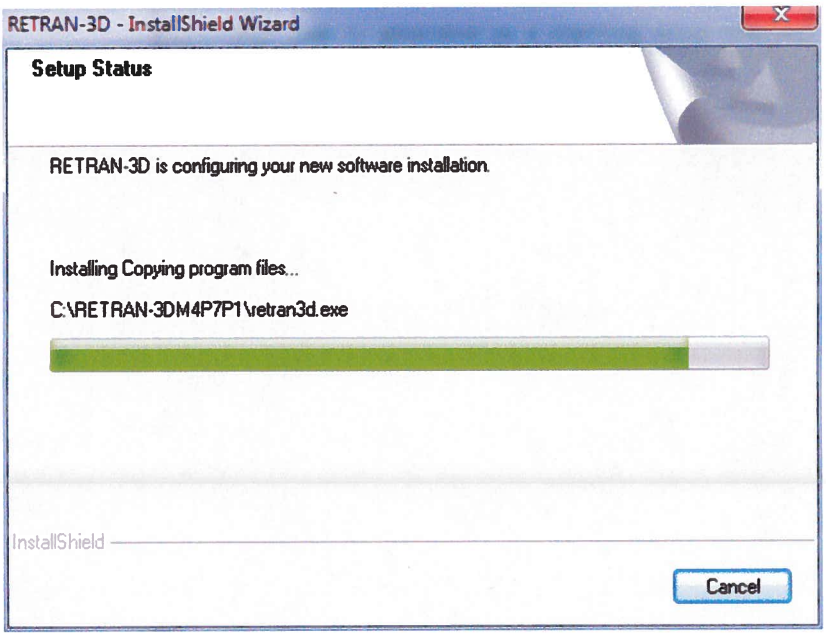
4. The **Platform Selection** window will appear (as shown to the right). This allows the installation target platform to be selected. Click the **Next>** button to continue.



5. The **Choose Destination Location** window will appear (as shown to the right). This window allows you to specify the location on your computer's hard disk for the RETRAN-3D software and its supporting files. The default location is **\Program Files\RETRAN-3D** on the computer hard disk where Windows is installed. If this default is satisfactory, simply select the **Next>** button to continue the installation process. You can use the **Browse...** button to select an alternative location for the software, followed by the **Next>** button to continue.

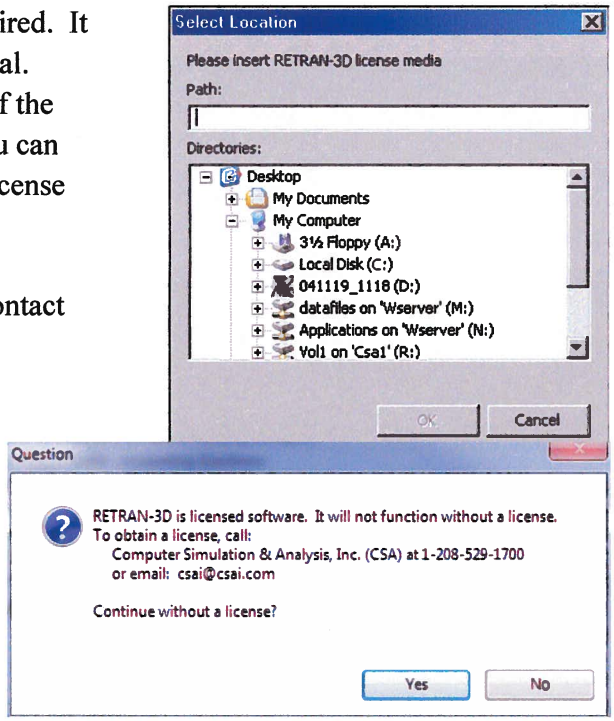


5. The installation process will then begin copying the needed files to your computer's hard disk and setting up the supporting files for the RETRAN-3D application. A progress meter (as shown below) will be updated throughout this installation step.

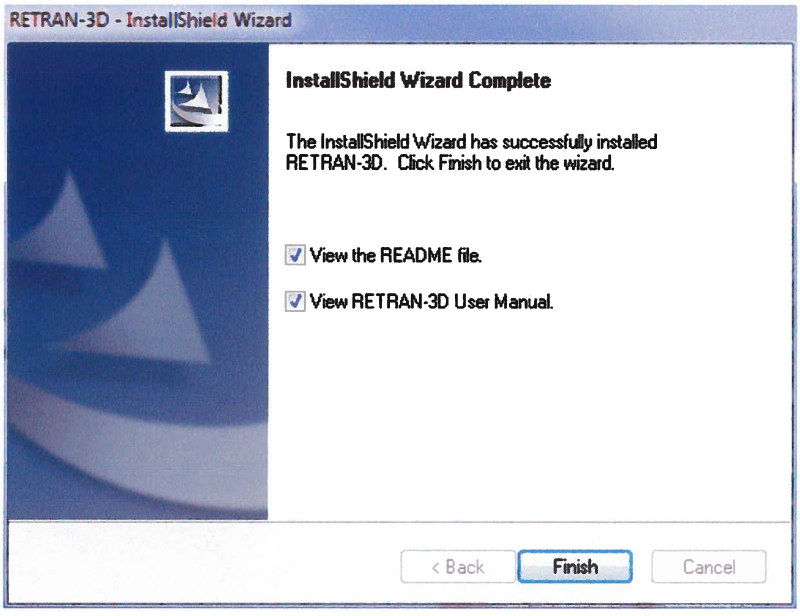


6. A license for the RETRAN-3D executable is required. It will generally be included with software transmittal. Insert the license media, then select the location of the license file as shown to the right. If necessary you can remove the installation CD-ROM and insert the license media at this point.

   If you do not have a license for RETRAN-3D, contact Computer Simulation & Analysis, Inc. (CSA) at 1-208-529-1700 or email: csai@csai.com. The installation will continue without a license file, but the RETRAN-3D executables will not function. Once a license file is obtained it must be copied to the destination location selected in step 4 above, the default location is **\Program Files\RETRAN-3D** on the computer hard disk where Windows is installed.

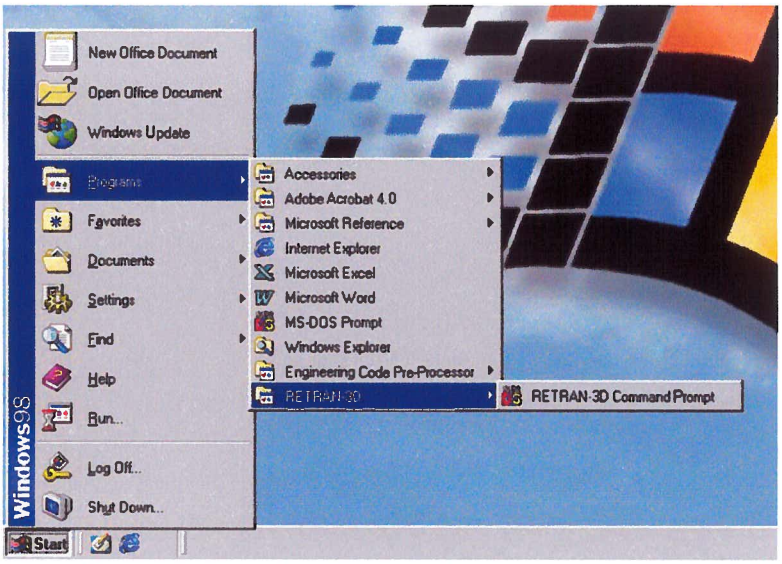


7. At the completion of RETRAN-3D installation, you will be shown the **Wizard Complete** window below. Select the appropriate box to view the README file and the RETRAN-3D User Manual. Select the **Finish** button to complete this installation.



8. The RETRAN-3D application is now installed on your computer system and ready to use. You should remove the installation CD-ROM and the license media and store them in a safe place, in case you should need to reinstall the RETRAN-3D application.

9. RETRAN-3D must be started from a command prompt window (DOS Prompt). A shortcut has been created for you on your Windows Start menu, as shown below. When started from the shortcut, the environment is automatically setup to run RETRAN-3D correctly. If



RETRAN-3D is started from a standard command prompt window, the environment must be manually setup. This process is described in detail in the README.TXT file. If RETRAN-3D was not installed in the default location, the starting location for the RETRAN-3D command prompt will need to be modified.

RETRAN-3D is not a Windows application; it is a console application run from a command prompt window by entering the appropriate command string from the keyboard. A procedure file, run.bat, shown in Table VI.2-1, is used to execute RETRAN-3D problems. This procedure file is installed with the RETRAN-3D executable(s). It provides the capability to specify the required input data files and optionally define the executable file. If **run** is entered in the command prompt window without any parameters, the procedure will list the input parameters that can be specified. RETRAN-3D will not be executed. If the executable file is defined using the default definition of the exec environment variable in the run.bat procedure file, the **–r** [rdeck] parameter is the only required parameter; otherwise, the executable file name must also be specified using the **–x** [exec] parameter. The **-m** [memsize] parameter can be used to adjust the size of the memory allocation, which is problem dependent. Other parameters may be required, depending on the code options specified in the input file.

To execute the sp1 sample problem, enter **run –r .\sample\sp1** from the keyboard. The output file will have the name of the input file with a .out extension, e.g., sp1.out. The output file can then be examined using a text editor.

10. RETRAN-3D is a licensed product. During installation, a license file is requested for the installed RETRAN-3D product. RETRAN-3D will not function without a license. If a license

Table VI.2-1

Windows Execution Procedure: run.bat

```
@echo off
rem
rem  RETRAN-3D run procedure (Command line version)
rem
rem
rem  Calling sequence
rem
rem    run -x exec -r rdeck -X xsecin -3 xsecin -B tape12 -R tape13 -C tape78
rem
rem   where all parameters are optional except -r (and rdeck).
rem   Note that parameters are case sensitive.
rem
rem
rem ================================================================
if "%1" == "/?" goto help
if "%1" == "-h" goto help
if "%1" == "--help" goto help

rem
rem
rem  Set environment variables
rem
    set exec=
    set exepth=
    set exenam=
    set exetim=
    set exesiz=
    set bdeck=
    set rdeck=
    set tape12=
    set tp12ed=
    set tape13=
    set tp13ed=
    set xsecin=
    set xseced=
    set tape78=
    set tp78ed=
    set CTAPE=
    set XTAPE=
    set opts=
    set nsm=
rem
    if not %1*==* goto parse
rem
rem ================================================================
rem
rem  Error processing
rem
:error
rem
    echo  **
    echo  **** ERROR ****  argument errors detected
```

Table VI.2-1 (Cont'd)

```
:help
    echo  *
    echo  * calling sequence
    echo  *
    echo  *      run -x exec -r rdeck -X xsecin -3 xsecin
    echo  *          -B tape12 -R tape13 -C tape78
    echo  *
    echo  *  where all paramters are optional except -r (and rdeck)
    echo  *
    echo  *  (parameters are order independent but are case sensitive)
    echo  *
    echo  *     -x  exec    = executable name
    echo  *     -r  rdeck   = retran input
    echo  *     -X  xsecin  = 1D X-section file (TAPE40)
    echo  *     -3  xsecin  = 3D X-section file (TAPE68)
    echo  *     -B  tape12  = old RETRAN data file (TAPE12)
    echo  *     -R  tape13  = old RETRAN restart file (TAPE13)
    echo  *     -C  tape78  = CDI file (TAPE78)
    echo  *     -m  memsize = pool memory argument
    echo  *     -ns         = no screen argument
    echo  *     -N  runnum  = run number argument
    echo  **
    exit /b
rem
rem  exec file does not exist
rem
:err_exec
    echo  *
    echo  *  Error **** RETRAN-3D executable file %exec% does not exist
    echo  *
    goto exit
rem
rem  RETRAN-3D input file does not exist
rem
:err_rdeck
    echo  *
    echo  *  Error **** RETRAN-3D input file %rdeck% does not exist
    echo  *
    goto exit
rem
rem  Cross section file does not exist
rem
:err_xsec
    echo  *
    echo  *  Error **** %XTAPE% cross section file %xsecin% does not exist
    echo  *
    goto exit
rem
rem  Input data file does not exist
rem
:err_tape12
    echo  *
    echo  *  Error **** TAPE12 data file %tape12% does not exist
    echo  *
```

Table VI.2-1 (Cont'd)

```
    goto exit
rem
rem  Restart data file does not exist
rem
:err_tape13
    echo  *
    echo  *  Error **** TAPE13 data file %tape13% does not exist
    echo  *
    goto exit
rem
rem  CDI file does not exist
rem
:err_tape78
    echo  *
    echo  *  Error **** TAPE78 data file %tape78% does not exist
    echo  *
    goto exit
rem
rem  Invalid parameter
rem
:err_opt
    echo  *
    echo  *  Error **** invalid parameter, %1 not valid option
    echo  *
    goto exit
rem
rem ================================================================
rem
:parse
    if %1*==* goto process
    if %1==-x goto x
    if %1==-r goto r
    if %1==-X goto ux
    if %1==-3 goto 3
    if %1==-B goto ub
    if %1==-R goto ur
    if %1==-C goto uc
    if %1==-m goto um
    if %1==-ns goto uns
    if %1==-N goto un
    if not %1*==* goto err_opt

rem
rem  Set executable file name
rem
:x
    if %2*==* goto error
    del /q _tmp789 2>nul
    set fff=%2
    set fff=%fff:.exe=%
    set exec=%fff%.exe
    if exist %exec% goto xx
    echo %fff%.exe >_tmp789 2>nul
    for /F "usebackq" %%a in (_tmp789) do set exec=%%~$PATH:a
```

Table VI.2-1 (Cont'd)

```
    del /q _tmp789 2>nul
    if %exec%*==*  goto err_exec
:xx
    echo %exec% >_tmp789 2>nul
    for /F "usebackq" %%a in (_tmp789) do set exepth=%%~dpa
    for /F "usebackq" %%a in (_tmp789) do set exenam=%%~nxa
    for /F "usebackq" %%a in (_tmp789) do set exesiz=%%~za
    for /F "usebackq" %%a in (_tmp789) do set exetim=%%~ta
    del /q _tmp789 2>nul

    shift
    shift
    goto parse
rem
rem  Set RETRAN-3D input deck
rem
:r
    if %2*==* goto error
    set bdeck=%~nx2
    set rdeck=%2
    set inpext=
    if exist %rdeck%%inpext%  goto r2
    set inpext=.inp
    if exist %rdeck%%inpext%  goto r2
    goto err_rdeck
rem
:r2
    shift
    shift
    if %rdeck% == INPUT goto parse
    if exist input    del input    > nul
    if exist output   del output   > nul
    if exist remarks  del remarks  > nul
    if exist TAPE2    del TAPE2    > nul
    if exist tape3    del TAPE3    > nul
    if exist tape14   del tape14   > nul
    if exist tape41   del tape41   > nul
    if exist tape42   del tape42   > nul
    if exist tape60   del tape60   > nul
    if exist ERR_LOG  del ERR_LOG  > nul
    if exist VBCFIL   del VBCFIL   > nul
    if exist R3D_PLOT del R3D_PLOT > nul
    copy %rdeck%%inpext% INPUT > nul
    goto parse
rem
rem  Set 1D cross section file
rem
:ux
    if %2*==* goto error
    set xsecin=%2
    set xseced=with TAPE40:
    set XTAPE=TAPE40
    if not exist %xsecin% goto err_xsec
    if not %xsecin%==TAPE40 copy %xsecin% TAPE40 > nul
```

Table VI.2-1 (Cont'd)

```
    shift
    shift
    goto parse
rem
rem  Set 3D cross section file
rem
:3
    if %2*==* goto error
    set xsecin=%2
    set xseced=with TAPE68:
    set XTAPE=TAPE68
    if not exist %xsecin% goto err_xsec
    if not %xsecin%==TAPE68 copy %xsecin% TAPE68 > nul
    shift
    shift
    goto parse
rem
rem  Set input retran data file
rem
:ub
    if %2*==* goto error
    set tape12=%2
    set tp12ed=with TAPE12:
    if not exist %tape12% goto err_tape12
    if not %tape12%==TAPE12 copy %tape12% TAPE12 > nul
    shift
    shift
    goto parse
rem
rem  Set restart data file
rem
:ur
    if %2*==* goto error
    set tape13=%2
    set tp13ed=with TAPE13:
    if not exist %tape13% goto err_tape13
    if not %tape13%==TAPE13 copy %tape13% TAPE13 > nul
    shift
    shift
    goto parse
rem
rem  Set CDI file
rem
:uc
    if %2*==* goto error
    set tape78=%2
    set tp78ed=with TAPE78:
    set CTAPE=TAPE78
    if not exist %tape78% goto err_tape78
    if not %tape78%==%CTAPE% copy %tape78% TAPE78 > nul
    shift
    shift
    goto parse
rem
```

Table VI.2-1 (Cont'd)

```
:um
    if %2*==* goto error
    set mem=%2
    set mem=%mem:k=000%
    set mem=%mem:K=000%
    set mem=%mem:m=000000%
    set mem=%mem:M=000000%
    set opts=%opts% -m %mem%
    shift
    shift
    goto parse
rem
:uns
    set opts=%opts% -ns
    set nsm=No
    shift
    goto parse
rem
:un
    if %2*==* goto error
    set opts=%2
    shift
    shift
    goto parse
rem
rem ================================================================
rem
rem  Start processing according to options supplied
rem
:process
rem
rem
    if %rdeck%*==* goto err_rdeck

rem
rem  header information
rem
    echo  OS:           PC          >HEADR 2>nul
    echo  User Name:    %USERNAME% >>HEADR 2>nul
    for /F "usebackq tokens=1,*" %%a in (`date /T`) do set dattim=%%b
    for /F "usebackq tokens=*" %%a in (`time /T`) do set dattim=%dattim% %%a
    echo  Start time:   %dattim%  >>HEADR 2>nul
    echo. >> HEADR 2>nul
    echo  Input File Information: >>HEADR 2>nul
    echo        Input File: %rdeck%%inpext% >>HEADR 2>nul
    if not %xsecin%*==*  echo       %xseced% %xsecin% >>HEADR 2>nul
    if not %tape12%*==*  echo       %tp12ed% %tape12% >>HEADR 2>nul
    if not %tape13%*==*  echo       %tp13ed% %tape13% >>HEADR 2>nul
    if not %tape78%*==*  echo       %tp78ed% %tape78% >>HEADR 2>nul
    echo. >> HEADR 2>nul
    echo  Command:  %exec% %opts% >> HEADR 2>nul
    echo. >> HEADR 2>nul
    echo  Program Name:     %exenam%  (%exepth%) >> HEADR 2>nul
    echo  Program Size:     %exesiz%  >> HEADR 2>nul
```

Table VI.2-1 (Cont'd)

```
    echo  Program Date:     %exetim%  >> HEADR 2>nul

rem
rem  Execute RETRAN-3D
rem
    if not %nsm%*==* goto go_exec
    echo .
    echo .  Running RETRAN-3D executable %exec% %opts%
    echo .          with  INPUT: %rdeck%%inpext%
    if not %xsecin%*==*  echo .          %xseced% %xsecin%
    if not %tape12%*==*  echo .          %tp12ed% %tape12%
    if not %tape13%*==*  echo .          %tp13ed% %tape13%
    if not %tape78%*==*  echo .          %tp78ed% %tape78%
    echo .

:go_exec

    %exec% %opts%
    set xstat=%errorlevel%

rem
rem  continue header information
rem
    for /F "usebackq tokens=1,*" %%a in (`date /T`) do set dattim=%%b
    for /F "usebackq tokens=*" %%a in (`time /T`) do set dattim=%dattim% %%a
    echo. >> HEADR 2>nul
    echo  Execution ended:  %dattim% >> HEADR 2>nul
    echo. >> HEADR 2>nul
    echo  Exit status:  %xstat%  >> HEADR 2>nul
    echo. >> HEADR 2>nul

rem
rem  cleanup files
rem
    if %rdeck%==INPUT goto exit
    if %rdeck%==input goto exit
rem
    if exist %bdeck%.tape14 del %bdeck%.tape14
    if exist %bdeck%.tape41 del %bdeck%.tape41
    if exist %bdeck%.tape42 del %bdeck%.tape42
    if exist %bdeck%.tape43 del %bdeck%.tape43
    if exist %bdeck%.tape60 del %bdeck%.tape60
    if exist %bdeck%.ERR_LOG del %bdeck%.ERR_LOG
    if exist %bdeck%.vbc     del %bdeck%.vbc
    if exist %bdeck%.plt    del %bdeck%.plt
rem
rem
    if exist TAPE14 move TAPE14  %bdeck%.tape14 > nul
    if exist TAPE41 move TAPE41  %bdeck%.tape41 > nul
    if exist TAPE42 move TAPE42  %bdeck%.tape42 > nul
    if exist TAPE42 move TAPE43  %bdeck%.tape43 > nul
    if exist TAPE60 move TAPE60  %bdeck%.tape60 > nul
    if exist ERR_LOG move ERR_LOG %bdeck%.ERR_LOG > nul
    if exist VBCFIL  move VBCFIL  %bdeck%.vbc     > nul
```

Table VI.2-1 (Cont'd)

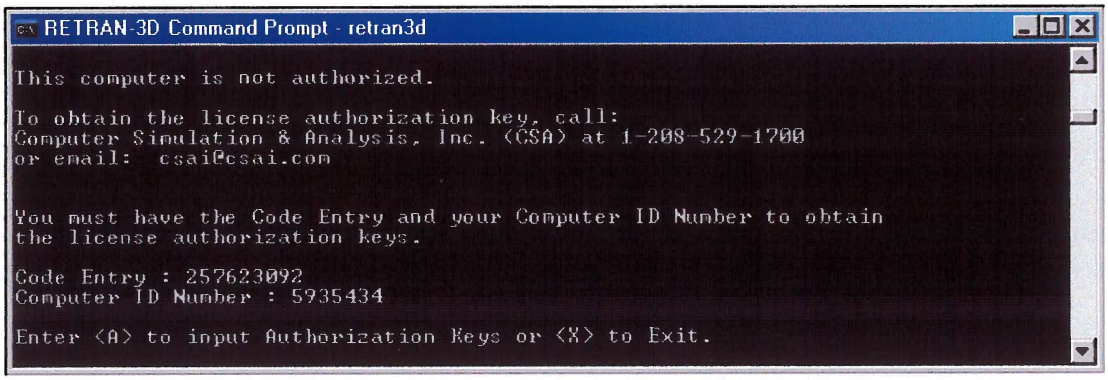
```
    if exist R3D_PLOT move R3D_PLOT %bdeck%.plt > nul
    if not %xsecin%*==%XTAPE%* del %XTAPE% > nul
    if not %tape78%*==%CTAPE%* del %CTAPE% > nul

rem
rem  finish header information
rem
    echo  Output File Information: >>HEADR 2>nul
    echo          Output File: %bdeck%.out >>HEADR 2>nul
    if exist %bdeck%.tape60  echo      TAPE60 file: %bdeck%.tape60 >> HEADR 2>nul
    if exist %bdeck%.tape14  echo     Restart file: %bdeck%.tape14 >> HEADR 2>nul
    if exist %bdeck%.ERR_LOG echo           Error Log: %bdeck%.ERR_LOG >> HEADR
2>nul
    if exist %bdeck%.vbc     echo            VBC file: %bdeck%.vbc     >> HEADR
2>nul
    if exist %bdeck%.plt    echo         PLOT file: %bdeck%.plt    >> HEADR 2>nul
    echo. >> HEADR 2>nul
rem
rem
    copy HEADR+OUTPUT+REMARKS %bdeck%.out > nul
    del INPUT > nul
    del OUTPUT > nul
    del REMARKS > nul
    del HEADR > nul

rem
rem  Exit path
rem
:exit
    if exist TAPE2  del TAPE2 >nul
    if exist TAPE3  del TAPE3 >nul
    if exist TAPE20 del TAPE20 >nul
    if exist fort.2 del fort.2 >nul
    if exist fort.3 del fort.3 >nul

rem
rem  UnSet environment variables
rem
    set exec=
    set exepth=
    set exenam=
    set exetim=
    set exesiz=
    set bdeck=
    set rdeck=
    set tape12=
    set tp12ed=
    set tape13=
    set tp13ed=
    set xsecin=
    set xseced=
    set tape78=
    set tp78ed=
    set CTAPE=
```

Table VI.2-1 (Cont'd)

```
set XTAPE=
set opts=
set nsm=
set xstat=
set dattim=
set fff=
```

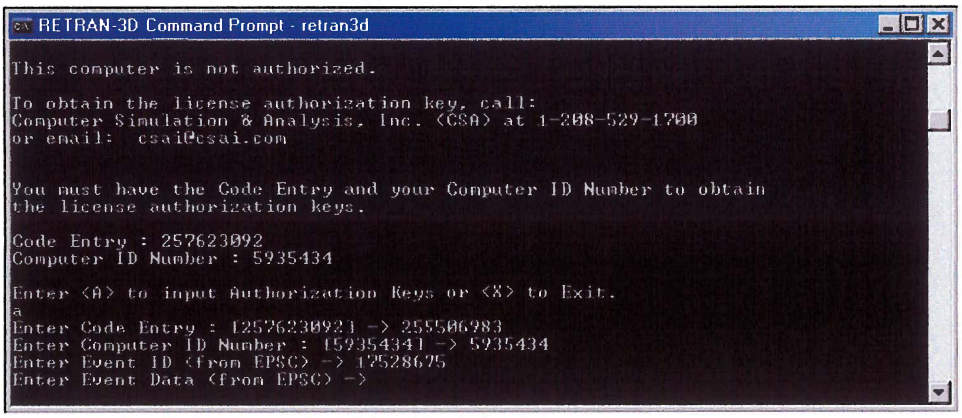
file was not installed in Step 6, RETRAN-3D will fail with a message that "This software is not licensed" followed by instructions for obtaining the license authorization. If these messages are encountered, refer to Step 6 for instructions on installing a license file after the initial installation. Once a valid license file is installed in the proper directory, RETRAN-3D will execute.

In most cases the RETRAN-3D executable is software bound only. This means that if the license file exists the code will execute. In some cases, the code is both software and hardware bound (i.e., each computer the code is installed on requires licensing). For these cases, the first time RETRAN-3D is executed, you will be required to contact Computer Simulation & Analysis, Inc. (CSA) to obtain an authorization key (or Event ID). RETRAN-3D can be executed using the run.bat procedure discussed in Step 10 (requires an input file) or by typing the executable file name (with or without the .exe extension). If the executable file name is used, the execution step may fail due to the lack of an input file. However, this should not affect the license authorization process. If the run.bat procedure is used, the RETRAN-3D will be run using the specified input deck upon completion of the license authorization. To initiate the license authorization process, you will need to furnish CSA with the Code Entry and Computer ID given by RETRAN-3D when executed, as shown below.



You may exit RETRAN-3D by entering an X. The authorization keys can be entered now or at a later time by executing RETRAN-3D again. If entered at a later time, the Code Entry value provided to CSA to obtain the license authorization key(s) must be re-entered. Each time RETRAN-3D is executed, a new Code Entry value is generated.

11. After obtaining the authorization keys from CSA, execute RETRAN-3D. Enter an A to input the keys. As shown below, the user must enter the Code Entry value (as given previously) and the Event ID. The Computer ID will not change when executed from the same computer. This specific Event ID does not require Event Data, when prompted press the **<Enter>** key. The specific version of RETRAN-3D will now execute as expected on the current machine. In the future, PC hardware changes may alter the Computer ID and would require repeating the licensing process.



12. The installation is verified by running the checkin.bat procedure. It is shown in Table VI.2-2. First, it runs the standard sample problems which create auxiliary file ouput (TAPE60). These files are then concatenated to a single file, TSTDTA, which is then compared to the baseline data in STDDTA. Table VI.2-1 shows the run.bat procedure that is used to execute the code. It is used once for each sample problem executed from the checkin.bat procedure. Before running the checkin.bat procedure by entering

    checkin

    from the keyboard, make the necessary changes to the checkin.bat procedure to ensure that the newly installed executable is tested.

    An alternate approach would be to enter

    checkin [executable-path/name]

    where the parameter gives the path and name of the executable to test. The **checkin.bat** procedure runs the sample problems located in the .\sample\ directory.

    After running the sample problems, the **checkin.bat** procedure verifies the installation by comparing results to the supplied results in the .\compare\ directory. A summary of these comparisons is written to the REPORT file in the .\compare\ directory. After **checkin.bat** has completed running the sample problems, use a text editor to examine the REPORT file to determine if the installation was successful. All sample problems should show 0 differences in the REPORT file. Any reported differences should be investigated to determine the cause.

3.0 LINUX CODE MAINTENANCE

It may be necessary to revise the RETRAN-3D source code to correct an error or add a new feature. If this is the case, the appropriate source code subroutine(s) can be revised using a text editor and a new executable created using the Fortran 95 compiler and linker. The **bld** installation script Table VI.1-1 can also be used to build an updated program executable file. This is

Table VI.2-2

Windows Installation Verification Procedure: checkin.bat

```
@echo off
rem
rem  Script to execute all 16 sample problems.  After running the sample
rem  problems, the TAPE60 files are concatenated to file TSTDTA which is
rem  then used as input to the COMPAR program.
rem
rem
rem  Select executable file
rem
if exist retran3d.exe set xxx=..\retran3d.exe
if exist r3m004p5.exe set xxx=..\r3m004p5.exe
if not %1*==* set xxx=..\%1.exe

rem
rem
rem  Run sample problems
rem
cd sample
if not exist %xxx% goto err_exec
call ..\run -x %xxx% -r sp1
call ..\run -x %xxx% -r accum
call ..\run -x %xxx% -r sp5
call ..\run -x %xxx% -r tlta
call ..\run -x %xxx% -r ttwob
call ..\run -x %xxx% -r ucrw
call ..\run -x %xxx% -r fl2d
call ..\run -x %xxx% -r turb
call ..\run -x %xxx% -r ttqx1 -X ttqx1.t40
call ..\run -x %xxx% -r atws
call ..\run -x %xxx% -r pipe
call ..\run -x %xxx% -r wovrs
call ..\run -x %xxx% -r lrhr
call ..\xfgen A pwr
call ..\run -x %xxx% -r pwr -3 pwr.bxf -C pwr.cdi
call ..\xfgen A slb
call ..\run -x %xxx% -r slb -3 slb.bxf -C slb.cdi
call ..\xfgen A bwr
call ..\run -x %xxx% -r bwr -3 bwr.bxf -C bwr.cdi

rem
rem  Concatenate the TAPE60 files to file TSTDTA
rem
type   sp1.tape60  > ..\compare\TSTDTA
type accum.tape60 >> ..\compare\TSTDTA
type   sp5.tape60 >> ..\compare\TSTDTA
type  tlta.tape60 >> ..\compare\TSTDTA
type ttwob.tape60 >> ..\compare\TSTDTA
type  ucrw.tape60 >> ..\compare\TSTDTA
type  fl2d.tape60 >> ..\compare\TSTDTA
type  turb.tape60 >> ..\compare\TSTDTA
type ttqx1.tape60 >> ..\compare\TSTDTA
```

Table VI.2-2 (Cont'd)

```
type  atws.tape60 >> ..\compare\TSTDTA
type  pipe.tape60 >> ..\compare\TSTDTA
type wovrs.tape60 >> ..\compare\TSTDTA
type  lrhr.tape60 >> ..\compare\TSTDTA
type   pwr.tape60 >> ..\compare\TSTDTA
type   slb.tape60 >> ..\compare\TSTDTA
type   bwr.tape60 >> ..\compare\TSTDTA

rem
rem  Run compare program
rem
cd ..\compare
if exist INDTA del INDTA > nul
if exist REPORT del REPORT > nul
copy R3D.TXT INDTA  >nul
.\compare2.exe
goto exit

rem
rem  exec file does not exist
rem
:err_exec
 echo  *
 echo  *  Error **** RETRAN-3D executable file %xxx% does not exist
 echo  *

:exit
cd ..
```

accomplished by placing the revised subroutine source code in the directory where the original build was done (the directory where the executable file was created). The **bld** script should be run by omitting the **–B** parameter (use with the initial installation). This causes the **compile_list** file to be ignored and any source files in the directory where **bld** is executed will be compiled and included in the new executable file.

When doing an update or partial build, the object files from the original installation must either reside in the installation directory or the object file directory specified using the **–O** parameter. The object files for the revised subroutines will replace the files from the original installation and a new executable will be created. This method is much faster than re-building the code from scratch using the steps given for the code installation.

4.0 TECHNICAL SUPPORT

Technical support related to the installation and use of RETRAN-3D and issues related to potential code errors is provided by the RETRAN maintenance contractor. This effort is funded by annual contributions to the maintenance group by member organizations. Support for nonmembers is limited to error reporting.

For questions relating to membership in the RETRAN Maintenance Group, contact Computer Simulation & Analysis, Inc. (CSA), the RETRAN maintenance contractor. To obtain support, contact

Mr. Garry C. Gose
Telephone: (208) 529-1700, Ext. 22
E-Mail: gcg@csai.com

or

Mr. Mark P. Paulsen
Telephone: (208) 529-1700, Ext. 16
E-mail: paulsen@csai.com

Computer Simulation & Analysis, Inc.
855 N. Capital Ave. Suite 1
P. O. Box 51596
Idaho Falls, ID 83405
Telephone: (208) 529-1700
Fax: (208) 529-1723

Office hours are nominally 8:00 A.M. to 5:00 P.M. Mountain Time. Additional RETRAN support information is available on the CSA web site at www.csai.com. It includes trouble reports that have been filed, including their status and possible workarounds; online facilities for submitting trouble reports and related supporting information (input and output files); a searchable RETRAN bibliography; links to current RETRAN items of interest, i.e., meeting information for

International RETRAN Meetings and RETRAN User Group Meetings; and RETRAN Newsletters containing articles covering current RETRAN issues, unique applications of RETRAN, and modeling tips.

The RETRAN Maintenance Group does not support formal training in the use of RETRAN-3D as part of the funded maintenance activity. However, CSA provides a range of training programs including both basic and advanced courses and on-site training that can be tailored to meet the specific needs of an organization. CSA staff members have been involved in the development and maintenance of the RETRAN series of computer codes, have performed a variety of RETRAN analyses, and have provided training and consulting assistance to code users for a number of years. This application and consulting experience combined with the knowledge of the code and interaction of the models in RETRAN-3D offers a unique learning opportunity for both new and experienced code users. Contact one of the CSA staff members listed above to discuss your training needs.

VII

REFERENCES

II.2-1 Paulsen, M. P., et al., "RETRAN-3D - A Program for Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems", Programmer's Manual, EPRI NP-7550(A), Volume 2, Revision 6.3, July 2007.

IV.2-1 Dias, A. F., et al., "CORETRAN-01 - A Three-Dimensional Program for Reactor Core Physics and Thermal-Hydraulics Analysis", Theory and Numerical Analysis, EPRI WO-3574, October 1997.

IV.2-2 Dias, A. F., et al., "CORETRAN-01 - A Three-Dimensional Program for Reactor Core Physics and Thermal-Hydraulics Analysis", User's Manual, EPRI WO-3574, October 1997.

IV.2-3 Jones, D. B., et al, "CPM-3 - A Core Physics Module for the Analysis of Nuclear Fuel Assemblies Using Arbitrary Geometry Modeling", User's Manual, EPRI RP-3418, October 1997.

IV.2-4 Edenius, M., et al., "CASMO-4 - A Fuel Assembly Burnup Program", User's Manual, Studsvik/SOA-95/1, Revision 0, September 1995.

IV.2-5 "SIMULATE-3 - Advanced Three-Dimensional Two-Group Reactor Analysis Code", User's Manual, Studsvik/SOA-95/15, Revision 0, October 1995.

VI.1-1 HP Fortran 90 Programmer's Reference, HP Part Number: B3908-90002, October 1998.

VI.1-2 XL Fortran for AIX: Language Reference, Version 8.1.1, SC09-4947-01, Second Edition, June 2003.

VI.1-3 SUN Fortran User' Guide, Forte Developer 7, May 2002.

VI.1-4 GNU Fortran, http://gcc.gnu.org/fortran/, August 2009.

VI.1-5 The G95 Project, http://g95.org, August 2009.

VI.1-6 Intel Professional Edition Compilers, http://software,intel.com/en-us/intel-compliers/, August 2009.

References

APPENDIX A

THE COMPARE2 PROGRAM

The COMPARE2 computer program provides a method for comparing results from a new RETRAN-3D installation and those for a standard or baseline set of results. The baseline results are created using a RETRAN-3D version that was previously deemed to have been installed correctly. The purpose of the program is to provide a means for the RETRAN-3D installer to determine if a new RETRAN-3D installation has been made successfully. COMPARE2 uses RETRAN-3D auxiliary files to provide the information to be compared against a baseline dataset. The auxiliary file contains a table of minor edit variable results, generated by a RETRAN-3D run. The minor edit variables have been chosen to capture important features or trends in each sample problem calculation. A companion shell script or batch file (checkin.sh or checkin.bat) runs the appropriate sample problems and then concatenates the individual problem auxiliary files (TAPE60) into a single file (TSTDTA), which is then compared against the baseline data (STDDTA).

The compare program, COMPARE2 is written Fortran 95 and is transmitted with the RETRAN-3D code transmittal packages. COMPARE2 requires three input files. The first file, STDDTA, contains the baseline data that are generated by the standard version of RETRAN-3D. This standard file contains data for all of the sample problems. STDDTA is transmitted with the RETRAN-3D package and should not be altered. The second file, TSTDTA, contains the concatenated auxiliary file (TAPE60) results for all sample problems for the RETRAN-3D version to be compared. COMPARE2 assumes that the problem order in TSTDTA is the same as in STDDTA. The third file, INDTA, allows the user to specify which of the set of sample problems are to be compared, to flag the desired output format, and to include user-defined title information.

COMPARE2 consists of four functions or procedures. A functional description of the modules of the program is provided in the following sections.

PREPROCESSING

This process defines or characterizes the comparison task to be performed. COMPARE2 assumes that the content of the standard data (STDDTA) and test data (TSTDTA) files is order dependent. While data from all sample problems is not required, the data must be presented in the correct order. The STDDTA file defines the order of the sample problems; it follows the order in which the sample problems are presented in the RETRAN-3D User's Manual – Volume 3, Section VIII. The STDDTA file also defines which minor edits are to be compared for each sample problem. The sample decks included in the RETRAN-3D transmittal will produce auxiliary (TAPE60) data files compatible with STDDTA. The appropriate ‘checkin’ script or batch file combines the newly created auxiliary files into a standard data file, STDDTA.

INPUT REQUIREMENTS FOR PREPROCESSING

The user must supply an input data file, INDTA. The minimum data supplied will be two lines. The first line will contain a title for the comparison data. The title must be 80 characters or less and set off by single quotes. The second line contains one of the three following words; 'FULL', 'SHORT', or 'DEBUG'. Line two specifies the desired amount of detail for the comparison report. Optionally, a third and fourth line can be added. Line three contains two floating point values, PTOL and SMALL. PTOL specifies the tightness for comparison and SMALL defines the significance limit. By default PTOL and SMALL are 1.0D-3 and 1.0D-6, respectively. If only some of the sample problems are to be compared, line four specifies by number which problems are desired. As an example, if line four consists of:

1 5 9

only three problems are to be processed; Sample Problem One, Turbine Trip Without Bypass with Point Kinetics, and Turbine Trip Without Bypass with Space-Time Kinetics (SP1, TTWOB, and TTQX1, respectively). These should be the only results contained in file TSTDTA for comparison with the standard. No card or a blank card indicates the default case of all problems.

FILE READING

The READER process reads TSTDTA for NPROB sample cases. NPROB defaults to all. In general, all parameters to be compared are read into arrays STD or TST. The arrays are three-dimensional containing NPROBxNCARxNRECORD values. That is, the arrays contain ALL of the information to be compared. Note that NRECORD is determined by the number of data records found in STDDTA. If the TST array in any given problem has NRECORD that is different, an error condition is present.

If a reading error occurs or record lengths are not consistent with that expected (as discussed above, the STD arrays are predetermined), then appropriate diagnostic information is printed and the case is terminated. The information should indicate which problem, and which record, caused the error condition. As discussed above in the preprocessing section, it is assumed
that the user has stored the information in TSTDTA in the correct order.

OUTPUT PARAMETER COMPARISONS

The comparison or tolerance test routine will have two levels of error testing. The first will be a simple percentage or ratio test. This will be a test in which, for all values, the error will be defined as:

STD(I,J,K) - TST(I,J,K) / STD(I,J,K)

Currently, if the difference is greater than PTOL (by default 0.1%) then a warning message will be printed. This tolerance was determined during the testing phase. If the value of the parameter is less than SMALL (by default 1.0E-06), then a ratio test will not be made because the parameter is

considered to be insignificant. This is consistent with the type of minor edit parameters that have been selected for each of the sample problems.

REPORTING RESULTS

The results of the comparisons are edited for the user to evaluate. The amount of information is controlled by the user. In general, the report includes the title information provided by the user and summarizes, by sample problem, where differences occur.

DATA STRUCTURES

The structure of the data is given below. There will be two three-dimensional arrays:

STD (NPROB,NPARAM,NREC)

and

TST (NPROB,NPARAM,NREC)

where

NPROB	=	Sample Problem Index (range: 1 to # of sample problems),
NPARAM	=	Parameter Index (range: 1 to # of minor edits per problem), and
NREC	=	Record Index (range: 1 to # of data record per problem).

STD is the array containing the baseline data values read from the STDDTA file. Array TST contains data values for comparison read from TSTDTA. If only some of the samples are being compared, TST will be a subset of STD.

HOW TO USE THE COMPARE2 PROGRAM

COMPARE2 is easy to implement any platform. The user can compile COMPARE2 using standard Fortran 90 with no special options. A typical installation process is illustrated below

f95 compare_data.f90 compare2.f90 report.f90 –o compare2

where the Fortran 95 compiler command is f95. This may vary from platform to platform and will also depend to the particular compiler uses. The order that the source files are compiled is important. File compare_data.f90 must be complied first since it is a data module. The compiler will create a compare_data.mod file that will be required for compilation of the two other files.

COMPARE2 requires three input files, STDDTA, TSTDTA, and INDTA. STDDTA will be included on the RETRAN-3D transmittal media. A shell script called 'checkin' will also be supplied with the transmittal. It will also concatenate the RETRAN-3D auxiliary files (TAPE60s)

for all sample problems to create the TSTDTA file. A sample copy of INDTA will be supplied and the user need only modify the title line.

APPENDIX B

THE BXFTOOL PROGRAM

The RETRAN-3D cross-section file format is binary and this can lead to portability problems between different platforms such as Linux and Windows. The cross-section files on the RETRAN-3D transmittal are therefore converted to ASCII format to help mitigate the portability problem. The computer program BXFTOOL is used to perform the RETRAN-3D cross-section file (BXF file) conversion task.

There are two forms of the binary cross-section file, a long form that can contain many zeros and a more compact shorter form. Only the long form was used with RETRAN-3D MOD004.1 and earlier versions. RETRAN-3D MOD004.2 supports both the long and short forms. BXFTOOL can be used to convert a long form binary cross-section file to a short form binary file.

BXFTOOL can read a RETRAN-3D binary cross-section file and convert it to a fixed ASCII format. It can optionally read either an ASCII or long form binary file and convert it to binary. Another option directs BXFTOOL to read and interpret cross-section array sizes and other descriptive integer values from the cross-section file and write the information to the standard output file as a diagnostic tool.

BXFTOOL is executed during the RETRAN-3D installation procedure. As part of this installation, a Linux shell script "xfgen.sh" (or Windows batch file "xfgen.bat") is used to convert ASCII format cross sections from the RETRAN-3D transmittal CD to binary format.

HOW TO USE BXFTOOL

BXFTOOL is written in Fortran 77 and can be installed using either a Fortran 77 or 95 compiler. The names for the various compilers vary from platform to platform and compiler vendor. An example command to compile and install BXFTOOL would be

f77 –o bxftool BXFTOOL.F

which creates an executable file named bxftool.

Input Requirements

BXFTOOL reads a single integer flag, ICASE, from standard input (FORTRAN Unit 5) and its numerical value directs processing of the cross-section files.

The possibilities are:

ICASE = 0	BXFTOOL reads a binary format cross-section file and writes detailed information about the structure of the file to standard output.
ICASE = 1	BXFTOOL reads a binary format cross-section file and writes one in ASCII format.
ICASE = 2	BXFTOOL reads an ASCII format cross-section file and writes a binary cross-section file (short form).
ICASE = 3	BXFTOOL reads a long form binary cross-section file and writes a new short form file.

Output Files

If ICASE = 0, BXFTOOL reads from an unformatted binary cross-section file, "bxfile" on FORTRAN Unit 7, and writes description information to FORTRAN Unit 6.

If ICASE = 1, BXFTOOL reads from an unformatted binary cross-section file, "old.bxf" on FORTRAN Unit 7, and writes a formatted ASCII file "new.axf" on FORTRAN Unit 9.

If ICASE = 2, BXFTOOL reads from a formatted ASCII file, "old.axf"' on FORTRAN Unit 9, and writes a unformatted binary cross-section file, "new.bxf" to FORTRAN Unit 7.

If ICASE = 3, BXFTOOL reads from an unformatted binary cross-section file (long form), "old.bxf" on FORTRAN Unit 7 and writes a new unformatted binary cross-section file (short form), "new.bxf" on FORTRAN Unit 9.

Script xfgen.sh or Windows xfgen.bat Batch File

All RETRAN-3D multidimensional cross-section files on the RETRAN-3D transmittal CD are in ASCII format and must be converted to binary. On Windows machines, xxx.bat files are used instead of the xxx.sh script files used on Linux.

The RETRAN-3D "checkin.sh" script will invoke the "xfgen.sh" script with a flag to convert the cross-section files to binary format, before a given three-dimensional kinetics sample problem is executed.

Any existing temporary files in the execution directory named "new.axf", "new.bxf", "old.axf" or "old.bxf" are deleted. xfgen.sh then moves the ASCII cross-section file for a given sample problem to a temporary ASCII file, "old.axf". It then directs BXFTOOL to read the temporary ASCII or binary cross-section file, converting it to temporary binary file, "new.bxf ".

APPENDIX C

CODE MODIFICATION SUMMARY

This Appendix contains a summary of code modifications that have been made to RETRAN-3D since MOD003.0 was review by the NRC. It is intended to provide users with an overview of each modification and identify those that are new models that have not been reviewed and may require validation and NRC review before they are used in licensing submittals. Modifications made to create all code versions between MOD003.1 and MOD004.7.1 are included.

RETRAN-3D MOD003.0 was reviewed by the NRC staff and a Safety Evaluation Report subsequently issued. During the review process, a number of code modifications were suggested by the staff. These modifications along with error corrections that were made during the review period were incorporated into the code by adding these revisions to MOD003.0. The new code version was identified as MOD003.1. The corresponding documentation was designated as Revision 5 of EPRI NP-7450(A) "RETRAN-3D - A Program for Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems", Volumes 1-4. It also contained the SER and related review requests for additional information and the associated responses.

Since the release of MOD003.1, subsequent versions of RETRAN-3D have been created and released. Each of these versions included modifications that added new features as well as those that corrected problems reported in trouble reports. Table C-1 summarizes the modifications to the three source code libraries comprising RETRAN-3D that were included in the released versions of RETRAN-3D. The name of each library is also included. Each code version identified was created by adding the corresponding modifications identified in Table C-1 to the previous code version.

Version numbers are used to uniquely identify different code revision levels that are released to the RETRAN-3D user community. The first nonzero digit indicates the major revision, while the digit following the decimal point indicates subsequent minor revisions. The major revision digit is incremented when new models are added to a previously reviewed code version. For example, the change in version number from MOD003.1 to MOD004.1 indicates that there were new models added that have not been reviewed by the NRC. Normally the new version would have been MOD004.0 rather than MOD004.1 and in fact there was a MOD004.0 code version created, but it was not released. It was an intermediate version that provided the basis for development of MOD004.1 which was subsequently released.

A summary of each code modification is given in either Table C-2 or C-3, depending on whether the modification was due to a new feature or error correction. Modifications related to new features or enhancements are summarized in Table C-2 and those associated with error corrections are summarized in Table C-3. The modifications are also arranged by numerical order for each code version.

Table C-1

Source Code Modifications Included In New Code Versions

Code Version	SLIB77 Source Code Library Modifications/Library RETRAN Library	3-D Kinetics Library	Environmental Library
MOD003.1	mod_142 through mod_178 (R3M003P1.PPL)	armd_020 through armd_037 (CSA007.PPL)	enmd_004 through enmd_005 (ENVMD08.PPL)
MOD004.1	mod_179 through mod_254 (R3M004P1.PPL)	armd_038 through armd_047 (3DKIN009.PPL)	enmd_006 through enmd_010 (ENVMD10.PPL)
MOD004.2	mod_255 through mod_280 (R3M004P2.PPL)	armd_048 through armd_049 (3DKIN010.PPL)	(ENVMD10.PPL)
MOD004.3	mod_281 through mod_303 (R3M004P3.PPL)	(3DKIN010.PPL)	enmd_011 through enmd_014 (ENVMD11.PPL)
MOD004.4	mod_304 through mod_351 Fortran 77 version of MOD004.4 no longer uses SLIB 77.	armd_050	enmd_015 through enmd_017

The Fortran 95 version is no longer maintained using SLIB77. A single source code archive is maintained using a version control program.

MOD004.4f95	All modifications through mod_351	armd_050	All through enmd_017
MOD004.5f95	mod_352 through mod_354		
MOD004.6f95	mod_355 through mod_410		
MOD004.7f95	mod_411 through mod_481		
MOD004.7.1	mod_482 through mod_497		

Some of the new features summarized in Table C-2 require input data either to activate or deactivate the feature. When the code developers considered the new feature to be the best modeling approach, it was made the default option; but, an option is generally available to obtain a backward compatible way to run the new code version. A more detailed description of the input options can be found in Section IV of the RETRAN-3D User's Manual.

The description field also indicates if the new feature is a new model that has not been reviewed, if it is a user convenience, an input or output feature, compiler- or platform-specific revision, and so on. New models that have not been reviewed by the NRC include the "**New Model - Not Reviewed**" identifier. This will allow users to determine when use of a new feature may require additional NRC review.

Table C-2

Modifications Implementing New Features

Version/Feature	Modification	Description
MOD003.1		
Generalize Laminar Friction Model	mod_148	The user-specified laminar wall friction model input was generalized to allow up to nine sets of correlation coefficients on Card 05000X (X represents the correlation number). Reviewed model not changed. Input is supplied on Card 05000X (X represents the correlation number).
Clean Up Condensation Heat Transfer Calculation	mod_175	This modification replaces the use of volume variables AWGV and AWLV used in Subroutines QDOT32, QDOT33, and AWGV and AWLV used in Subroutines QDOT32, QDOT33, and QDOT34 used to weight the noncondensable and liquid heat transfer coefficient components with phase mass fraction. No input.
FTB Memory Allocation for 3D Kinetics	mod_176	If the SLIB switch R3DKIN is set, which activates the multidimensional kinetics option, LPOOL is set to 22,000,000 otherwise the pool is set to 750,000 words. Minimizes the need to update the code to obtain more space for three-dimensional kinetics. Does not affect any model results. No input.
Fix Edit Formats	mod_177	Fix edits: either extraneous debug messages or formats not big enough to handle all options in the new code version. Also, fix edit options not working as advertised (such as SMALLR or NED14). Revised input/output. No input.
Cross-Section Extrapolation on Moderator Density	armd_023	Read the DM extrapolation flag IXTRP from CDI file and pass it to the cross-section calculation subroutine. Use previously unused extrapolation flag from CDI file. Insures consistency with upstream core code. No other input.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
3D Kinetics Boundary Conditions	armd_024	This modification removed the limitation on the available boundary conditions. The "no return flux" boundary condition (IBC=2) is allowed if the PARCS numerical solution is used. Provides input path to allow user to request previously inaccessible boundary condition that was supported by the solution method. This feature is activated using IPURDU on the 670020 data card.
3D Kinetics Executable Size	armd_036	LINEID (size of FTB array for three-dimensional kinetics) was increased to 22,000,000 words. The PNM solution method uses fixed dimension arrays which were reduced somewhat to allow for a more reasonable executable size. Minimizes the need to update the code to obtain more space for three-dimensional kinetics. Does not affect any model results. No input associated with revision.
3D Kinetics Cleanup	armd_037	Removed unused edit routines left over from old code versions and old models that are no longer relevant in current code. Does not change results of active code. No input associated with revision.
Source Code Cleanup	enmd_005	Revise the SLIB77 switching logic to select options for various platforms easier to understand. Provide support features for COMPAQ FORTRAN and F2C. Extend use of code to new compiler. No affect on executable code.

Table C-2 (Cont'd)

Version/Feature	Modification		Description
MOD004.1 BWR Fuel Models	mod_196	1.	Provides capability to compute flows for active core and bypass channels. New input option to an existing model. Results from new input agree with previous results. Previous code version could compute flow splits, but only given core pressure drop. Code can now compute slow splits given upper plenum pressure and total core flow, which is input on 232XXX junction initial condition specification data cards.
		2.	Allows FIBWR style calculation of core support plate and lateral leakage flow paths. **New model - not reviewed.** Only affects results of previous code versions when the model is explicitly activated through input. Junctions are flagged as FIBWR leakage junctions by setting IPUMP on the 08XXXY Junction Data Cards (not used by default). FIBWR leakage model coefficient data cards, 6351XX and 6353XX, are also required when model is used.
		3.	Allows modeling of advanced fuel designs utilizing part length rods and water rods. Water rods and part length rods are modeled using standard RETRAN-3D volumes, junctions, heat conductors, and core sections. The flow split logic will compute the steady-state water rod flows, while the normal flow solution computes the flows for the transient solution scheme.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
BWR Fuel Model (Cont'd)		Part length and water rod geometry must be input using the channel model for three-dimensional kinetics (new input option) or via volume (05XXXY), junction (08XXXY), heat conductor (15XXXY), and core section (16XXX0) data cards. 4. Accounts for Reynolds number dependent grid losses. **New model - not reviewed.** Only affects results of previous code versions when the model is explicitly activated through input. Model allows vendor grid loss models to be used directly rather that specifying through the control system. This simplifies models and reduces the possibilities for errors and allows for consistency with upstream vendor codes. Pressure drops are typically compared to vendor results. The grid loss model option is activated at specific junctions by appropriately setting JCALCI on the 08XXXY Junction Data Cards (not used by default). Grid loss model coefficient data cards 2330XX cards are required when model is used.
Choking Model Improvements	mod_202	Use stagnation properties to evaluate functions. This revision makes the choking implementation consistent with the underlying theory for the various choking models available. Choking models are typically tuned by judicious choice of the contraction coefficient, which may need revision. All choking models now use stagnation properties by default. Options are available on 08XXXY junction data to approximate RETRAN-02 MOD005.2 and RETRAN-3D MOD003.1 choking formulations.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Choking Model Improvements (Cont'd)	mod_221	Replace isentropic HEM curve fits with table and interpolation. Table is more accurate and was used to generate curve fits used previously. No input and no way to run old curve fits.
Automatic Bypass Heating Model	mod_204	Used with channel model and three-dimensional kinetics. User convenience. New input simplifies use of existing model and reduces possibility of error. QBMDCH on 67000Y model option data cards and IBYPCH on 672XXX channel data cards must be supplied to activate option.
Control System Improvements	mod_205	Added super summer block - sum multiple inputs. User convenience that reproduces results of multiple cascaded summer blocks but with much less input, which can reduce to possibility for errors. New super block request must be supplied on 704XXX or 704XXXX cards to activate.
	mod_207	Increase number of input and control blocks. New user convenience that allows use of more control input and output blocks. Results from new input card series are the same as the old card series. New 703XXXX control input and/or 704XXXX control block description data cards must be supplied. No affect on compute results. New input for existing control system models.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Control System Improvements (Cont'd)	mod_245	Added super min and max blocks - multiple input maximum and minimum blocks. User convenience that reproduces results of multiple cascaded maximum or minimum blocks but with much less input, which can reduce the possibility for errors. New super block request must be supplied on 704XXX or 704XXXX cards to activate.
Variable Junction Inertial	mod_206	Allow for a variable junction inertia using a control block. User convenience that allows a user to specify a time varying inertia via the control system. INERTA < 0 must be specified on the junction description data card, 08XXXY, where it points to the control block that defines the inertial for the junction.
Option to Include Condensation Heat Transfer with Forced Convection Map	mod_218	Previously condensation heat transfer was only available with the combined (IHTMAP=1) heat transfer map. An option to make condensation heat transfer available with the forced convection map. User option to allow use of existing condensation correlations with the forced convection map. This option is activated by setting IHTMAP=2 on the 01000Y data cards (long of short form).
Option to Force Single-Phase Heat Transfer	mod_219	Allows specification of a Dittus-Boelter single-phase heat transfer coefficient for use with a given heat conductor. User option to force single-phase heat transfer. Use requires justification for specific application. This option is activated by appropriately defining IMCL or IMCR on the 15XXXY conductor description data card.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Option to Use a Multiplier with the Chexal-Lellouche Algebraic Slip	mod_220	Allows a multiplier to be applied to the slip velocity computed using the Chexal-Lellouche drift flux correlation. May be used to obtain a target mass inventory in a steam generator secondary. User option to force single-phase heat transfer. Use requires justification for specific application. The slip multiplier SLPMUL must be specified the each 08XXXY junction description data card for which the multiplier is to be applied. Default is 1.0.
Option to Use a Multiplier on Thermal Conductivity	mod_231	This is the same model that is available in the RETRAN-02 computer program. This is equivalent to the model available in RETRAN-02 and is activated by supplying a value for MULT on the 17XXYY conductor geometry description data cards.
FTB Dynamic Memory Allocation	mod_235	Use F90 feature to dynamically allocate memory at run time using a user defined memory size. Removes need to recompile the code to change the size of fixed arrays and need for a separated three-dimensional kinetics version. Simplifies use and has no affect on results. This option is always used. A default memory size is used when a -m isize specification is not supplied as a RETRAN-3D execution parameter.
A Run Time Option to Disable Screen Messages	mod_236	Run time option to turn screen message writes off. User feature. Option activated by providing a -ns execution parameter.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
BWR Separator Centrifugal Δp Term	mod_237	A new model option for an additional pressure drop term in steam-water separators. **New model - not reviewed.** Only affects results of previous code versions when the model is explicitly activated through input. The optional centrifugal pressure drop model parameters must be defined on the 60XXXY separator description data cards (model inactive by default).
Revised Stagnation Pressure Major Edit Term to be Velocity Head	mod_238	Removed the hydrostatic head from the stagnation pressure edit for junctions. Now contains velocity head terms only. User convenience. Aids users in interpretation of results. Not an option - always done.
Simplified Problem Dimension Input	mod_241	A new option that reduces the information supplied on the problem dimension input. The code computes the problem dimensions from the input file. User convenience. Only 1 to 12 parameters are now required on new 01000Y problem description data (user convenience). The old long form input is also supported.
NEMTAB Table Based Cross Section	mod_243	Option to allow use of NEMTAB table based option cross-sections with the three-dimensional kinetics model. User convenience. Eliminates need to convert NEMTAB cross-section tables to standard bxf format. Activated using IPURDU flag on Card 670020 and then supplying NEMTAB cross-section file.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Automated RETRAN-3D to VIPRE Interface	mod_248	Option to automate the transfer of boundary conditions from RETRAN-3D to VIPRE-01. User convenience for VIPRE-01 users. A boundary condition file for use with VIPRE-01 will be generated if a 02600Y data card is supplied.
U-Tube Steam Generator Initialization	mod_249	Auto initialization at off normal initial conditions using user-supplied target values for key primary and secondary parameters. This option is a user convenience where manual iteration using steady-state initialization is now replaced by an automated outer iteration to converge on target parameters. It does not change the base steady-state solution method, which is used by the outer iteration. The optional off rated condition initialization feature automates what can be done manually to obtain steady-state initial conditions. It requires input on 2360XY, 2370XY, 2380XY, and 2390XY data cards to activate the new initialization scheme.
Improved Error Message Information	mod_250	Provide more information on cause of error and time of occurrence. Remove redundant and misleading error messages. Provide a trouble shooting guide to aid users in resolving the error. User convenience - helps interpret and respond to error conditions. Always active - no associated input.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Contractual Requirement	mod_253	Adds copyright information to RETRAN-3D source library. Comments only - added to source code. No affect on results.
3D Kinetics Input	armd_042	This change allows the ¼ and 1/2 assemblies on reflective boundaries to be described as full assemblies in the CDI file User convenience.
User Convenience	armd_044	Modified the RETRAN-3D and FTB source code to use dynamic memory allocation. Removes need to recompile the code to change size of fixed arrays and need for a separated three-dimensional kinetics version. Simplifies use and has no affect on results. Memory size supplied as -m isize on execution command line.
User Convenience	armd_046	The screen write activation was removed form the three-dimensional kinetics source code library an implemented as an execution time option. User feature. Option to deactivate screen messages input as -ns on execution command line.
Contractual Requirement	armd_047	Added the EPRI Copyright Notice to source code. Comments only - added to source code. No affect on results.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
User Convenience - Short Form Problem Description	enmd_006	Subroutine NCARDS was written to count the number of cards in an INP free form input deck that lie within a specified range. Subroutine LINK was modified so it can be called with a flag that will optionally set the use flags for cards found. No affect on computed results.
User Convenience	enmd_008	The FTB initialization was revised in subroutine INITAL to use an allocatable array to define the FTB storage array size. Removes need to recompile the code to change size of fixed arrays and need for a separated three-dimensional kinetics version. Simplifies use and has no affect on results. Users can change the memory allocation by supplying the -m isize execution parameter. Has no affect on computed results.
User Convenience	enmd_009	Modify the RETRAN-3D source code so that screen messages are a run-time option rather than a source code option set at compile time. Remove unnecessary compiler/operating system dependent coding options and any associated SLIB77 switches. User convenience. Users can deactivate screen messages by supplying -ns on the execution command line. Has no affect on computed results.
Contractual Requirement	enmd_010	Added the EPRI Copyright Notice to source code. Comments only - added to source code. No affect on results.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
MOD004.2		
Junction Input Processing for Junctions Connected to Bubble Rise Volumes	mod_265	Added an option to automatically overlap volumes connected to bubble rise volumes and make junctions exiting the bubble rise volume vertical junctions. New input option to automatically use existing features and common modeling practice for overlapping bubble rise volumes. Activated using JVERTL=3 on 08XXXY cards.
New Enthalpy Transport Options Designed to Avoid Code Failures	mod_269	Added an option to the enthalpy transport model to determine if the junction enthalpy is realistic. If it is out of range or unreasonable, a donor or volume averaged enthalpy is used. New input option to automatically activate common modeling practice that previously required use of restart to deactivate enthalpy transport in problematic junctions. 1. New option is used by default. Deactivate using IENTRN=1 on 080000 card. 2. New selective deactivation option activated using ISELCT>0 on 080000 card. Requires 08000X card input.
New Option to Ensure Consistency with Auxiliary Calculations	mod_276	Added an option to allow use of either the Blasius or Moody form models for turbulent wall friction. User convenience that allows accepted vendor turbulent friction models to be used directly rather than approximating their results by tuning input l/d. Simplifies model use and reduces possibilities for errors and allows for consistency with upstream vendor codes. Pressure drops are typically compared to vendor results.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
		Activated using IFRIC=XXYY on 05XXXY cards or IFRICH=XXYY on 672XXX cards. Note: The laminar friction model coefficients previously input on 05000X cards are now input on 2332XX cards as a result of this modification.
Option to Reduce Memory Requirements fpr 3-D Kinetics	armd_049	Added an option to the three-dimensional kinetics model cross-section model to use less memory. User convenience. Automatically activated when new reduced size cross-section file (described in Volume 2, Section IV.2.3) is read. Older style cross-section files can still be used.
MOD004.3		
New Time-Step Control Algorithms	mod_287	Added time-step control logic based on rate of change in normalized power for point kinetics, one-dimensional and three-dimensional kinetics, and pressure. User convenience but also improves accuracy of the solution. The new algorithms provide control where users had to manually control previously by limiting the maximum allowed time-step size. These new algorithms are used by default but can be disabled by overriding the default algorithm multiplier via input on 03XXX1 data. When deactivated the time-step control algorithms will be the same as previous code versions.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Extend Vapor Properties Below 0.1 psia for Noncondensable Gas Flow Model Use	mod_291	Extended vapor properties for pressures from 0.1 to 0 psia. This included saturation enthalpy, specific volume and temperature. Extension of existing properties - not reviewed. Only affects analyses with vapor pressures below 0.1 psia. No input - always active for pressures < 0.1 psia
Optional Minor and Major Edit for Fuel Enthalpy	mod_296	Fuel enthalpy calculation and edit option added for cores/channels. New optional output fuel edit. Does not affect any other calculations. Activated by supplying fuel density on 119XXYY data cards.
New Output Edit Summary for Minor Edit. Variables	mod_297	Minor edit summary edited at the end of run for maximum and minimum values for minor edit variables along with time of occurrence. User convenience. No input.
Optional Two-Region Enthalpy Transport Model	mod_299	Two-region enthalpy transport model implemented to accurately account for heat transfer and enthalpy transport in a steam generator that is drying out. Removes limitation of current enthalpy transport model. **New model - not reviewed.** Activate the two-region model using a 008001 card.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Change Warning Message to a Fatal Error	enmd_012	When input data is encountered beyond Column 80, the condition is now treated as an error whereas it was previously a warning. User convenience - aids in detecting input errors. No input - always active.
Migration to FORTRAN 95 Compilers	enmd_013	Revised the interval timing to use the Fortran 95 intrinsic function cpu_time. Facilitates migration to Fortran 95 compilers. No input - always active.
Extended Input Processing to Simplify User Input	enmd_014	Free form INP processing function added to count number of data items on range of cards. User convenience. No input - transparent input processing aid.
MOD004.4		
Enhanced Pressurizer Major Edit	mod_314	Add edits for void fraction and power to H20 for liquid and vapor regions for PRZR volumes. User convenience. No input - transparent input processing aid.
Enhanced Trip Feature	mod_318	Added trip to initiate an end problem on user action. User convenience. Requires user input to set up the trip. Also requires user to activate the end-problem trip by copying a STOPRUN file to the working directory

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Pressurizer Initialization with Spray Flow	mod_331	Added a steady state initialization feature for a pressurizer with initial spray flow. User convenience. Requires additional input to activate the feature. Does not affect the balance or constitutive models used for the pressurizer. Computed initial conditions can be tested by running a null transient.
Steam Only Choking Option	mod_335	Added a choking option for vapor only flow. **New model - not reviewed.** Only affects results of previous code versions when the model is explicitly activated through input. The optional vapor only choking feature is requested using the X field for JCHOKE on the 08XXXY junction description data cards (model inactive by default).
SPERT Benchmark Cross Section Model Option	mod_339	Added a cross section model input option to allow SPERT test cases to be run a code change. The model was used during the NRC review. The coding required to read the cross sections supplied by the NRC was originally added to the code so the SPERT cases could be run. This cross section model is now available as an input option by appropriately providing the 6th word (IPURDU) on card number 670020. This modification requires use of armd_050 to the 3-D kinetics source code (new cross section model).

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Constant Block Initialization	mod_342	If a constant block is found, the initial condition is set to the gain; or COUT = CGAIN. User convenience. This will eliminate some initial conditions specification errors or constant blocks. This change is not optional and is always used.
Point Kinetics Subcritical Initialization Option	mod_346	Added an option to allow an initial subcritical reactivity to to be specified.
Actinide Decay Heat Multiplier Option		Added an option to apply the decay heat multiplier to the actinides. **New models - not reviewed.** An initial subcritical reactivity must be specified using RHOIN (new input variable) on the 140000 point kinetics data card. Otherwise, the initial reactivity defaults to the historical value of 0.0. By default, the actinide decay heat contribution is not multiplied by KMUL. To include KMUL in the actinide decay heat, a new actinide option must be requested using IACT=2 (new option) on the 146000 decay heat data card.
Heat Transfer Coefficient Multiplier Option	mod_347	An option was added to apply a user specified multiplier to a selected heat transfer correlation(s). **New model - not reviewed.** Heat transfer coefficient multipliers are only used when requested by using the optional heat transfer coefficient multiplier data cards 15000x.
New Grid Loss Two-Phase Multiplier Option	mod_348	Added a new grid loss two-phase multiplier option.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
		New models - not reviewed. The new grid loss two-phase multiplier option is commonly used in vendor and other two-phase thermal-hydraulics analysis codes. Its use may be necessary to insure consistency with other codes in a given methodology or analysis package. The new grid loss two-phase multiplier (Romie) is only used if requested by using the new LSET=3 option on the grid loss model data cards 2330XX.
New Wall Friction Two-Phase Multiplier Option		Added a new wall friction two-phase multiplier option. **New models - not reviewed.** The new wall friction two-phase multiplier option is commonly used in vendor and other two-phase thermal-hydraulics analysis codes. Its use may be necessary to insure consistency with other codes in a given methodology or analysis package. The new grid loss two-phase multiplier (Romie) is only used if requested by using the new LSET=3 option on the grid loss model data cards 2330XX. The new wall friction two-phase multiplier (Martinelli-Nelson) is only used if requested by using the new JTPMJ=4 option on the junction data cards 08XXXY.
SPERT Benchmark Cross Section Model Option	armd_050	Added code required to read cross section model input used during the NRC review. Originally, the coding was added to the code so the SPERT cases could be run. This modification must be used with RETRAN-3D modification mod_339.
FTB Revision for Linux	enmd_016	Revised LOCF to treat LOC results as an unsigned integer. This increases the maximum size that can be used for an FTB index.
MOD004.4f95		Includes all modifications associated with enhancements that were included in MOD004.4. Appendix D describes the conversion and validation processes used for MOD004.4f95.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
MOD004.5f95		
New minor edit variables and an optional plot file	mod_352	Added new minor edits for time in minutes and hours, flow in gallons/minute and liquid velocity. An option was also added to write a new output file for use with plotting packages. User convenience. The new minor edit request flag input is listed in the User's Manual – Volume 3 Section IV.4.0 for data card 0200YY. The input to activate the optional plot file is given the User's Manual – Volume 3 Section IV.4.4.
MOD004.6f95		
Licensing enforcement for Windows distributions.	mod_366	Revised the interface with license validations software used with Windows distributions. The revision allows use of a new model for uniquely identifying individual computers. It also uses updated software that supports 32- and 64-bit installations. User convenience. The licensing feature is only used with Windows distributions and has no effect on computed results.
Add restart capability for 3-D kinetics	mod_373	Previous versions of RETRAN-3D did not support restart for 3-D kinetics. This modification implemented changes required to allow models using 3-D kinetics to be restarted. User convenience Adding restart does not affect the results for RETRAN-3D problems with or without use of 3-D kinetics. Restarts that re-solve part of the solution domain are essentially identical to the original solution

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Add replacement cards needed to convert to the short form of the problem description data to the output file	mod_380	When the long form of the problem description data card is used, coding was added to write the replacement cards needed to convert to the short form. The replacement cards are written to the output file after the long form input in written to the output file. User convenience The short form input will reproduce the results obtained using the original long form input. The short form input simplifies use of the code and should eliminate some user errors.
Limt the flow reversal time-step to the minimum time-step size	mod_387	When the flow reversal time-step control algorithm computes a time-step size less than the minimum, an error condition is set, which generally leads to a time-step failure. The code was revised to use the maximum of the computed size and the minimum value. User convenience Allowing the code to use the minimum time-step size value allows the code to continue with a difficult calculation.
Improved the error messages written during input processing	mod_388	The error messages written during input processing were revised and written to both the output and error log files. Errors are individually identified and descriptions are included in Appendix C of the User's Manual. User recommendations are also provided for each error to provide users guidance on how to resolve the error. User convenience
Extended the list of plot variables	mod_396	A selection of additional minor edit variables were included as plot variables that are automatically written to the RETRAN-3D plot file (R3D_PLOT) . User convenience
Revised coding syntax	mod_407	The syntax used in a number of subroutines violated the strict enforcement by UNIX compilers.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
MOD004.7f95		
Plot file extension	mod_445	Added the time dependent junction area (AJNT) to the plot file content. User Convenience.
3-D Kinetics	mod_446	Revised the 3-D kinetics model so it can be use by other codes such as VIPRE-01. User Convenience. for other codes – no effect on RETRAN-3D
Plot file Enhancement	mod_455	Revisions were made so the R3D_PLOT feature is activated through input rather than the configuration file. Options were also added to use a full list of plot variables, a short list of variables or a user-defined list. Either form can also be expanded via user-supplied additions to the plot variable list. User Convenience.
Pressurizer Thermal Stratification Option	mod_456	Added the ability to subnodalize a pressurizer to account for thermal stratification. **New model - not reviewed.** The pressurizer thermal stratification model has not been reviewed but the components that comprise the model have, i.e., the two-region nonequilibrium model, the temperature transport delay time model, the heat conductor stack model, and the heat conduction model have been reviewed. Volume 4 contains validation results for the stratification model. The stratification model must be specifically requested using 610XXX two-region nonequilibrium data cards. The vapor region and a small liquid region are still modeled using the two-region nonequilibrium model. The liquid subnodes are individual volume that can use the temperature transport delay time model. Any two-phase subnodes are moved into the two-region node as soon as two-phase conditions appear. These volumes that are absorbed into the two-region node are disabled until they

Table C-2 (Cont'd)

Version/Feature	Modification	Description
		refill with subcooled liquid. As a pressurizer drains or flashes, the two-region node can occupy the full pressurizer domain. As the pressurizer refills with subcooled liquid, subnodes are re-form.
Steady-state Initialization Error Messages	mod_457	Revised all error messages associate with steady-state initialization so they are more descriptive and provide information to aid the user in correcting the error. Each error has a unique error message and corrective action that can be found in Appendix C of the User's Manual – Volume 3. User Convenience
Momentum Flux Model	mod_460	The original momentum flux model often leads to code failures when two-phase flow conditions exist. The momentum or flow equation solution can produce anomalous flows, driven by the excessive momentum flux pressure changes that cause the volume mass and energy inventories to deplete. The cause of the problem was due to the fact that the volume mass is in the denominator of the momentum flux calculation. This provides a positive feed-back effect on the flow reversal as the mass goes to zero. This causes the flows to accelerate and further deplete the mass, eventually causing the code to fail with a negative mass or extremely low pressure. A new formulation for the momentum cell boundary momentum flux values (average at center of volume) is obtained by averaging the junction momentum flux terms for the junctions entering and leaving the volume. It replaces the old model. The resulting steady-state momentum flux values are similar to those of the old model; however, the undesirable effects of the old model are eliminated for two-phase conditions where the mass in a volume is being depleted. The positive feedback problem observed in the original model was classified as a model limitation because there is no error in the implementation of the documented model. The problem is related to the undesirable effect the model

Table C-2 (Cont'd)

Version/Feature	Modification	Description
		formulation has on the numerical solution for the stated conditions. The revised (improved) model does not significantly affect the results, except for cases that no longer fail when the mass inventory in a volume depletes. Improvement to an existing model.
Water Packing Time-step Control	mod_461	A new time-step control algorithm was added to the iterative time-step control option that mitigates the anomalous pressure and flow spikes associated with water packing. It also eliminates pressure equation of state failures resulting from pressure spikes that exceed 6000 psia. By default, the water packing mitigation time-step control algorithm is active. It can be disabled using constants provided on the iterative time-step control constants data cards 03XXX2. User Convenience -- Improved Solution
Pressurizer Inter-region Heat Model Option	mod_473	Added a default interregion heat transfer model for the pressurizer. The default uses the maximum of a free convection correlation and a turbulent model when spray flow is active. The pressurizer input was also modified to set a default rainout velocity and default multipliers for heat transfer coefficients. The multiplier on heat transfer coefficient will allow accounting for uncertainties. The McAdams free convection correlation was used. It is used for similar applications in codes such as RELAP5 and CONTEMPT-LT. The Murphy turbulent model for inter-region heat transfer was in the code previously reviewed by the NRC. Its combination with the free convection correlation in the new implementation will account for proper heat transfer conditions when spray flow is active or inactive.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
		Free convection inter-region heat transfer coefficient - not reviewed. All of the pressurizer data comparisons in Section III.12.0 of the Applications Manual - Volume 4 use the new free convection heat transfer correlation for the inter-region heat transfer model. It is the recommended model and is also the default option. The analyses also use recommended constants, which are also the default values.
Automatic Subnodalization	mod_477	Added logic to automatically renodalize control volumes, junctions and heat conductors. The feature and be used for simple pipe volumes, specifically pressurizer volumes that use the stratification model. User Convenience This new option automatically subdivides pipe volumes into N new volumes, associated junctions and heat conductors if associated with the base volume.
Accumulator Model	mod_479	The accumulator model solves a gas region energy equation and a new pressure search. The model accounts for the effect of heat transfer from the vessel wall and liquid region to the gas region, **New model - not reviewed.** The accumulator model has not been reviewed. It must be intentionally activated by using an accumulator data card 620XXX to define the liquid level and vessel wall thickness and volumetric heat capacity,
MOD004.7.1		This code version was distributed as a replacement for MOD004.7 so the default interregion heat transfer is the same as for MOD004.6 and earlier versions. A new input format will be required to use the new best-estimate interregion heat transfer, thus requiring a conscious change to the input by the user.

Table C-2 (Cont'd)

Version/Feature	Modification	Description
Restore Backward	mod_491	Two-Region Nonequilibrium Model Input Compatibility This input was revised so the 610XXX input was identical to that used by MOD004.6 and prior versions. This prevents use of the pressurizer stratification and new best-estimate interregion heat transfer model unless specifically requested via new input data. Use of the pressurizer stratification model (mod_456) and/or the new interregion heat transfer model requires use of the 611XXX data card series. If a 611XXX card is used, the default interregion heat transfer will be based on the new free convection model added by mod_473. **New models added in MOD004.7 - not reviewed.** The stratified pressurizer model must be intentionally activated by using a 611XXX data card. Likewise, the best-estimate interregion heat transfer model must be intentionally activated using a 611XXX data card. Accumulator Model Input This was also revised so the 620XXX input data was as nearly the same as MOD004.6 as possible given that the previous polytropic expansion model is no longer available. This was done to provide backward compatibility with MOD004.6 and prior versions. Use of a control system to define the expansion coefficient is no longer allowed and expansion coefficient values for isentropic and isothermal expansions are the only values permitted. The polytropic expansion model coefficients are used to define the parameters for the new two-region solution so as to reproduce the expansion model requested. The input for the new two-region accumulator model added by modification mod_479 was moved from the 620XXX data to the 621XXX data. **New model added in MOD004.7 - not reviewed.** Neither the previous nor the current accumulator models have been reviewed. The model must be intentionally activated by using either a 620XXX or 621XXX data card.

Table C-3

Modifications Implementing Error Corrections

Code Version/ Modification No.	Trouble Report No.	Description of Change
MOD003.1		
mod_142	197	Corrected index error In INCHAN and INGEOM that affect retrieval of geometry information from CDI file.
mod_143	199	Coding was added to RETWRT to calculate POWER using the updated PNORM.
mod_144	209	The logic in subroutines CORQ, INMODH, and MODQF was modified to allow input on 145xxx cards to read when the three-dimensional kinetics model is used.
mod_145	210	This modification moves the line that saves DTC for the STF block within the ID loop, so all STF blocks will be correctly handled.
mod_146	225	Subroutine INTV had an incorrect test on the number of control block entries for card 07XXYY (if no NCG control block was entered, input processing failed). The incorrect test was fixed.
mod_147	213	In INCHAN the heated diameter was calculated after the area was expanded but the heated perimeter had not. The calculation was changed to use the expanded heated perimeter which is stored as the surface area per foot (ARCH).
mod_149	220	The original logic which skipped reading 6300XX cards for the MOC option for three-dimensional core channels was revised so the 6300XX cards are read.
mod_150	221	Remove the tests in subroutines MDOT and MDOTWF which prevent calculation of wall heat addition in the post-CHF regime with the five-equation solution.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_151	204	This update corrects several errors found in the implicit pressurizer model.
mod_152	228	The correction is to RESERV Filid 38 with a length equal to the max of Filid 5, Filid 6, and Filid 43, in EDITST, call TAPEBC and then delete the file.
mod_153	229	The problem was corrected by replacing the test for "IREAD .GT. 0" with "IREAD .NE. 0" in subroutine ADVFLO.
mod_154	198	The junction angle was removed and two new optional angles were added to the junction data card.
mod_155	230	This update causes the steady-state equation for DELTAV to be used all the time. This equation has a minimum value for DELTAV of 1.E-6, so the divide by zero error described above can never occur.
mod_156	231	The effective inlet density for the volume upstream of the tdv is computed using a flow weighting.
mod_157	232	Function C3FUNC was modified by replacing "XREF = -REF" with "XREF = ABS(REF)". This fixes the problem and does not change the solution for normal countercurrent or cocurrent down flow.
mod_158	174	The problem was traced to the section of coding in XANDH , which adds h(j) linearization for negative fill junctions was found to be in error and was corrected.
mod_159	222	Several errors in subroutine GENTRN were corrected.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_160	233	Logic in INSLAB was added to calculate the number of user defined conductors (the number of 15XXXY cards supplied in input). Also, a check was added to insure that 9 or fewer conductor stacks were supplied (per the User's Manual).
mod_161	234	This modification extends the acceptable range for Z (assuming IMCL = 1YZ) from 2 to 5 to cover values for the Catton-Swanson correlation.
mod_162	165 205	The option to use the dynamic gap conductance model with multi-dimensional kinetics option was added to the code (it had been inadvertently omitted).
mod_163	224	The momentum flux terms are revised when multiple junctions are located at the exit or inlet of a volume.
mod_164	236	The input processing was modified to detect if NBORON (W2-I on Card 670021) is used to set initial boron PPM or select the critical boron search option prior to processing the generalized transport option.
mod_165	235	Several errors in the flow propagation and pressure initial condition specification were corrected.
mod_166	211	Logic was added to subroutine INVOL to test for a bubble rise set (IBUB > 0) if the volume is IX1 for a mixture level trip (|IDSIG| = 5).
mod_167	223	Logic was added to subroutine ICVOL so that when the initial calculation of FMAS and GASM are made from the specified conditions, LIQM, LIQL and LIQV are also defined.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_168	239	Subroutine ENTRAN was modified so that if the calculated junction quality (XJUN) is 0 or 1, the "not slip" flag (NTSLIP) is set and a jump from the iterative loop.
mod_169	022	When the searched pressure exceeds PCRIT, resetting it to a value lower but close to PCRIT allows the search can go on and converge. This modification was made.
mod_170	237	The tests for net steam flow in the vapor region of a bubble rise volume indicating a possible negative separation velocity or a negative partial bubble density are trapped and reported as an error only if they occur on the very last steady-state iterate.
mod_171	---	Modification not included.
mod_172	241	Logic is added to subroutine JSVEL for negative fill junctions to use a forward difference solution.
mod_173	243	The derivative of the centrifugal pump head with respect to the diagonal junction flow is added to the FSUBK variable and the correct junction weighting term was included.
mod_174	244	The logic in subroutine JSVEL for the Chexal-Lellouche algebraic slip model was revised to use the horizontal form of the model when IFRJ > 10.
mod_178	245	The revision uses the forced convection heat transfer results (obtained prior to calling the condensation model) rather than terminating the problem when the condensation correlation fails to converge.
armd_020	---	Not included.
armd_021	---	Not included.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
armd_022	206	The trouble is fixed by starting the guessed adjoint eigenvalue from steady-state eigenvalue, instead of starting from 1.0.
armd_025	214	Logic to support the various NFREEZ options was in place but the variable LFEED overrides the options and forces an "ALL or NOTHING" condition. Removing the reference to LFEED, all NFREEZ options become available.
armd_026	215	The modified code divides the sum of the power densities by the number of powered axial planes to get an average.
armd_027	217	All sets in IDAFLX are processed (PROCES) and then the call to TRNCAT is replaced with FTBCLS then IDAFLX has the correct number of sets when EDITD9 is called.
armd_028	218	Added XISP isotope number densities to BXF file a CORETRAN modification). Added revision to allow RETRAN user to select number densities from BXF file (default), equlibrium XISP or zero XISP.
armd_029	219	Cod modified to assume default of 0 rather than use CDI values. User has the ability to set all NED flags on card 670010.
armd_030	227	The historic rodded fraction is set to zero. This effectively eliminates the adjustment from historic rod cross sections, so only the instantaneous contribution is considered.
armd_031	236	The input processing was modified to detect if NBORON (W2-I on Card 670021) is used to set initial boron PPM or select the critical boron search option prior to processing the generalized transport option.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
armd_032	195	The code was modified to detect ESHIFT and CETAK for some possible unreasonable ranges that could lead to failure of neutronics calculation. More checking was added to avoid other unreasonable values.
armd_033	238	The code was changed to pass unit 14 to TRAN. This allows TRAN to write the restart file to the correct unit. Also, the arguments passed to TRAN from DRIVE2 were in the wrong order.
armd_034	164	Corrections were made to avoid divided-by-zero and negative index errors.
armd_035	242	This update moves a write statement back where it originally was prior to a previous update. Other changes remove local variables that are never used.
enmd_04	216	Subroutine INP was modified to scan the input for TABs and replace them with blanks. If data, not comments, are beyond column 80 a warning is issued.
MOD004.1		
mod_179	246	Fix control block CP2 floating point error.
mod_180	250	FTB file definition moved up one line.
mod_181	251	Correct VSLPV calculation for volumes with multiple junctions.
mod_182	252	Correct ICVOL to edit the correct transport volume mesh data.
mod_183	256	Fix flag which activates noncondensable gas logic in EPRIDV.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_184	247	Fix error in scratch space reservation for flow splits case.
mod_185	257	Correct logic in SSSEP that overrides local energy balance when P is input for a two-region nonequilibrium volume.
mod_186	255	Count the number of entries on the material property data cards to calculate how much memory to reserve.
mod_188	202	Correct enthalpy error when mixture level passes through junction; correct five-equation model errors when critical pressure is reached.
mod_189	261	Correct the error for enthalpy transport with flow splits.
mod_190	203	Correct the logic when single phase exists in two-region nonequilibrium model.
mod_191	262	Add derivatives of slip velocity with respect to p, x, and w for ISFAG = 2. Revised relaxation scheme for steady-state slip velocity.
mod_192	264	Correct errors in NC state routine and NC condensation nonconvergence.
mod_193	265	Smoothing logic in the mass transfer model changes the transfer term from steady-state to transient.
mod_194	266	Several equilibrium thermodynamic initial condition options were not included for five-equation volumes.
mod_197	269	Added test to ensure that countercurrent properties are not use to compute cocurrent slip velocity.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_198	270	The call to ENTHAL was replace with a Newton-Raphson iteration to solve for enthalpy.
mod_199	271	Test for flow split option before defining do loop indices.
mod_200	263	The branch junction model added in MOD003.1 was revised to use the single junction form for separators.
mod_201	274	Correct logic to trap negative relative volume.
mod_208	277	Remove KMUL from Actinides (consistent with Theory Manual).
mod_210	278	Modified to use "to" volume enthalpy for junction enthalpy when flow is zero. Fixed a restart error. Another modification turns the transport delay model off when a junction flow is two-phase.
mod_211	280	Bypasses the liquid volume convergence test for single-phase volumes.
mod_212	281	Revised to use consistent time level values for the wall temperatures and replaced the iterative solution with a linear approximation.
mod_213	282	Revised the limiting void fractions, add logic to neglect countercurrent flow for low void fractions, and added a cut off to neglect slip for void fractions > 0.999 when the Chexal-Lellouche model is used.
mod_214	283	Use the donor volume density for a TDV momentum flux.
mod_215	284	Add option to initialize same as RETRAN-02.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_216	285	Limit the junction area such that (0 <= ajunt <= ajun).
mod_217	286	Apply density ratio to torque difference.
mod_222	273	Reset pump stop/reverse flag trip.
mod_223	276	Replace intrinsic SIGN with function SYGN.
mod_224	287	Revise scratch space reservation.
mod_225	290	Removed definition of phase=2 in common path.
mod_226	291	Delete extra call to TRNSPT.
mod_227	292	Add values in INVOL for correct interpretation by ICVOL for P>0, T=0, H=-1, ZM=ZVOL, Separated Volumes.
mod_228	293	Revise zero flow test for volumes to use (wjsum1+wjsum2) rather than volume average flow.
mod_229	294	Revise Bernoulli term in momentum equation to include cosine of angle.
mod_230	295	Added logic to define the boundary temperature to local conditions value for specified HTC.
mod_232	296	Removed a fix-up path that uses a hardwired value of the critical specific volume.
mod_233	299	Revise local conditions model for setting bulk fluid temperature for nonequilibrium volume.
mod_234	300	Correct consistency check for Chun and Seban conductor stack.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_239	302	Correct an error in input checking logic. Also, cleaned up some complicated branching logic.
mod_240	305	Eliminate logic that over indexes array during three-dimensional kinetics input processing.
mod_242	306	Correct slip calculation for negative fills
mod_244	307	Correct index problem in the low power SG initialization model.
mod_246	308	Remove unnecessary error condition from bubble rise velocity calculation
mod_247	309	Correct enthalpy error on 0th iteration for a low the low power SG.
mod_251	304	Removed junction area change term from the inertial flow estimate used with the choking model.
mod_252	301	Smooth positive slip velocity to zero for low void.
mod_254	297	Add logic to limit velocity used in stagnation pressure and enthalpy.
armd_038	249	The default values for the PNM numerics floating point control parameters were defined as double precision values to eliminate erroneous initial values on the Win95/NT platform.
armd_039	258	The subroutines that contain the IFXISP flag need to be revised so the XISP options match user's manual. The modification includes correcting the logic, revising the error message, and resetting the default option in case the input IFXISP is out of range.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
armd_040	170 254	The variables KAPPA and OMEGAM were modified to include a dependency on core nodes as well as groups. Common FKIN (COMDECK KIN), and subroutines PRECURS, and SETTFSP were modified.
armd_041	259	Subroutine INCDIA that initializes the 670040 card was revised by giving correct default values to the card options.
armd_043	276	This modification replaces references to the intrinsic functions SIGN and ISIGN with the functions SYGN and ISYGN, which provide f77 compatible handling of signed zeros.
enmd_007	276	Functions SYGN and ISYGN were added to the environmental library to replace the intrinsic functions SIGN and ISIGN to obtain f77 compatible handling of a signed zero.
MOD004.2		
mod_255	310	Added logic to skip loop used to identify junctions exiting a bubble rise volume when bubble set isn't used by a volume.
mod_256	311	Revised the beginning card number for each of the card sequences used to determine a dimension.
mod_257	315	The linear form of the inertial flow solution that is used to determine if choking occurs was modified to use a quadratic form.
mod_258	048	Modified use of an energy equation convective term derivative for low void fraction and positive slip during steady state. Skipped bulk mass transfer evaluation for subcooled liquid and superheated liquid.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_259	316	Revised an output format statement.
mod_260	317	The internal time-step control logic uses the maximum power change to control the logic path followed. The power change was being evaluated at the end of the 1st stage of the Runge-Kutta solution. It was moved after the solution was completed.
mod_261	319	Coding in subroutine INVOL to increment MTDV for every IREAD<0 was removed. Logic was added to subroutine SETDIM to determine the number of tdv BCs from tape by scanning the volume input data.
mod_262	320	The coding was revised to eliminate the use of the undefined variables.
mod_263	322	Certain values used for thermal conductivity multiplier cause code to fail on PC.
mod_264	324	In the Lellouche subcooled boiling model used with the five-equation model, a flashing model is activated when the liquid superheats. The derivatives used for this condition were incorrect and caused the solution to fail to converge. The derivatives were corrected and new derivatives were added.
mod_266	327	Corrected an error that gave an infinite loop when a bad enthalpy is input for a pressurizer volume.
mod_267	330	Option to perturb one-dimensional cross-sections did not correctly reset the perturbations for inner loops. Also enabled option to perturb cross-section and continue.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_268	325	Add logic to trap TDV input errors and provide informative error messages. Eliminate unexplained FTB errors.
mod_270	332	Added capability for more than 3 regions in fuel geometry. Revised input processing to use region number and material identifiers from CDI file.
mod_271	339	An error message was added to the ERRLOG file when a surface temperature less than zero is computed in QDOT28 and QDOT25.
mod_272	334	Subroutine JSVEL was modified to eliminate logic that limited the dVs/dt derivative. When activated, the logic also erroneously redefined the slip velocity.
mod_273	336	Subroutine JSVEL was modified to correct logic that smoothed the slip velocity toward zero at low void.
mod_274	337	The grid loss coefficient was multiplied by the ratio of the mixture and liquid densities.
mod_275	340	Output formats for several routines were revised.
mod_277	342	The normalized area is allowed to be >1, consistent with the table option and RETRAN-02.
mod_278	343	A test for two-phase conditions above and below that bypassed the countercurrent flow initiation logic was eliminated.
mod_279	344	Coding was revised to allow the debug edit option to be used.
mod_280	345	A faulty convergence test indicated that the pressure EOS was not converged when in fact it was.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
armd_048	320	The coding was revised to eliminate the use of an undefined variable.
MOD004.3		
mod_281	347	Subroutine INDATA was modified to correct logic that redefines the value of NUMRCS.
mod_282	348	Subroutine SLIP was modified to correct the error and allow for better code flow in the subroutine.
mod_283	350	Logic was added to the flow estimate solution in subroutine ADVFLO to trap degenerate cases, e.g., no friction, and then calculate the appropriate inertial flow estimate.
mod_284	351	Logic was added to subroutine ENTRAN to correct an error in the enthalpy transport deactivation option.
mod_285	352	GAPCDI was renamed IGAPCD to maintain the variable as an integer type.
mod_286	272	Derivatives modified/added for CX term, junction enthalpy, and convection term and nonequilibrium volume energy equation.
mod_288	353	Corrected search logic for TDV volumes from tape.
mod_289	354	Initialize variable BUFR in subroutine INMOC to 0.d0 to make it re-entrant.
mod_290	356	Added new function DTSPDP to compute partial of TSUP wrt pressure.
mod_292	346	Pressure search for volume with air, liquid and very low amount of vapor is added. It requires use of mod_290,mod_291 to work.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_293	200	Equation-of-state solution modified. Fluid properties and derivatives after convergence changed.
mod_294	358	Corrected a number of derivative terms and revised the slug to annular-mist transition to use a linear void interpolation of the log10 of the bounding interfacial area * HTC values. Revised the numerical solution to solve for mixture quality and noncondensable quality.
mod_295	359	Corrected steady-state edit of input values of bubble velocity and gradient.
mod_298	361	Number of iterations that compute pressure or loss coefficients are limited to square of number of junctions.
mod_300	362	Warning message for discontinuity in enthalpy function at 850 psia removed (no discontinuity exists in current saturation enthalpies).
mod_301	363	Mixture temperature changed from vapor temperature to quality weighted average of liquid and vapor temperature.
mod_302	364	Mixture level is changed corresponding to volume height for bubble rise volume when volume height and mixture level are input equal.
mod_303	365	Logic was added to edit (major) the FIBWR related junction path flags when the FIBWR model is used without the channel model.
enmd_011	349	The modification changes the type of variable FTBRCL from real to integer.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
MOD004.4		
mod_304	366	Revised the logic used to call subroutine CCFPRP.
mod_305	367	Defined variable SPVAP when the vapor phase donor volume is a bubble rise volume.
mod_306	368	Revised code to limit the magnitude of the argument to the exp function.
mod_307	369	Fixed incorrect logic test for heat conductors associated with a steam generator.
mod_308	370	Revised minor edit summary to include values between edit times.
mod_309	371	Fixed error where use of 15xxx0 cards (invalid) leads to a misleading error.
mod_310	372	Corrected error where code hangs-up when a RESTART job uses a Super Summer block.
mod_311	373	Corrected error where bubble velocity remains constant after time zero when controlled by control system.
mod_312	374	Corrected error where the accumulator liquid mass minor edit is always zero.
mod_313	379	Corrected error for implicit two-region noneq. model where a volume is initially liquid, doesn't develop two regions when volume goes two-phase.
mod_315	378	Added logic to skip the section of code that turns off enthalpy transport if already deactivated.
mod_316	377	Added logic to correct LIQV and LIQL edits for an accumulator. Also corrected error where subroutine STATEW was called from GENOPT rather than STATAC.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_317	380 382	Added logic to skip Thom if void fraction >= 0.8. Also added logic to prevent condensation if vapor is subcooled (pressurizer model).
mod_319	381	Corrections to implicit two-region nonequilibrium model for rainout and wall condensation.
mod_320	383	Corrections for the heat transfer mode printout in the major edit for specified HTC.
mod_321	385	Logic was added to disable countercurrent flow if the junction is located below both mid-points in the connecting volumes.
mod_322	386	During the first iteration some of the pressures may not be known so logic was added to subroutine FILL to skip the portion of code that returns an error if the pressure for a negative fill is out of range.
mod_323	387	Logic was added to subroutine INJUN to test if the junction is a fill junction before writing the warning message.
mod_324	389	Added a call to subroutine BOY_FLAGS added in enmd_015.
mod_325	388	Moved the error check for energy flow reversal in steam generators and feedwater heaters from SINITL to STSTAT.
mod_326	390	The pressurizer liquid region quality (NELX) is calculated based on bubble mass and liquid mass. This logic is removed and NELX is forced to be zero during steady state initialization.
mod_327	384	Tests on the liquid region energy and vapor region energy (normalized to the total energy in the volume) for establishing a two region volume in subroutine WAT.8 were modified to account for the affect of the time-step size.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_328	391	Logic was added to skip calculation of the two-phase partial derivates for volumes initialized with a pressure greater than or equal to the critical pressure.
mod_329	393	Logic used to select the coefficients for evaluating the Wilson bubble velocity was revised to be independent of bubble velocity.
mod_330	392	The data initialization for variables ONE and ZERO were deleted since they reside in the UNITS include file, which is already initialized.
mod_332	395	Changed the beginning card number used to locate heat conductors.
mod_333	376	Subroutine INSTGN is modified so that when index for junction sequence list IDXJNR is updated, a new index to the volume solution order file is used.
mod_334	394	Correct storage for super summer blocks.
mod_336	396	Corrected compiler errors for IBM.
mod_337	397	Corrected the spelling of derivative DASADA in subroutine argument list.
mod_338	398	A line of inadvertently deleted code (mod_305) was reinserted.
mod_340	399	Reported problem was due to nonconverged solution. Subroutine TSTP.F was modified to account for the limit on the normalized power change time-step control algorithm for the 1D and 3D kinetics options, eliminating unnecessary time-step size reductions.
mod_341	401	A check was added so that if file 75 is not reserved then swapping between file 75 and file 40 is not performed.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_343	402	The index for saturated liquid enthalpy was changed from I to L.
mod_344	404	Subroutine INJUN was modified to issue an error message if the automatic overlap option (JVERT = 3) is requested for a fill junction.
mod_345	403	The variable 'DT' is passed from subroutine PRZR to WAT8.
mod_349	400	Subroutine DEFORM.f was generalized to use the fuel pellet geometric description for each axial node and to also allow multiple regions in the fuel pellet. An error was corrected in subroutine GAPHTC.f for the ratio of mean fuel surface roughness to wavelength equation.
mod_350	405	An error that set the time derivative of the vapor continuity eqn. to zero was corrected.
mod_351	403	Logic was added to subroutine HTRC to prevent use of a condensing heat transfer correlation if the wall heat flux is positive.
enmd_015	389	Added new subroutine BOY_FLAGS to dynamically detect the Little or Big Endian processor type and set appropriate flags.
enmd_017	392	The data initialization for variables NN, which is in common, was changed such that a local variable IZ is used to initialize NN (IZ initialized via data statement).
MOD004.4f95		Includes all modifications related to error corrections that were included in MOD004.4. Appendix D describes the conversion process and validation for MOD004.4f95. This code version was not formally released.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
MOD004.5f95		
mod_353	407	A test was also added to set an error return code indicating that the surface temperature is essentially equal to the saturation temperature (condensation not appropriate), which results in a convection regime being used . Errors in two derivatives used when smoothing between condensation and convection were corrected.
mod_354	408	A new variable, c0dv, was added to module m_junctions. This insures that c0old (occasionally used to compute c0 when a weighted solution is used) is reset to the correct value when a time step is reset.
MOD004.6f95		
mod_355	409	The first element of the rdata array was not initialized to zero before call to inp2 from incnst. Coding was added to initialize the array element to 0.
mod_356	411	The space_time field in the me_list derived data type was not initialized for problems not using the 1-D kinetics model. Coding was added to m_minor_edit_search to initialize the field.
mod_357	412	Coding was revised to update the value of vliq that is a minor edit variable.
mod_358	416	Interface module s_water was modified to define the type of function dtspdp as kind=8.
mod_359	413	The length of a character variable used to identify dynamic restart blocks used to provide time dependent boundary conditions (form restart file), was revised to be 12 characters. This is the length used to define the block names on the restart file.
mod_360	414	The memory allocation for the Purdue thermal-hydraulics solution was increased in module m_sparse_mapping.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_361	415	An option in the 3-D kinetics model was inadvertently omitted during the Fortran 95 conversion. It was added.
mod_362	410	A warning message was added when the 3-D kinetics model computes a negative flux. The message recommends that the NEMCY input parameter be revised to improve solution convergence.
mod_363	417	Subroutine 'contrl' was modified to add a check to insure that the BOXX has been allocated before it is reset when iterative solution is not converged.
mod_364	418	During steady state, for ISFLAG=3, the junction orientation multiplier was added to the liquid and gas velocity calculations.
mod_365	419	Revised code to allocate space for a minimum of one volume, irrespective of NCFLOW value.
mod_367	420	The calculation of derivative DVDPHG was modified is module water.
mod_368	341	An error was corrected in the power normalization for the decay heat model solution used for the 1-D and 3-D kinetics options. The code was also revised to use same decay heat model solution for point kinetics.
mod_369	422	Revisions made in mod_326 were undone. New coding was added to correct the original problem.
mod_370	423	The transition quality used when the heat transfer mode switches between Collier and Siddique in the presence of noncondensables was changed.
mod_371	421	An equation in the gap contact conductance evaluation was revised to match the documentation.
mod_372	375	Logic was revised that is used to activate and deactivate bubble rise volumes in a stack as per the mixture level.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_374	427	Dummy arguments NXPT and NYPT in subroutine edtd10 were incorrectly specified as 2-D arrays. The code was revised to treat them as 1-D arrays.
mod_375	428	Coding inadvertently deleted in BICG during Fortran 95 conversion was restored. Boron transport mapping to neutronics mesh was corrected. Several editing errors corrected.
mod_376	429	The allocation for the old control rod mapping array was modified
mod_377	425	Subroutine ENTRAN was modified to use state derivatives already available from the state solution rather than recalculating them (incorrectly).
mod_378	355	Modified the 5-eqn air-water pressure search for steam flow. Also modified and mass transfer derivative for inter-phase heat transfer.
mod_379	430	The array size of RHSI was increased in subroutines insolv and inslv2. The calculation for determining THETA in the critical velocity calculation was also modified to prevent an out-of-bounds index from being calculated.
mod_381	431	Correct short form problem dimension processing so printer plots work.
mod_382	432	Added a lower limit of 1.e-6 to the mass flux used to evaluate CHF.
mod_383	433	Corrected the argument list for calls to bubint & bubinc in ststat. Corrected error logic and message in bubint.
mod_384	434	Skipped all of the logic that checks trips supplied with a RESTRT input deck when no trips are provided in restart deck.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_385	435	Added logic to skip writing the trip summary edit when no trips were generated during a run.
mod_386	436	For the IPURDU flag, any value in the units place is treated as if it were input as 1.
mod_389	438	The slip multiplier is incorrect when it is defined using a control block. The coding was corrected by referencing the COUT variable for the block used.
mod_390	437 439	The friction loss between the volume and junction included a sign of the flow. Since the donor volume was use the sign of the flow was removed. Comments changed in stpm and an invalid index was corrected in xncalc.
mod_391	444	When steady-state initialization computes FJUNR, it was stored in FJUNF for the next junction. Logic was updated to correctly store FJUNR.
mod_392	445	Corrected a restart failure when the long form problem control and description is used in the original problem. Added generalized transport model concentrations and nonconducting heat exchanger power to the restart file dynamic blocks.
mod_393	441	An averaging relaxation method is added to the iterative solution for the liquid volume change (part of the pressure search) in the explicit pressurizer solution.
mod_394	443	The void fraction limit for transition from slug to the annular flow regime changed to 0.75.
mod_395	446	The index of a pointer into the reactivity tables is corrected for the region wise reactivity tables.
mod_397	447	Subroutines called with argument lists that are longer than the called subroutine dummy argument list were revised to match the list for the called subroutine.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_398	442	The void fraction limit for transition from slug to the annular flow regime changed to 0.75. The entrainment correlation solution was also revised to improve convergence.
mod_399	440	Subroutines called with argument lists that are longer than the called subroutine dummy argument list were revised to match the list for the called subroutine.
mod_400	448	Trip summary was corrected to use correct control system indexes.
mod_401	449	The flow reversal time step size changed.
mod_402	450	The index used to print the conductor stack information was changed.
mod_403	451	Minor edits that used the trip number for the region were removed. The array containing the trip ID numbers used for creating the plot file was changed.
mod_404	452	Local arrays changed to dynamical allocation.
mod_405	453	Logic was added to prevent an infinite loop if a restart file is run without saving a new restart file.
mod_406	454	An error was corrected where the timing edits for a restart run were being printed out in the wrong time intervals.
mod_408	456	Changes were made to a couple of error messages and a flag was added to determine if the errors are input or run-time errors.
mod_409	458	Corrects an error involving reset trips in subroutine tedit.
mod_410	457	DNBR was added to the minor edit search. An error was correct in subroutine dnbm where incorrect dnbr and quality values were being calculated.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
MOD004.7f95		
mod_411	459	Corrects an error where both the forward and reverse loss coefficients are computed (same value) when the junction initial condition specification data card 232XXX is use to identify junctions where the loss coefficients are to be computed. Logic was also enhanced to detect when a request is made to compute a loss coefficient with zero flow, which is not a valid request.
mod_412	460	Corrected an error in the low power steam generator initialization feature that occurs when a volume is a boundary for several heat conductors. Also corrected a restart problem that indicated that the 2350XY data cards were not use during restart.
mod _413	461	The restart file structure was revised to allow certain variables to be redefined at restart time using generalized restart.
mod_414	462	The accumulator model minor edit feature was revised to use the control volume number as the region number for accumulator minor edit variables.
mod_415	463	A sign error was corrected in the slip energy convection term for the explicit two-region nonequilibrium volume that resulted in the vessel upper head cooling rather than going two phase.
mod_416	465	An error was corrected where the interval between edits was incorrect for restart problems.
mod_417	464	Modifications to subroutine invol that were tested as part of mod_388 were inadvertently omitted from the version control system. They were added by this modification.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_418	466	Corrected an error in the ISFLAG=5 slip option that resulted in a code failure due to an index that is out of bounds.
mod_419	467	Corrected an error indicating that the executable has stopped working, where the error was in the memory allocation for the 3-D kinetics XISP model.
mod_420	468	A memory allocation error in the 3-D kinetics model was corrected.
mod_421	469	Modified the low power steam generator initialization feature to recognize closed valves in tube volumes used to model break junctions for steam generator tube rupture models.
mod_422	471	Revised the minor edit summary output formats to account for large component numbers (i4 >> i7).
mod_423	473	Corrected problems encountered during restart.
mod_424	470	Corrected a problem where the enthalpy transport model does not converge during steady-state initialization.
mod_425	474	Corrected an error in the frequency at which edits are written.
mod_426	472	An error that occurred when attempting to compute saturation properties for a volume filled with dry air was corrected.
mod_427	476	A correction was made to restart for the two-region nonequilibrium model to allow model constants to be changed during generalized restart.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_428	477	An interpolation heat transfer mode was added to smooth the transition from the Schrock-Grossman correlation to the Dougall-Rohsenow correlation for qualities greater than 0.95. This eliminates an oscillation that appeared previously.
mod_429	480	An error was corrected for 3-D kinetics system problems where the conductor geometry cards were not processed, resulting in and error.
mod_430	478	The number of iterations allowed for the iterative solution of the Chun-Seban condensation correlation was increased, resolving a convergence problem in the pressurizer environmental heat loss.
mod_431	481	The logic used to write the error message was revised to correct the error.
mod_432	479	The error was corrected, allowing multiple boundary conditions to be obtained from a previous restart file.
mod_433	462	The logic for processing heat conductor stacks was revised for 3-D kinetics models using the CDI file to define core stacks and input for other stacks.
mod_434	483	An error was corrected that resulted in steady-state not failing when a negative loss coefficient was calculated.
mod_435	484	An error that resulted in calculated loss coefficient (those specified on the 08XXXY and 672XXX cards) not appearing in the initial condition summary table.
mod_436	486	The plot file option was modified to allow for larger component numbers. Control block flags COUT and CBLK were also changed to CU and CB for large control block numbers.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_437	487	Replaced a block copy with a do loop to eliminate a stack over flow for problems with large input decks.
mod_438	475	Eliminated an error message generated by the subroutine that times the execution of other subroutines that results from a processor that is so fast that it executes the subroutine and returns before the clock is updated.
mod_439	485	Added the capability to include temperature transport delay time mesh enthalpies as minor edit variables.
mod_440	488	Corrected several error messages that had in core write errors the resulted in a fatal runtime error when encountered.
mod_441	489	The pressurizer model was corrected so the vapor region will reappear as soon as the liquid region flashes after filling.
mod_442	497	The error message processing for some memory allocation errors and input errors were revised to eliminate possible errors in the error processing.
mod_443	495	An error was corrected in the enthalpy transport deactivation option that resulted in enthalpy transport not being deactivated even though requested.
mod_444	491	Revisions were made to the equations of state used to initialize separated volumes containing noncondensible gas, eliminating the reported error.
mod_447	490	Corrected an edit frequency problem when the requested edit frequency is not an integer multiple of the maximum time-step size.
mod_448	492	Corrected an edit frequency problem that occurs for long duration transients (>20,000 sec.).

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_449	493	The implicit pressurizer model was revised to correct a situation where the wrong volume number was used for the vessel wall boundary condition evaluation.
mod_450	498	An error correction resolved the problem that indicated incorrect trip was provided even though none was provided in the restart deck.
mod_451	499	An error that inadvertently cause printer plots to be active even though not requested was fixed.
mod_452	501	Revisions were made to the code to eliminate warning messages, e.g., unused variables and passing the wrong type variable through the argument list, that were generated when the warning message feature was turned on in Visual Studio.
mod_453	496	A warning message from the generalized transport model was revised to be more informative. The activation limit was also changed so the message will be encountered less frequently.
mod_454	500	Corrected minor edit input processing so it writes an informative error message when 3-D kinetics minor edit requests are made for none 3-D cases rather than terminating with no message.
mod_458	504	Corrected an error in the iterative time-step control that skipped updating volume properties when a time-step is reset and resolved with a smaller time-step size.
mod_459	503	Corrected an output error of the variable (minor edit flag) used as input to the super summer control block.
mod_462	506	Corrected an error message that identified the wrong volume as being the cause of the error condition.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_463	507	Added logic to prevent using a negative pressure in the evaluation of the Chexal-Lellouche drift flux correlation. Prevents NaN from propagation through the matrix solution for mass, energy, flow, and slip velocity.
mod_464	509 510	Improved the implementation of JVERTL=3 smoothing to prevent geometry inconsistencies that result in steady-state not converging.
mod_465	512	Modified the free convection boiling curve so heat transfer coefficients less than 5.0 are allowed.
mod_466	519	Corrected syntax errors that are encountered with the IBM Fortran complier.
mod_467	533	Corrected a limitation in the pressurizer stratification model where is was assumed that subnodes would only have one inlet and one exit junction. Some models use dummy junctions to model pressure tapes. Also added approximate thermodynamic conditions use in minor edits for inactive subnodes.
mod_468	527 529	Corrected input processing errors for the slip multiplier, SLPMUL, and junction inertia, INERTA, from a control block.
mod_469	523	Corrected calculation of GPM* minor edit so it is available at time zero for control blocks.
mod_470	528	Corrected restart so DLY blocks work correctly for restart runs.
mod_471	522	Corrected an error in input processing that could result in inadvertently setting IHQCOR > 0.
mod_472	532	Revised the explicit pressurizer solution so dp/dM and dp/dU are calculated directly from known water properties rather than using the HEM pressure search which can fail to converge (causing a fatal error).

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_473	534	The modification implemented a model enhancement as noted in Table C-2, but it also defined the spray plus vapor condensed on spray flow used in the Murphy turbulent heat transfer model used by the implicit form of the pressurizer model. An incorrect time-step size multiplier on the spray junction flow and the related condensation flow for the explicit pressurizer model was removed.
mod_474	517	Added logic to convert minor edit variables in the minor edit summary from British to SI units when the SI output option is requested.
mod_475	521	Corrected errors in the logic for the volume and junction input error processing to edit the correct trip numbers.
mod_476	536	Revised code used to define the flow in GPM (minor edit) for fill junctions
mod_478	538	Added logic to trap input for generalized data tables and fill tables that are too large for the RDATA array used to read them.
mod_480	542	Corrected an error when recalculation conductor elevations for the multinode pressurizer model.
mod_481		Added minor corrections for AIX compiler syntax and run-time errors.
	540	Corrected a restart error associated with mod_456 (pressurizer stratification model) where the flow length was moved from the static block to the dynamic block (tr_540) because it can change during the transient for the stratification model.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
	541	Corrected a restart error associated with mod_479 (accumulator model) where the gas internal energy was not saved on the restart file. The wall and surface heat transfer coefficient multipliers CWALL and CSURF were left in the static block so they can be changed during generalized restart while all other accumulator variables were moved to a dynamic block.
MOD004.7.1		This code version was distributed as a replacement for MOD004.7 so the default interregion heat transfer is the same as for MOD004.6 and earlier versions. A new input format will be required to use the new best-estimate interregion heat transfer, thus requiring a conscious change to the input by the user.
mod_482	548	A compiler-dependent syntax that resulted in a compilation error when using the Intel Composer 13.1 compiler (release used Ver. 11.1), was revised.
mod_483	544	Logic for flow regimes -1 and -6 (countercurrent flow predicted but one phase is not available) was revised and several related errors were corrected.
mod_484	545	The algorithm used to determine the space allocated for the input card deck was revised for very small input decks with only 3 data cards and no comment cards.
mod_485	550	The interfacial area was not initialized for options using a constant heat transfer coefficient and a heat transfer coefficient from a control system. The area initialization was moved so it is defined properly for all heat transfer options.
mod_486	553	Added logic needed to save the control system index in ingeom and then added logic to use it in tkandc. Also corrected polate error logic.

Table C-3 (Cont'd)

Code Version/ Modification No.	Trouble Report No.	Description of Change
mod_487	551	The mixed stream density was used for the drive junction rather than that for the drive junction. Flow sign terms were also added to the momentum terms to deactivate the jet pump mixing model when the drive flow reverses.
mod_488	554	Revised the logic for defining the heat transfer mode flag index.
mod_489	539 546 509	Skipped the logic for redefining the volume mass for two-phase conditions when the volume is single phase. Fixed input edit and added new error checks. Removed the zmorig and zvorig logic. Created a steady-state warning message to signal a changed enthalpy.
mod_490	552	Corrected the error number and revised the descriptive error information included in the error message.
mod_492	543	Revised the logic to read a user-specified heat transfer coefficient defined on a control block.
mod_493	547	Revised logic to zero out the prop array for each volume. Determined heat capacity from local quality for the local conditions model.
mod_494	555 556	Revised the logic to allow return to power and eliminate very small time step.
mod_495	549 558	Revised logic for calculating the volume average flow. Also added error messages and percent area change for conductor data actually being used.
mod_496	557	Removed extraneous input reflections from the restart output. Also fixed free form input Error 225 to correctly list unused cards.
mod_497	559	Corrected compatibility error on accumulator input. Removed unused variables.

APPENDIX D

FORTRAN 95 CONVERSION AND TESTING SUMMARY

The first Fortran 95 release of RETRAN-3D was intended to be MOD004.4f95. The Fortran 95 conversion and testing processes for MOD004.4f95 are discussed herein. As part of the testing for this version, two errors in the base MOD004.4 code were discovered. While the errors existed in the base code, all test cases ran to completion. Very slight difference in calculated results and time-step selection resulted in the errors causing the MOD004.4f95 to terminate for several of the validation cases. The errors had been observed previously for MOD004.4, but slight changes in time-step size or restarting the problem prior to the failure resulted in the problem running to completion. As such, they were difficult to consistently reproduce.

The identified errors were reported as Trouble Reports 407 and 408. Modifications correcting the errors were developed and tested for MOD004.4f95. Another modification that added several commonly needed minor edit variables and an option to generate a plot data file was developed. These modifications are discussed in Appendix C. They were added to MOD004.4f95 using CSA's code maintenance and development procedures to create MOD004.5f95, which was released to the RETRAN user community.

The first complete Fortran 95 version of RETRAN-3D was MOD004.4f95. It is functionally equivalent to its predecessor Fortran 66/77 code version MOD004.4. This Appendix discusses the processes used to convert the source code and create the base Fortran 95 code version and the implementation of the code modifications that both corrected errors and added code enhancements comprising the migration from MOD004.3 to MOD004.4. It also summarizes the verification and validation effort that demonstrates MOD004.4f95 is functionally equivalent to MOD004.4 (Fortran 66/77 version).

RETRAN-3D MOD004.3 was released to the user community in July 2007. It is a mixture of Fortran 66 and 77 code that uses a few Fortran 95 features. Source code for this version provided the base for the Fortran 95 conversion effort.

D.1 Source Code

As released, MOD004.3 was comprised of three separate SLIB77 portable source code libraries [1], one for the RETRAN thermal hydraulics, one for the environmental library and another for the three-dimensional kinetics. To begin the conversion effort, the source code for each of the three portable libraries was extracted and placed in separate directories. Minor changes were then made so the source could be maintained using a version control program, rather than SLIB77 whose use was often confusing and frustrating for new users.

The SLIB77 potable libraries contain *CALL [comdeck_name] directives that refer to common blocks of code that are inserted by SLIB77 as the source is extracted for use by a compiler. The

Fortran INCLUDE feature is similar in purpose to the *CALL, but can be inserted directly into the source code; thus eliminating the need to use SLIB77 to create source code for use by the compiler (SLIB77 has other uses). The source code was extracted using an option that left the *CALL directive in the source rather than inserting the content of the comdeck (common code). A script was written to replace the *CALL directives in the extracted source code with equivalent INCLUDE directives (include "comdeck_name.h"). Each SLIB77 comdeck was also extracted as a separate INCLUDE file with the name of the original comdeck and a ".h" extension. Figure D.1-1 illustrates this process used to eliminate the need to use SLIB77. The Fortran 77 source code was demonstrated to produce an executable that gave results identical to an executable using the source created by SLIB77.

The MOD004.3 source code is now maintained using the QVCS version control software [2] and is used for all current and future maintenance and development activities. Installation and testing of the QVCS source code confirmed that the results for the 16 standard sample problems were identical to those from the release form of MOD004.3.

D.2 Fortran 95 Conversion Process

The Fortran 95 conversion effort and ongoing maintenance and development were performed as parallel activities. Code modifications were developed to resolve reported problems and to also implement code enhancements. The conversion proceeded simultaneously, with both efforts using the source controlled by QVCS.

The conversion effort was divided into four separate phases as described below.

- Phase 1 – Feasibility Study and Prototyping,
- Phase 2 – Conversion of the Base RETRAN-3D Code,
- Phase 3 – Conversion of the One-Dimensional Kinetics Option, and
- Phase 4 – Conversion of the Three-Dimensional Kinetics Option.

For the purpose of discussion, preliminary work done as Phase 1 will be treated as part of Phase 2.

Phase 2 of the conversion effort started with the MOD004.3 source code for the RETRAN-3D source and the environmental library. Many of the environment library subroutines were removed because they are no longer used. For example, all subroutines associated with the FTB dynamic memory allocation feature were removed since they were replaced by Fortran 95 supported features. All bit masking, shifting, and Calcomp plotting features were also removed. Environmental library routines that were retained were included in the RETRAN-3D source code.

The plusFORT package of conversion programs developed by Polyhedron Software,[3] was used in the RETRAN-3D code modernization effort because of its capabilities, particularly the SPAG program, which was be used to convert the Fortran 77 source code to Fortran 95. The SPAG conversion process

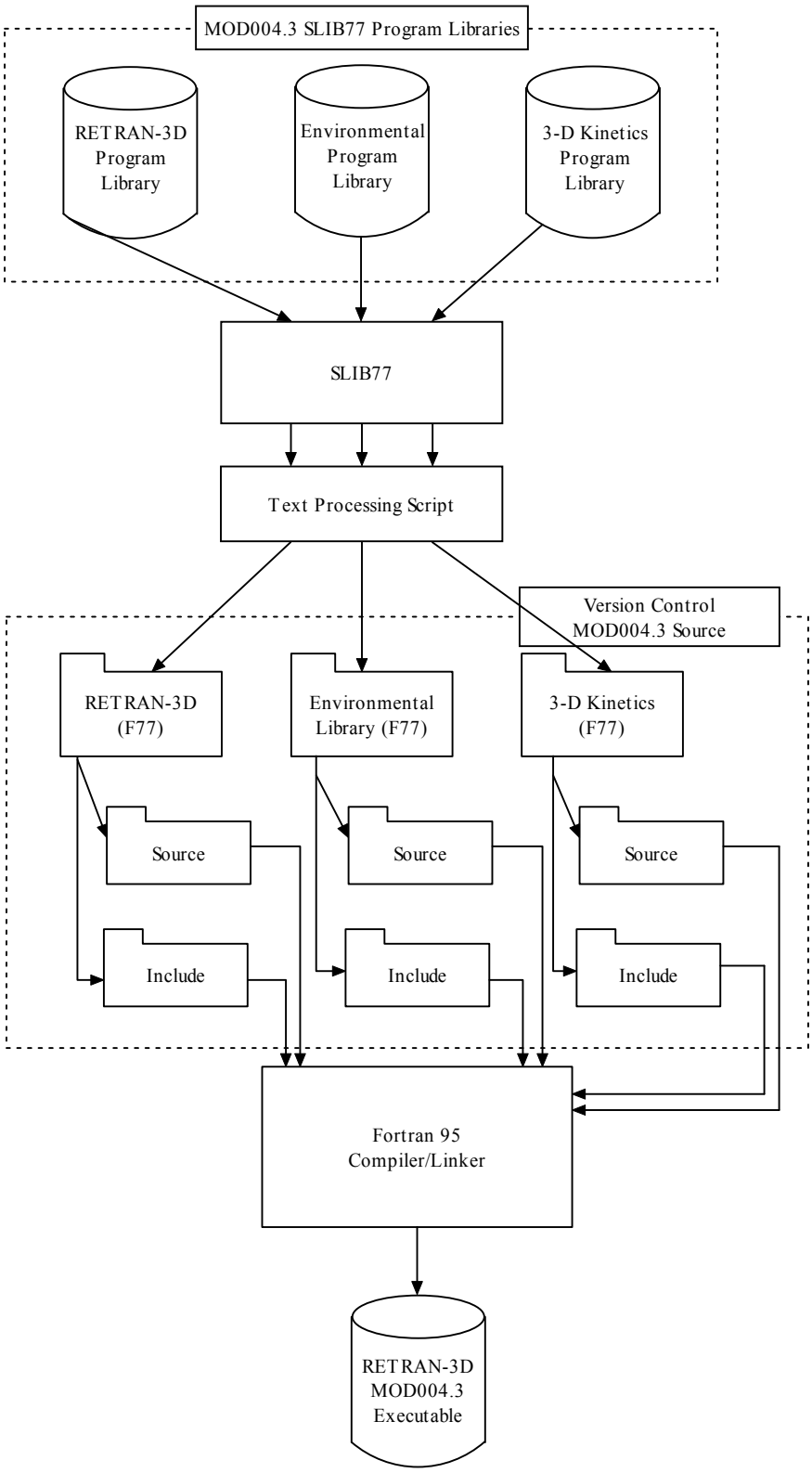


Figure D.1-1. Base Fortran 77 Source Code Preparation

- converted the fixed format Fortran 77 source code to free format source – up to 132 characters per line,
- converted the upper case coding to lower case, with the exception of text strings in FORMAT statements,
- converted Fortran 77 comments to Fortran 95 comments,
- converted Fortran 77 continuations to Fortran 95 continuations,
- indented the bodies of DO-loops and IF-blocks,
- replaced CONTINUE statements terminating a single DO loop with an END DO statement,
- restructured "spaghetti" code by replacing obsolete constructs with structured constructs, relocating or replicating code as necessary,
- eliminated most labels,
- revised type specifications to Fortran 95 syntax,
- inserted IMPLICIT NONE statements and generated explicit type statements for all variables and arrays used in a subroutine or function,
- converted common blocks to MODULES,
- eliminated obsolete constructs (assigned go to, computed go to, etc.), and
- eliminated unused code and variables.

Following generation of the translated source code by the SPAG program, additional revision(s) were made using script files and/or manual revisions. SPAG and the script files automated much of the conversion, which was more efficient than doing the work manually and also less prone to introducing errors.

During the Phase 2 conversion effort dummy subroutines were used to satisfy the external references for the one- kinetics and three-dimensional kinetics options. This disabled these options for the Phase 2 effort.

Much of the modification required to create the dynamically allocated data modules was automated using existing variable definitions found in subroutine RMAIN as comments; data type declarations found in the COMDECKS or INCLUDE files and a subroutine template. Some manual revisions were required to complete the data modules. Output editing and restart features were implemented using new methods, requiring a complete rewrite of the affected code. Additional details of the conversion effort are found in the code modernization design report.[4]

Testing for the Phase 2 conversion consisted of running the twelve of the sixteen standard sample problems, those that do not use the one-dimensional or three-dimensional kinetics options. Results for these sample problems were compared with those obtained using MOD004.3. They were found to be essentially identical.

Conversion of the one- and three-dimensional kinetics options was performed as Phase 3 and 4 tasks, respectively. The conversion efforts for these tasks were similar to those discussed for the RETRAN-3D source code as shown in Figure D.2-1. Phase 3 was begun following the successful testing for the Phase 2. Upon its completion, the TTQX1 sample problem and several other one-dimensional kinetics test cases were run on the converted code and MOD004.3. Again, the results were essentially identical.

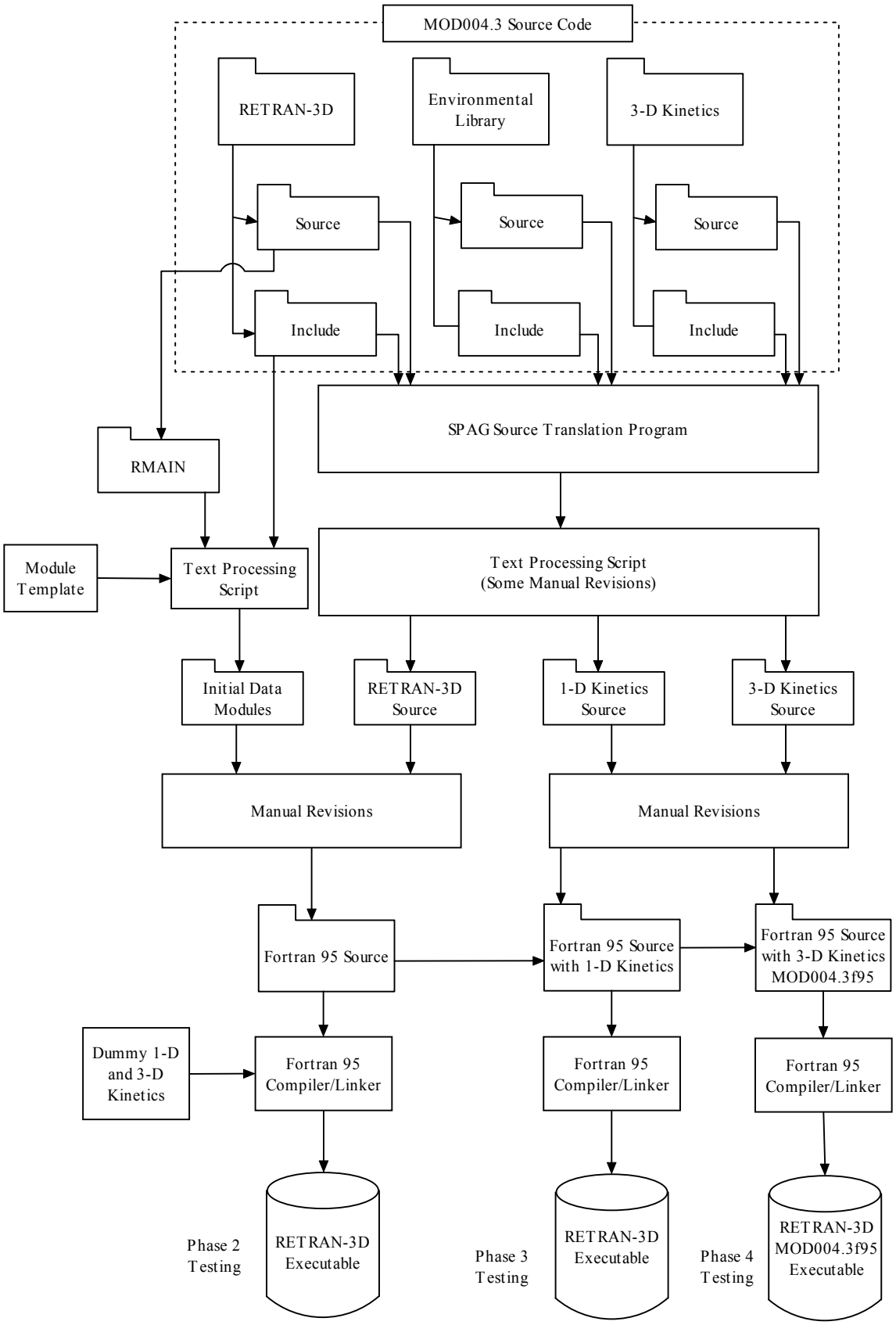


Figure D.2-1. Fortran 95 Source Code Conversion

Three-dimensional kinetics in MOD004.3 is comprised of two different solution options, the original ARROTTA option and the Purdue Numerical Method (PNM). A survey of RETRAN-3D users found that all who were using the three-dimensional kinetics option were using the PNM option. Consequently, it was the only option included in the Phase 4 conversion effort. The ARROTTA option was not converted at this time and is not available in the Fortran 95 code version. The Phase 4 conversion was performed following successful testing of the Phase 3 conversion. Upon completion of the Phase 4 conversion, the three three-dimensional kinetics sample problems and several other three-dimensional kinetics test cases were run on the converted code and MOD004.3. Again the results from the two code versions were essentially identical.

During the Phase 2, 3 and 4 conversion efforts, parallel efforts were ongoing to develop error corrections and code enhancements for MOD004.3. These modifications were prepared, tested, and validated in accordance with QA procedures. Each modification was then converted to Fortran 95 using SPAG and/or manually and added to the Fortran 95 code as illustrated in Figure D.2-2. Testing and validation for the Fortran 95 path was performed using the same QA procedures used for the Fortran 77 code path. The results for the validation test cases for the Fortran 95 and 77 code versions were then compared. The results were essentially identical. This conversion/testing process was repeated for each modification.

Upon completion of the conversion and testing for each modification, new code versions were created for both Fortran versions. The Fortran 77 code version was designated as MOD004.4 and the Fortran 95 code was designated as MOD004.4f95. The Fortran 95 code is functionally equivalent to the Fortran 77 code, only the implementation language has changed.

D.3 MOD004.4f95 Validation Process and Results

The Fortran 77 version of MOD004.4 satisfies all QA requirements for release to the user community; however, its intended purpose was to provide a means of validating the MOD004.4f95 code version. Figure D.3-1 illustrates the integral testing for the new code versions using a suite of test cases comprised of the 16 standard sample problems, 13 separate effects cases and 35 additional systems analysis cases, for a total of 64 validation cases. Many of the separate effects and system effects validation test cases were for analyses included in the Applications Manual – Volume 4. Others came from system models provided with error reports or models being used by RETRAN-3D users.

Each test case was run on both MOD004.4 and MOD004.4f95 and auxiliary files containing the minor edit variable histories were created. The minor edit variables included for the 16 standard sample problems were selected to represent the important parameters for the particular simulation. The minor edits included in the decks for the other cases were used as is.

Data Analysis Tools

The auxiliary files were then compared using the COMPARE2 program and an Excel autoplot macro. The MOD004.4 results are the standard or baseline data and the MOD004.4f95 results are the test data. The COMPARE2 program identifies when individual time point values in the test results (MOD004.4f95) differ from the standard results (MOD004.4) by a specified tolerance (0.001 or 0.1%). The number of differences reported provides an indication of whether or not

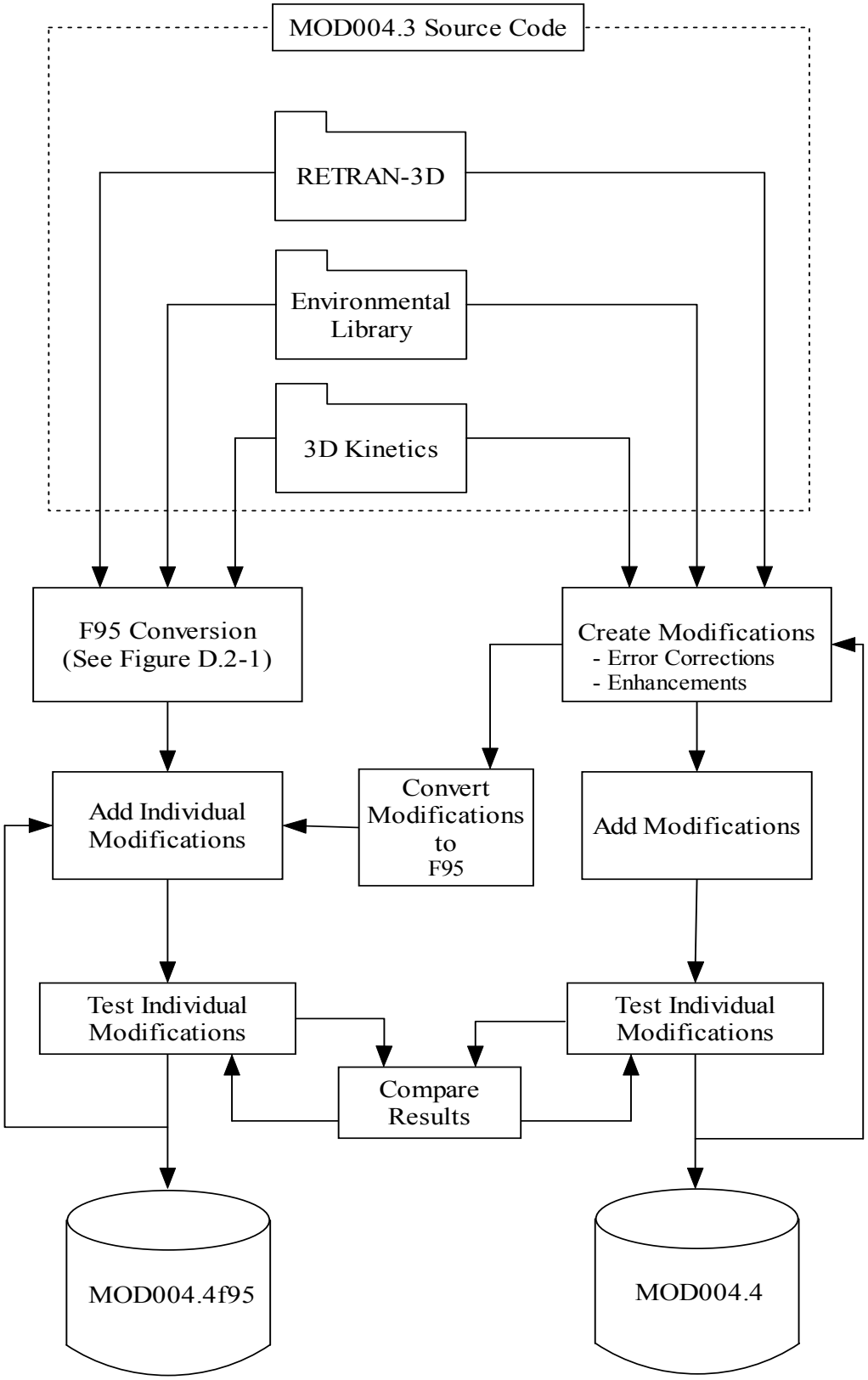


Figure D.2-2. Creating the MOD004.4f95 Code Version

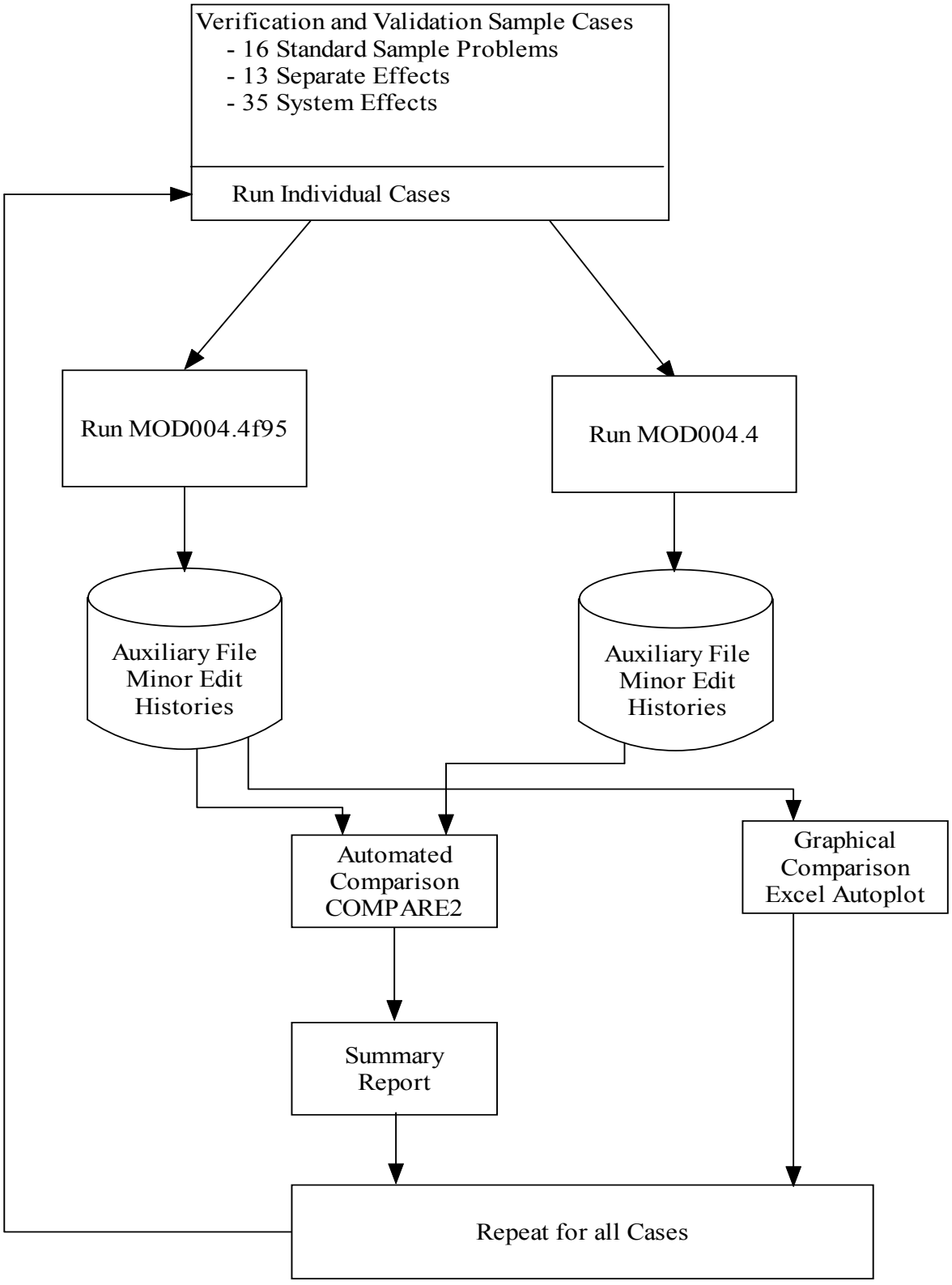


Figure D.3-1 Code Verification and Validation

the results are equivalent (0 difference indicates they are essentially equal). When differences are reported, the two results may be equivalent for practical purposes even though they are not numerically identical. To help make this determination, the Excel auto plot macro is used to generate overlay plots for each minor edit. This allows visual qualitative evaluation of the difference between the results for each minor edit variable.

Another custom Excel macro was used to analyze the difference between the two curves on each plot. It quantitatively evaluates the differences between the standard and test results on a point by point basis. If the normalized differences are less than a tolerance of 0.001, they are considered to be equivalent. The macro creates a new worksheet containing plots of the differences and the upper and lower tolerance bands. It and also computes the standard deviation of the test case results from those for the standard results. This is done for each variable. The standard deviations (SD) are then used to determine if the results are equivalent even though they are out or tolerance.

The computed SDs are based on the normalized standard and test data for each minor edit variable. Two different SDs are computed; the first only uses data points that are outside the tolerance limits or out of tolerance (OT). As such, the points within tolerance do not reduce the magnitude of the out of tolerance OT SDs. The second form of the SD is the normal form that uses all data points. If the two different SDs have significantly values, this indicates that the only part of the solution is out of tolerance. On the other hand, when both are similar in magnitude, this indicates that there is no segment (or region) that is significantly out of tolerance.

The maximum SDs and OT SDs will be for the minor edit variable with the largest value (for any given test case). The average SDs and OT SDs will be the average of the SD and OT SD for all minor edits for the given test case. When the maximum values differ significantly from the average values, this indicates that one or a few of the minor edits variables had differences that contributed to the larger SDs.

A maximum SD, below which results can be accepted without further review is needed. A review of the SDs indicated that 0.1 would be a reasonable cutoff. To justify this value, maximum SDs in the vicinity of 0.1 were evaluated by examining the quantitative and qualitative results for two validation cases where the maximum SD is near but larger than the tentative cutoff. The two cases are the lrhr standard sample problem and the Cofrentes MSIV Closure case.

The maximum SD for the lrhr standard sample problem validation case is for the Junction 10 flow and is 0.3399. The corresponding OT SD is 0.3607. Since both are about equal in magnitude, differences are expected for most of the duration of the simulation. This is confirmed by the results shown in Figure D.3-2 where there is a slight shift in time. For practical purposes, the results are equivalent. In spite of these differences, the MOD004.4f95 predicted flow is very similar to the MOD004.4 results. Given that the SD is three times the cutoff value, this indicates that a value of 0.1 is acceptable.

The next largest SD for the Lrhr case is 0.0273 and the corresponding OT SD is 0.0578. They are for the void fraction in Volume 15 as illustrated by Figure D.3-3, where there is no significant difference between the two curves shown. The SD for these results is about 1/3 of the cutoff limit.

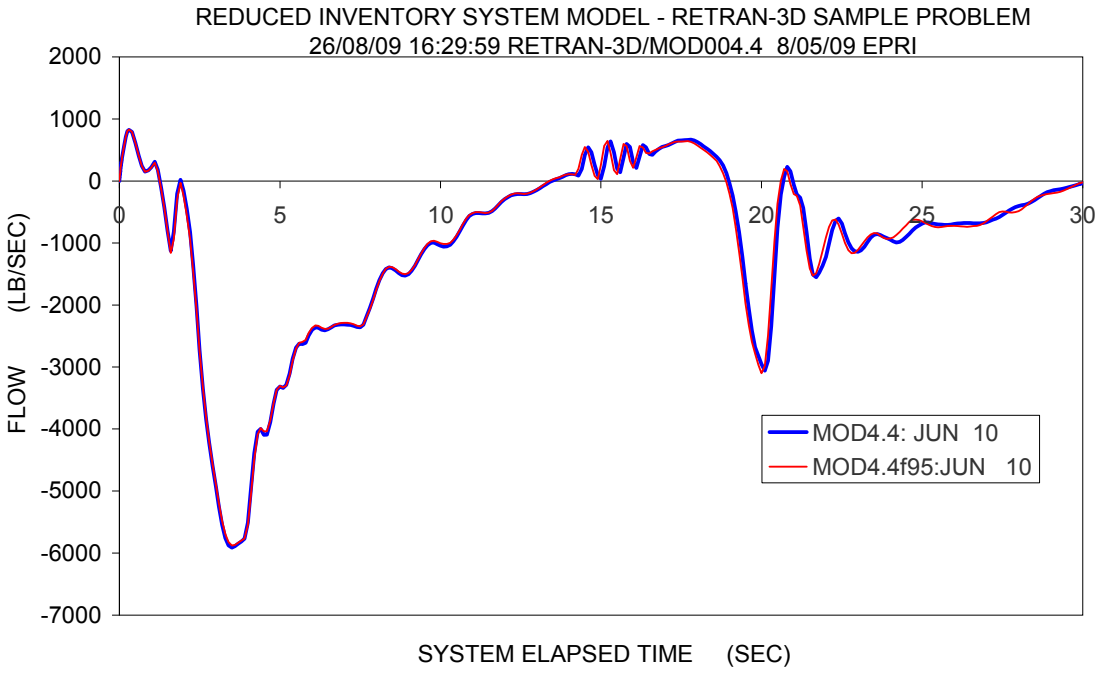


Figure D.3-2. Example SD of 0.3399 and OT SD of 0.3607

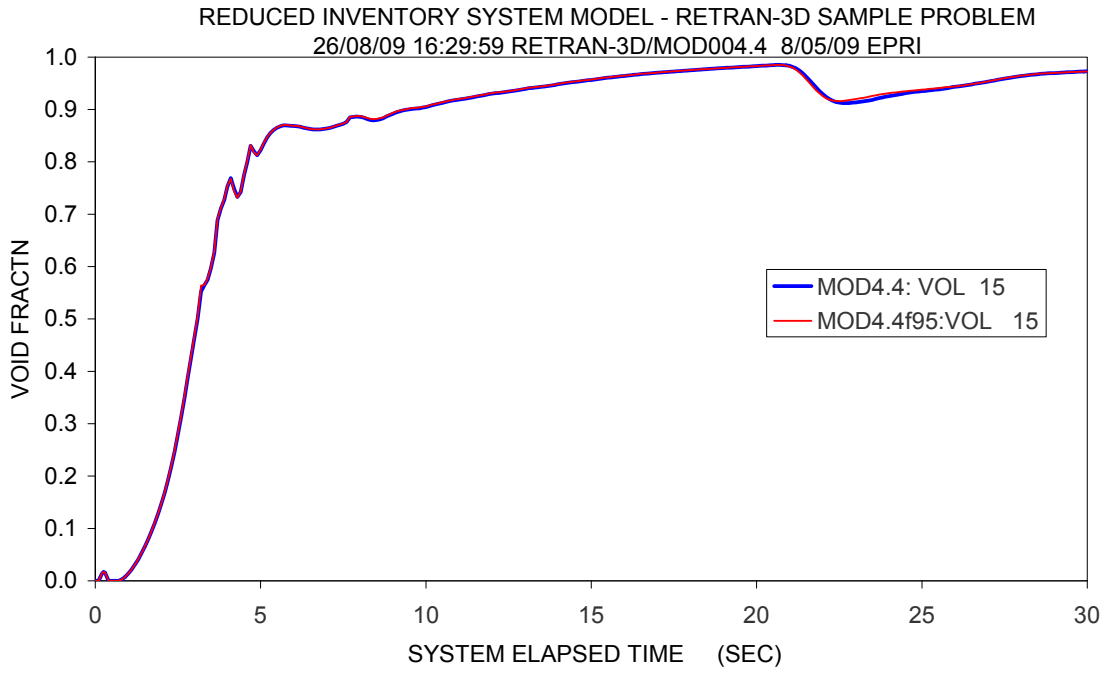


Figure D.3-3. Example SD of 0.0273 and OT SD of 0.0578

The maximum SD for the Cofrentes MSIV ATWS case is for the flow through Junction 460 and is 0.5522. The corresponding OT SD is 0.9369. The results used to obtain these SDs are shown in Figure D.3-4. The fact that the SD is smaller than the OT SD indicates that the observed differences occur over a limited region of time and not for the full duration of the simulation. For practical purposes, the results are comparable, even though the SD is five times larger than the tentative cutoff of 0.1.

Figure D.3-5 illustrates the minor edit variable with the next largest SD for the MSIV closure case. The SD is 0.0033 and the corresponding OT SD is 0.0118. As the figure illustrates, there is no perceptible difference between the two curves.

Based on the results shown for the Lrhr and Cofrentes MSIV closure validation cases, a cutoff of 0.1 appears to be acceptable. A detailed evaluation of the comparisons is not necessary for cases with a maximum SD less than or equal to 0.1.

D.3.1 Validation Analyses

Table D.3-1 lists the statistics for the 64 test cases used to validate RETRAN-3D MOD004.4f95. It includes the maximum and average values are for the OT SD and SD. This provides a better measure of how much the results differ from the standard results. Visual comparisons of the overlay plots for the various test cases have shown that SDs less than 0.1 indicate that for practical purposes the standard and test case results are equivalent. Each MOD004.4f95 validation case with a maximum SD greater than 0.1, is identified below. For these cases, the maximum SDs in Table D.3-1 are bold.

D.3.2 Lrhr Sample Problem

Given the results shown in Figures D.3-2 and D.3-3, the associated discussion and the fact that the average SD for the Lrhr validation case is 0.0419, the MOD004.5f95 results are equivalent to MOD004.4 and it is an acceptable replacement for MOD004.3.

D.3.3 Cofrentes MSIV Closure

Given the results shown in Figures D.3-4 and D.3.5, the associated discussion and the fact that the average SD for the Cofrentes MSIV closure is 0.0441, the MOD004.5f95 results are equivalent to MOD004.4 and it is an acceptable replacement for MOD004.3.

D.3.4 LOFT SB1

This case has several flows in which the SD is above 0.1. The highest SD of 1763.3 occurs in the flow through Junction 6. The flow rate through Junction 4 also has a high SD of 11.0392. Figure D.3-6 illustrates the oscillating natural circulation flow for Junction 6 and Figure D.3-7

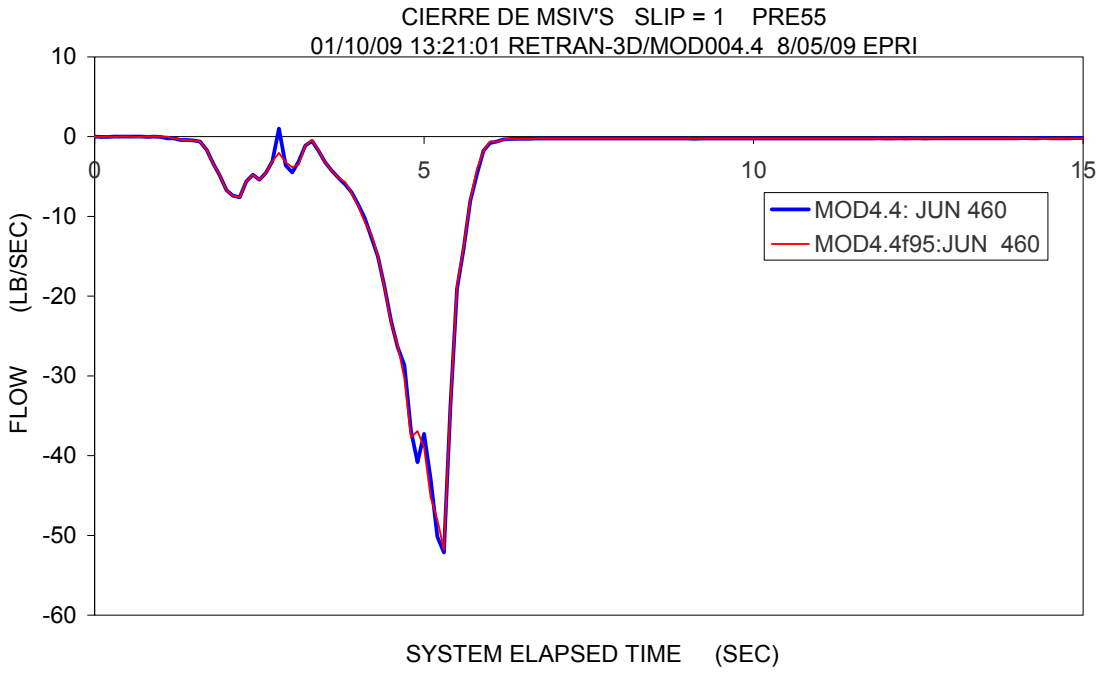


Figure D.3-4. Example SD of 0.5522 and OT SD of 0.9369

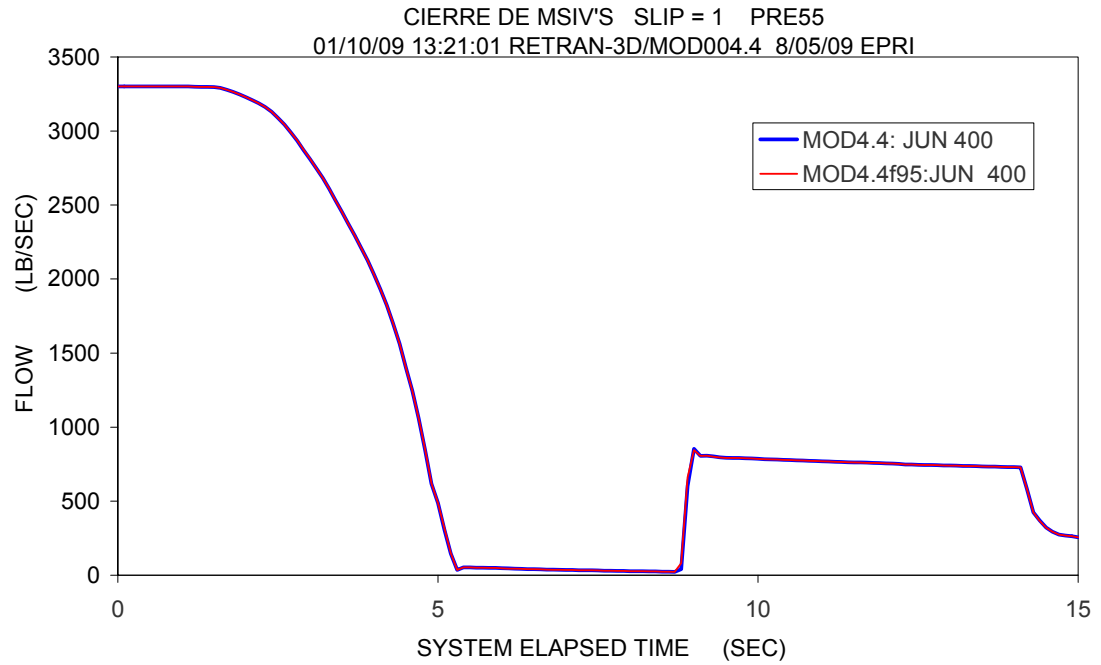


Figure D.3-5. Example SD of 0.0033 and OT SD of 0.0118

Table D.3-1

Statistics for RETRAN-3D MOD004.4f95 Test Cases

Problem Name	Average		Maximum	
	OT Standard Deviation	Standard Deviation	OT Standard Deviation	Standard Deviation
Standard Sample Problems				
sp1	0.0293	0.0168	0.1292	0.0775
Accum	0.0004	0.0001	0.0004	0.0002
sp5	0	0	0	0
Tlta	0	0	0	0
Ttwob	0.0067	0.0026	0.0172	0.0117
Ucrw	0.0008	0	0.0008	0.0001
fl2d	0	0	0	0
Turb	0	0	0	0
Atws	0.0030	0	0.0030	0
Pipe	0	0	0	0
Wovr	0.0010	0.0007	0.0016	0.0013
Lrhr	0.0619	0.0419	0.3607	**0.3399**
ttqx1	0.0058	0.0023	0.0154	0.0110
Pwr	0.0029	0	0.0040	0.0002
Slb	0.0007	0.0004	0.0012	0.0012
Bwr	0.0027	0	0.0075	0.0001
Separate Effects Tests				
Conduction Anal.	0	0	0	0
CSNI	0.0091	0	.0091	0.0001
Decay Heat	0	0	0	0
Fauske Crit. Flow	0	0	0	0
Ferrell McGee	0	0	0	0
LOFT L1-3 Accum.	0	0	0	0
NCG Press. Drop	0	0	0	0
ORNL THTF	0.0023	0.0008	0.0054	0.0021

Table D.3-1 (Cont'd)

Problem Name	Average		Maximum	
	OT Standard Deviation	**Standard Deviation**	**OT Standard Deviation**	**Standard Deviation**
Purdue Thrmsyphn	0	0	0	0
Round tube	0	0	0	0
Schrock Grossman	0	0	0	0
System Analysis Tests				
Almazar TT	0.0104	0.0002	0.0538	0.0050
ANO-2 TT	0.0016	0	0.0035	0
Hot Channel Fuel Model	0.0223	0.0004	0.4713	0.0178
Calvert Cliffs SLB	0.0046	0	0.0096	0.0013
Calloway BE	0	0	0	0
CE SLB	0.0037	0	0.0121	0.0001
Cofrentes FW Failure	0.0053	0.0007	0.0335	0.0146
Cofrentes HPCS inf.	0	0	0	0
Cofrentes MSIV Close.	0.0889	0.0441	0.9369	**0.5522**
Cofrentes MSIVC ATWS	0.0066	0.0009	0.0292	0.0041
Cofrentes 3D Kin Scram	0.0131	0.0078	0.0388	0.0236
Comanche Peak LR	0.0067	0.0012	0.0323	0.0143
SG Superheat	0.0173	0.0035	0.3412	0.0750
INER Hot Bundle	0	0	0	0
KOPEC	0.0012	0	0.0012	0.0011
KORI LLR	0.0145	0.0006	0.1428	0.0083
KORI Locked Rot	0	0	0	0

Table D.3-1 (Cont'd)

Problem Name	Average		Maximum	
	OT Standard Deviation	**Standard Deviation**	**OT Standard Deviation**	**Standard Deviation**
KORI LOP	0.0141	0.0007	0.1522	0.0101
LOFT SB1	51.0883	48.0799	1787.8943	**1763.3260**
LOFT SB2	0.0051	0.0037	0.0499	0.0453
Omega Blowdown	0.1105	0.1036	1.1495	**1.0797**
Oyster Creek FWC Malfcn.	0.0090	0.0006	0.0560	0.0082
PB Core Stability	0	0	0	0
Peach Bottom TT	0.0025	0	0.0045	0.0006
PECO PBTT	0.0042	0.0008	0.0194	0.0094
PI SGTR	0.0088	0.0015	0.1535	0.0855
River Bend 2 RCP Trip	0.0041	0.0001	0.0088	0.0005
SPERT 81	0.0009	0.0006	0.0020	0.0017
St. Lucie MSLB	0.0571	0.0100	06076	**0.2086**
Susquehanna FWCF	0.0557	0.0025	0.4341	0.0270
Susquehanna FWHF	0.0036	0	0.0073	0.0002
TMI LOFW	0.0027	0	0.0051	0.0008
Trojan LOF ATWS	0.0026	0	0.0066	0.0001
WCNOC Fire Scenario 3	0.1615	0.0473	6.8289	**4.4345**
Yonggwang 1 TT	0.0025	0	0.0042	0

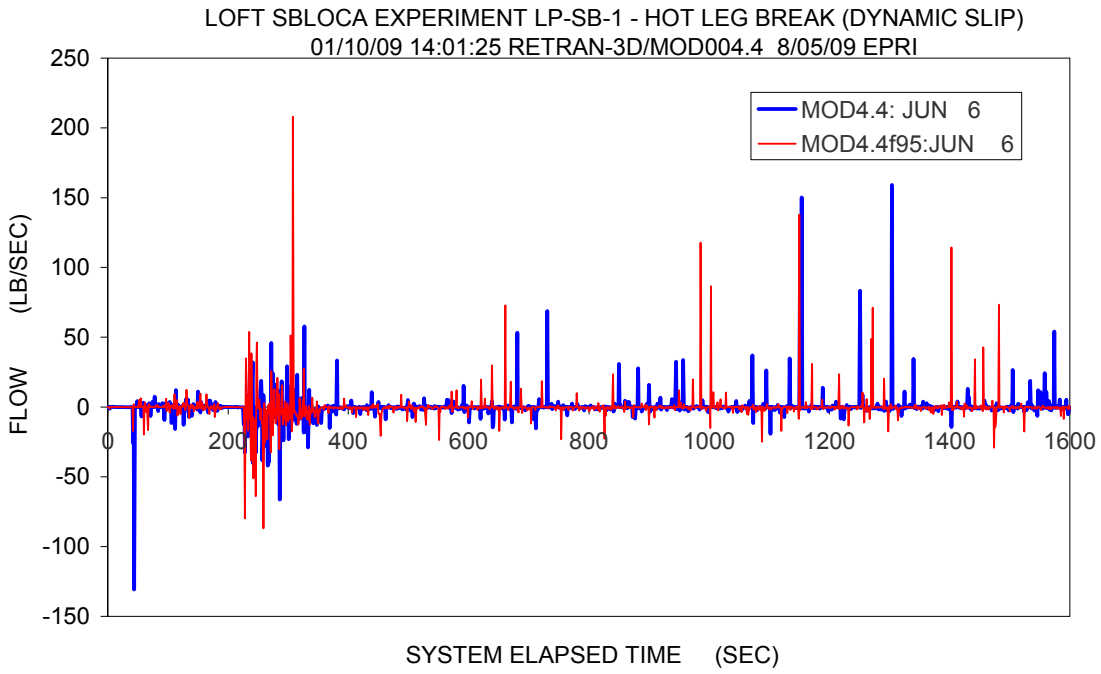


Figure D.3-6. Junction 6 Flow

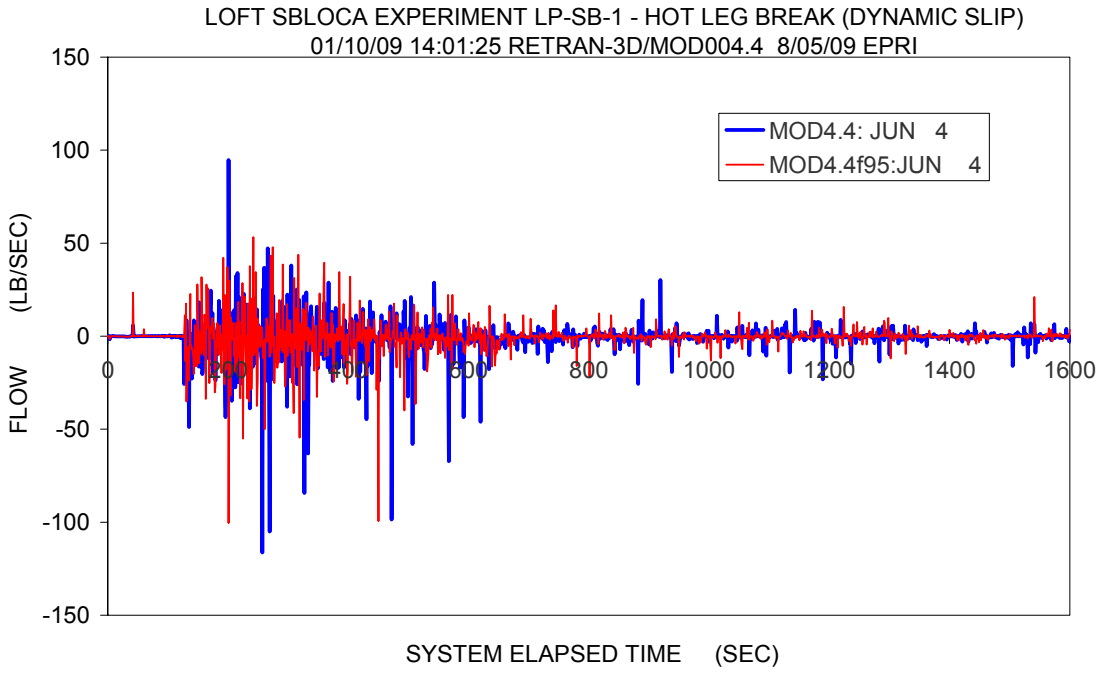


Figure D.3-7. Flow through Junction 4

shows the oscillating behavior for Junction 4. The high frequency oscillations are typical for natural circulation flows. As seen in the figures, the results are very similar even though the oscillations are slightly out of phase and the magnitudes differ slightly. The out of phase oscillations cause the SD to be extremely large, even though the overall behavior is similar.

Figure D.3-8 shows the behavior of the temperature in Volume 10. The two predictions are nearly indistinguishable, which agrees with the SD of 1.0e-5. For practical purposes, the results are equivalent and the MOD004.5f95 results are equivalent to MOD004.4 and it is an acceptable replacement for MOD004.3.

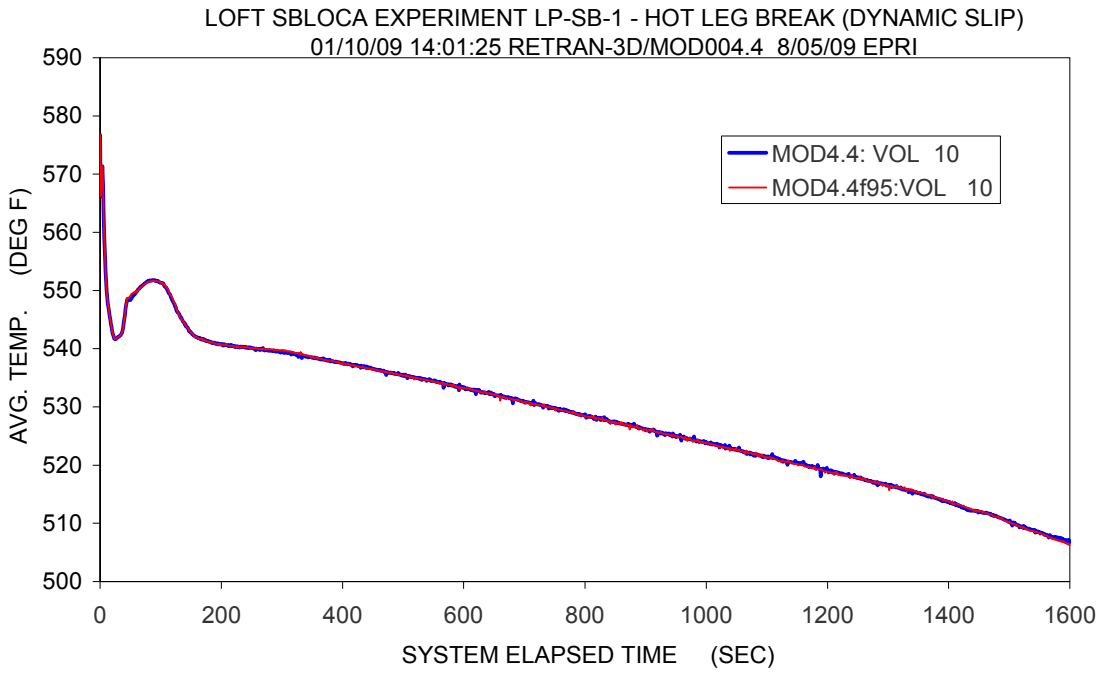


Figure D.3-8. Volume 10 Temperature

D.3.5 Omega Blowdown

The only minor edit variable that has an SD greater than 0.1 is the flow through Junction 1, which is 1.0797 and the corresponding OT SD is 1.1497. Figure D.3-9 shows high frequency oscillations as the flow approaches zero. As noted previously, when such oscillations occur and are even slightly out of phase, the SD becomes large because the individual data point errors are large. Figure D.3-10 shows the behavior of the pressure in Volume 34 where the SD is 0.0005. For practical purposes the results are equivalent and the MOD004.5f95 results are equivalent to MOD004.4 and it is an acceptable replacement for MOD004.3.

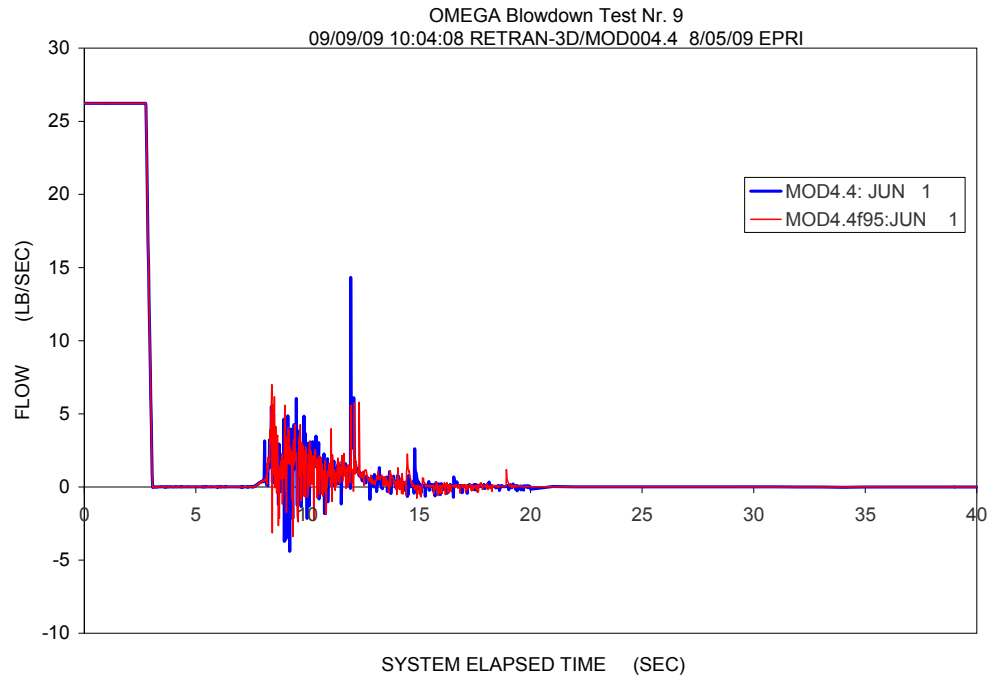


Figure D.3-9. Junction 1 Flow

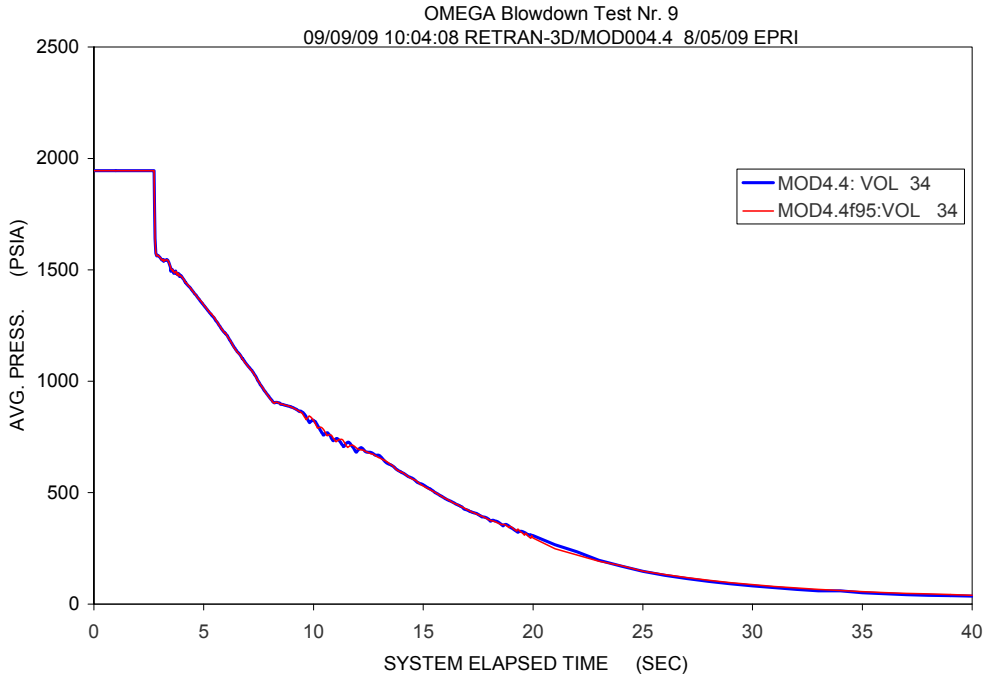


Figure D.3-10. Volume 34 Pressure

D.3.6 St. Lucie MSLB

For this case, the heat fluxes through a few of the conductors have SDs above 0.1. The highest occurs in Conductor 407 with a value of 0.2086. Figure D.3-11 shows that most of these differences occur after 65 seconds when the steam generator (SG) heat transfer should be due to condensation given the wall temperature and secondary-side volume conditions. Upon examination of the oscillations, it is observed that both MOD004.4 and MOD004.4f95 results oscillate, more so for MOD004.4. A detailed review of the results found that the oscillations occur when the heat transfer modes switch from condensation (larger negative heat flux) to forced convection. The conditions in both are such that they should remain in condensation heat transfer.

Trouble Report TR_408 identified an error in Mode 22 condensation heat transfer that was corrected by modification mod_354, which was included in the MOD004.5f95 code version. Figure D.3-12 shows the results of Figure D.3-11 with the MOD004.5f95 results included. The MOD004.5f95 results show that the heat transfer correctly remains in the higher negative heat flux condensation mode (22) rather than switching between forced convection and condensation.

The average SD for all the MSLB results is 0.0100. Give the MOD004.4f95 statistics, which are quite good, the results are equivalent to MOD004.4 and it is an acceptable replacement for MOD004.3.

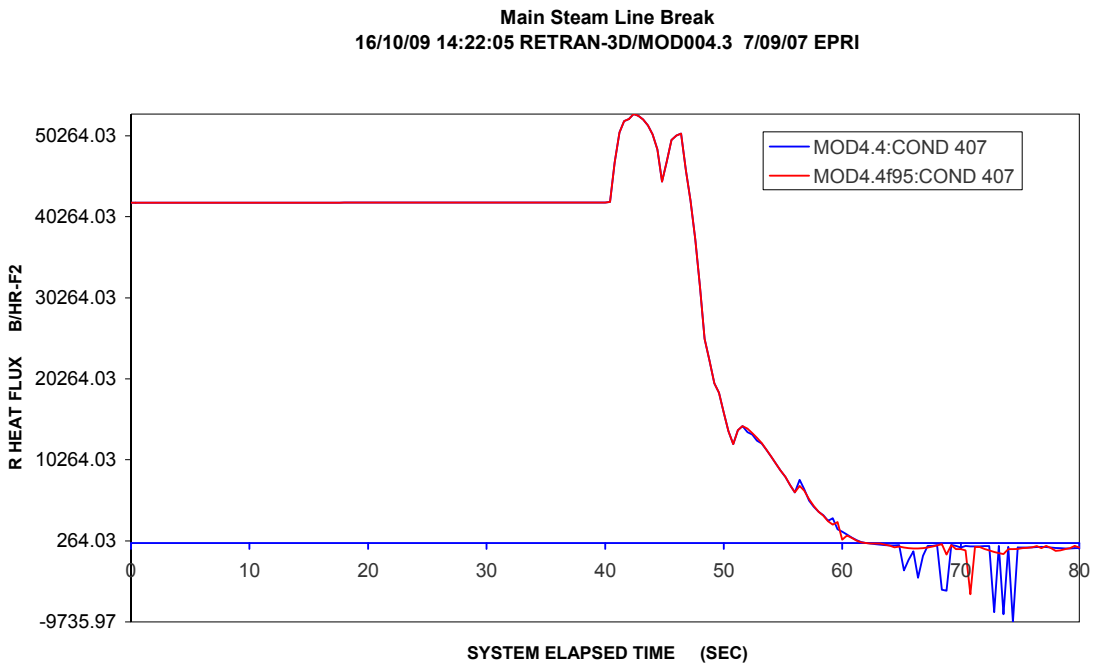


Figure D.3-11. SG Wall Heat Flux

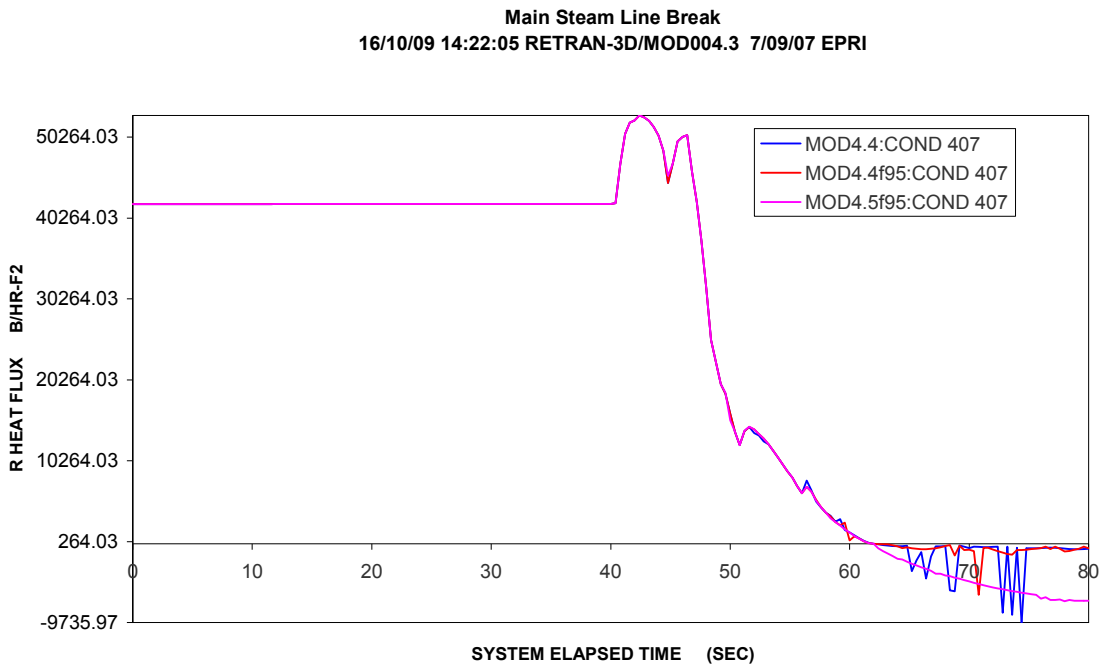


Figure D.3-12. Improved SG Wall Heat Flux

D.3.7 WCNOC Fire Scenario 3

Most all of the SDs in this case are very small except for the flow through Junction 363. It has a SD of 4.4345. If scaled to show the initial flow, the differences are not discernable. Figure D.3-13 has been scaled to show the differences, which are quite small and occur as the flow approaches zero, near the end of the transient when the two solutions become out of phase. These differences are negligible for practical purposes. The average SD for all minor edit variables is two orders of magnitude smaller than the maximum value, indicating that the results for the two code versions are equivalent for this simulation. As such, MOD004.4f95 is an acceptable replacement for MOD004.3.

D.4 Documentation

The documentation included with the RETRAN-3D MOD004.4f95 (and subsequent) transmittal(s) was prepared in accordance with QA procedures. Revisions to Volume 1 – Theory and Numerics, Volume 3 – User's Manual and Volume 4 – Assessment Manual were made to correct errors and add discussion for new features or results. Volume 2 – Programmer's Manual was rewritten to describe the code's implementation using Fortran 95 and to include new maintenance and installation instructions.

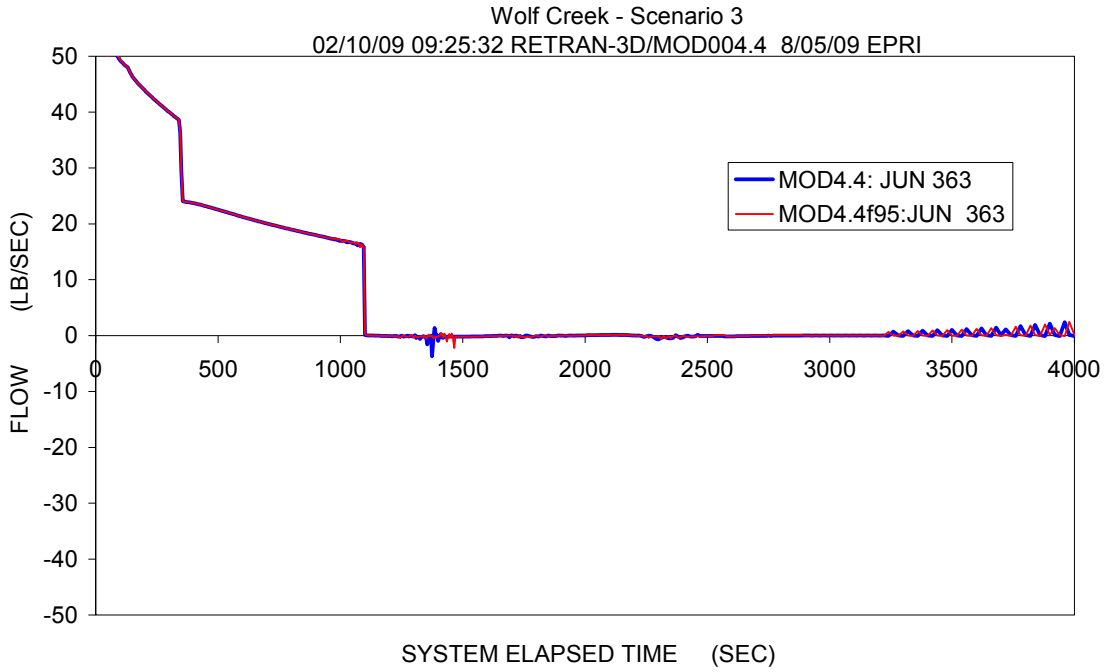


Figure D.3-13. Junction 363 Flow

The documents are released as Revisions 7.0. They are based on the MS Word master documents that were converted from the Word Perfect master documents previously used. The documents were converted by Zandar Corporation using their Tag Write conversions software.[5]

Products of the conversion effort were the converted MS Word documents and log files that recorded changes and identified possible problem areas. Both the logs and converted documents were reviewed page by page to ensure that the new master documents are accurate representations of the old master documents. Manual corrections were made as needed for accuracy and editorial purposes.

D.5 References

1. Paulsen, M. P., et al., "RETRAN-3D – A Program for Transient Thermal-Hydraulic Analysis of Complex Fluid Flow Systems", Volume 2 – Programmer's Manual, EPRI NP-7450(A), Revision 6.3, July 2007.

2. Quma Version Control System – Users Manual, Quma Software, Inc., 2005, http://www.qumasoft.com.

3. plusFORT Reference Manual, Revision E, Polyhedron Software Limited, 2002, http://www.polyhedron.com.

4. Software Design Document for RETRAN-3D Code Modernization, RETRAN User Group, CSA, Inc., RVUG-R3D-SDD-004, Revision 0, September 2007.

5. TagWrite, Zandar Corporation, http://www.tagwrite.com.

APPENDIX E

THE get_R3D_plot_vars UTILITY PROGRAM

RETRAN-3D will optionally create a data file that contains solution results for a predetermined set of RETRAN-3D variables (see Programmer's Manual - Volume 2 – Sections IV.2-8 and IV.2-11). This data can then be used by other applications to create X-Y plots or other graphical representations of problem solutions. A utility program, get_R3D_plot_vars has been written to allow data for selected plot variables (minor edit variables) to be extracted from the plot file and written to a new file according to a user-defined format.

E.1 RETRAN-3D Plot File

The plot file is similar to the auxiliary file (with header) in that it contains problem-specific documentation of the run date and time, problem name, and code version. It also contains the minor edit descriptions and units that can be used for default plot labels and minor edit request flag and region number pairs that can be used to locate the data for a given minor edit. Following the header information, separate records are written for each time point for which a minor edit is written. The record contains the values for each plot variable.

A subset of all minor edit variables comprise the plot variables included in the plot file. They are identified in the User's Manual, Volume 3, Section IV-4.0, as the italicized minor edit variables. Plot variables for all valid regions for the current problem, e.g., volumes, junctions, etc. are included in the plot file. These minor edit variables are supplemented by user-specified minor edit pairs that are unique, i.e., if a requested minor edit is in the default or fixed list of minor edits, it is not duplicated in the plot file; however, if it is not in the predefined list, it is added, but only for the given region. Since the plot file can be very large, it contains single precision data.

The structure and content of the RETRAN-3D plot file is discussed in the Programmer's Manual - Volume 2 - Section IV-2.8.

E.2 get_R3D_plot_vars Plot Data Extraction Utility

A utility program, get_R3D_plot_vars, has been written to allow information to be retrieved from a RETRAN-3D plot file. This utility reads a R3D_PLOT file and extracts time-dependent data for a list of plot variables. The data is written to file PLOT_VAR according to a user-defined format (Fortran). An option is available to create a PLOT_VAR file with the header information similar to the RETRAN-3D auxiliary file (TAPE60). This allows get_R3D_plot_vars to be used to create an auxiliary file that can be used in the same manner as auxiliary files created directly by RETRAN-3D. An example is to use the PLOT_VAR file to generate EXCEL plots using existing applications or procedures that are based on using an

auxiliary file to provide the data to be plotted. Input options also allow the PLOT_VAR file to be created as either an ASCII or binary file (ASCII will be most common).

For a description of the header information that may be included in the PLOT_VAR file, refer to Section IV.4.1 of the User's Manual – Volume 3.

The get_R3D_plot_vars program also provides a template that can be used in other applications for extracting plot file data.

E.3 How to Use get_R3D_plot_vars

The get_R3D_plot_vars utility program is written in Fortran 90 and can be installed using a Fortran 90 compiler. The names for the various compilers vary from platform to platform and compiler vendor. Modules kind_specs and read_R3D_plot_file, and program get_R3D_plot_vars reside in the get_R3D_plot_vars utility subdirectory of the RETRAN-3D source and will be found in files named kind_specs.f90, m_read_R3D_plot_file.f90, and get_R3D_plot_vars.f90. They must be copied into a directory where the program get_R3D_plot_vars will be installed.

In order to satisfy module dependencies, file m_kind_specs.f90 should be compiled first, file m_read_R3D_plot_file.f90 second, and finally get_R3D_plot_vars.f90. An example compile statement follows

```
f90 -o get_R3D_plot_vars.x  m_kind_specs.f90 m_read_R3D_plot_file.f90
get_R3D_plot_vars.f90  #no wrap
```

It would create an executable file named get_R3D_plot_vars.x.

E.4 Input Requirements

The get_R3D_plot_vars utility program requires two input files, the RETRAN-3D plot file which must be named R3D_PLOT. The second input file VAR_LIST contains a list of plot variables to be extracted from the R3D_PLOT file. Results for each minor edit variable will be extracted and written to a new file named PLOT_VARS. All time points in the R3D_PLOT file will be included in the PLOT_VARS output file.

The VAR_LIST file contains the three record types.

Format	Variable	Description
3i4	num_vars	number of plot variables to be extracted from the R3D_PLOT file and written to file PLOT_VARS
	plt_type	plot file type (R3D_PLOT) = 1 - ASCII = 2 - binary

Format	Variable	Description
	plt_ihdr	format and header information for file PLOT_VARS = 0 - ASCII file with no header = 1 - ASCII file with header = 2 - Binary file with no header = 3 - Binary file with header
a80	fmt	Fortran format used to write the plot variable values to file VAR_LIST. For example, a format string of '(1p,10e13.5)' will give an output file with a maximum of 10 13-character data items in each record. Multiple records would be required and if num_vars is greater than 10, with the concluding record containing mod(num_vars,10) data items. Note: Longer records (50 to 100 data items) are more efficient if a large number of variables are being written.
15(a4,i5,1x)	var_flg	plot variable request pairs (the variable flag must be upper case)
	var_reg	region number (num_vars pairs must be provided - up to 15 pairs can be specified in one record - multiple records may be required)

File PLOT_VARS will contain the plot variables in the order specified in file VAR_LIST. One record will be written for each record (time) in the R3D_PLOT file.

The plot file, R3D_PLOT, can be either binary or ASCII. Binary files are approximately one-third the size of the equivalent ASCII file. The file type is specified in the RETRAN.cfg file discussed in the RETRAN-3D Programmer's Manual - Volume 2 - Section IV.2-11.